

## [Dreamhack CTF Season 6 Round #7]: Wasm rev for beginners

이 문제를 고른 이유는 4개의 문제들 중에서 그나마 가장 난이도가 쉬워보였기 때문이다. 비록 리버싱은 해본 적이 없지만 문제 이름이 'for beginners' 이었기 때문에 시도해보려 하였다.

Input

Put your flag here

Check!

index.html을 열면 플래그를 입력할 수 있는 텍스트박스과 이를 검증할 수 있는 버튼이 보인다.

다음으로 challenge.js 파일을 열어보았다.

```
let wasm;

const cachedTextDecoder = (typeof TextDecoder !== 'undefined' ? new TextDecoder('utf-8', { ignoreBOM: true, fatal: true }) : { decode: () => { throw Error('TextDecoder not available') } } );

if (typeof TextDecoder !== 'undefined') { cachedTextDecoder.decode(); };

let cachedUint8Memory0 = null;

function getUint8Memory0() {
    if (cachedUint8Memory0 === null || cachedUint8Memory0.byteLength === 0) {
        cachedUint8Memory0 = new Uint8Array(wasm.memory.buffer);
    }
    return cachedUint8Memory0;
}

function getStringFromWasm0(ptr, len) {
    ptr = ptr >>> 0;
    return cachedTextDecoder.decode(getUint8Memory0().subarray(ptr, ptr + len));
}

let WASM_VECTOR_LEN = 0;

const cachedTextEncoder = (typeof TextEncoder !== 'undefined' ? new TextEncoder('utf-8') : { encode: () => { throw Error('TextEncoder not available') } } );

const encodeString = (typeof cachedTextEncoder.encodeInto === 'function'
    ? function (arg, view) {
        return cachedTextEncoder.encodeInto(arg, view);
    }
    : function (arg, view) {
        const buf = cachedTextEncoder.encode(arg);
        view.set(buf);
        return {
            read: arg.length,
            written: buf.length
        };
    });
```

```

function passStringToWasm0(arg, malloc, realloc) {

  if (realloc === undefined) {
    const buf = cachedTextEncoder.encode(arg);
    const ptr = malloc(buf.length, 1) >>> 0;
    getUint8Memory0().subarray(ptr, ptr + buf.length).set(buf);
    WASM_VECTOR_LEN = buf.length;
    return ptr;
  }

  let len = arg.length;
  let ptr = malloc(len, 1) >>> 0;

  const mem = getUint8Memory0();

  let offset = 0;

  for (; offset < len; offset++) {
    const code = arg.charCodeAt(offset);
    if (code > 0xFF) break;
    mem[ptr + offset] = code;
  }

  if (offset !== len) {
    if (offset !== 0) {
      arg = arg.slice(offset);
    }
    ptr = realloc(ptr, len, len - offset + arg.length * 3, 1) >>> 0;
    const view = getUint8Memory0().subarray(ptr + offset, ptr + len);
    const ret = encodeString(arg, view);

    offset += ret.written;
    ptr = realloc(ptr, len, offset, 1) >>> 0;
  }

  WASM_VECTOR_LEN = offset;
  return ptr;
}

```

```

* @param {string} flag
*/
export function check(flag) {
  const ptr0 = passStringToWasm0(flag, wasm.__wbindgen_malloc, wasm.__wbindgen_realloc);
  const len0 = WASM_VECTOR_LEN;
  wasm.check(ptr0, len0);
}

async function __wbg_load(module, imports) {
  if (typeof Response === 'function' && module instanceof Response) {
    if (typeof WebAssembly.instantiateStreaming === 'function') {
      try {
        return await WebAssembly.instantiateStreaming(module, imports);
      } catch (e) {
        if (module.headers.get('Content-Type') !== 'application/wasm') {
          console.warn("'WebAssembly.instantiateStreaming' failed because your server does not serve" +
            "application/wasm files. Try configuring the server with 'Content-Type: application/wasm'.");
        } else {
          throw e;
        }
      }
    }
  }

  const bytes = await module.arrayBuffer();
  return await WebAssembly.instantiate(bytes, imports);
} else {
  const instance = await WebAssembly.instantiate(module, imports);

  if (instance instanceof WebAssembly.Instance) {
    return { instance, module };
  } else {
    return instance;
  }
}
}

```

```

function __wbg_get_imports() {
    const imports = {};
    imports.wbg = {};
    imports.wbg.__wbg_alert_9606c3146601d524 = function(arg0, arg1) {
        alert(getStringFromWasm0(arg0, arg1));
    };

    return imports;
}

function __wbg_init_memory(imports, maybe_memory) {

}

function __wbg_finalize_init(instance, module) {
    wasm = instance.exports;
    __wbg_init.__wbindgen_wasm_module = module;
    cachedUint8Memory0 = null;

    return wasm;
}

function initSync(module) {
    if (wasm !== undefined) return wasm;

    const imports = __wbg_get_imports();

    __wbg_init_memory(imports);

    if (!(module instanceof WebAssembly.Module)) {
        module = new WebAssembly.Module(module);
    }

    const instance = new WebAssembly.Instance(module, imports);

    return __wbg_finalize_init(instance, module);
}

```

```

async function __wbg_init(input) {
    if (wasm !== undefined) return wasm;

    if (typeof input === 'undefined') {
        input = new URL('challenge_bg.wasm', import.meta.url);
    }
    const imports = __wbg_get_imports();

    if (typeof input === 'string' || (typeof Request === 'function' && input instanceof Request) || (typeof URL
        input = fetch(input);
    }

    __wbg_init_memory(imports);

    const { instance, module } = await __wbg_load(await input, imports);

    return __wbg_finalize_init(instance, module);
}

export { initSync }
export default __wbg_init;

```

이 코드는 WebAssembly(WASM) 모듈을 브라우저에서 로드하고 사용하기 위한 JavaScript 코드이다. WASM 파일은 웹 브라우저에서 실행되는 바이너리 형식의 파일으로, 브라우저에서 고성능으로 실행될 수 있도록 설계된 저수준의 이진 포맷을 제공하는 기술이다.

이 코드는 JavaScript와 WebAssembly 사이에서 문자열을 변환하고 메모리를 관리하는 역할을 합니다. 코드를 분해해보면 다음과 같다.

## 1. 텍스트 인코딩 및 디코딩

``cachedTextDecoder``와 ``cachedTextEncoder``는 문자열 인코딩과 디코딩을 처리한다.

- ``TextDecoder``는 WebAssembly 메모리에서 바이트 배열을 문자열로 변환한다.
- ``TextEncoder``는 문자열을 UTF-8 바이트 배열로 변환한다.
- ``encodeString``과 ``decode``는 WebAssembly에서 전달된 데이터를 JavaScript에서 사용할 수 있도록 변환하는 함수이다.

## 2. 메모리 관리

``getUint8Memory0`` 함수는 WebAssembly 메모리 버퍼를 Uint8Array 형식으로 가져온다.

- WASM 모듈의 메모리 접근을 위한 캐시를 관리한다.

## 3. 문자열 처리

- ``getStringFromWasm0(ptr, len)``는 WASM에서 전달된 포인터와 길이를 사용해 해당 메모리 영역에서 문자열을 읽는다.
- ``passStringToWasm0(arg, malloc, realloc)``는 JavaScript에서 문자열을 WASM 모듈로 전달할 때 사용된다.
- 문자열을 UTF-8로 인코딩한 후, WASM 메모리의 특정 위치에 저장한다.

## 4. WASM 모듈 로드

- ``wbg_load(module, imports)``는 WebAssembly 모듈을 로드하고 인스턴스화하는 함수이다.
- ``WebAssembly.instantiateStreaming`` 함수가 사용 가능하면 이 기능을 사용해 모듈을 스트리밍 방식으로 로드하고, 그렇지 않으면 ``arrayBuffer()``로 모듈을 로드한다.

- ``wbg_get_imports()``는 WASM 모듈이 사용할 JavaScript 함수들을 정의한다. 이 코드에서는 ``alert()`` 함수가 WASM에서 호출될 수 있도록 정의되어 있다.

## 5. 초기화 및 인스턴스화

- ``wbg_finalize_init(instance, module)``는 WASM 모듈을 초기화하고, WASM 인스턴스의 메모리를 초기화하는 역할을 한다.
- ``initSync(module)``는 동기적으로 WASM 모듈을 초기화한다.
- ``wbg_init(input)``는 비동기 방식으로 WebAssembly 모듈을 초기화하는 함수이다. 이 함수는 ``input``으로 전달된 WebAssembly 바이너리를 로드하고 인스턴스화한 후 반환한다.

## 6. WASM과의 상호작용

``check(flag)`` 함수는 JavaScript 문자열을 WASM으로 전달하고, 해당 문자열을 WASM에서 처리하게 한다. 이 함수는 문자열을 WASM 메모리로 복사한 후 ``wasm.check`` 함수를 호출한다.

이렇게 일단 열 수 있는 js파일의 코드를 먼저 분석해 보았지만 문제 해결에 중요할 것 같은 `challenge_bg.wasm` 파일은 열 수도 없었다. 리버싱에 대해 좀 더 공부하고 시도해보아야 할 것 같다.