

Node inspect를 이용한 윈도우 자바스크립트 난독화 해제

작성자: 고남현

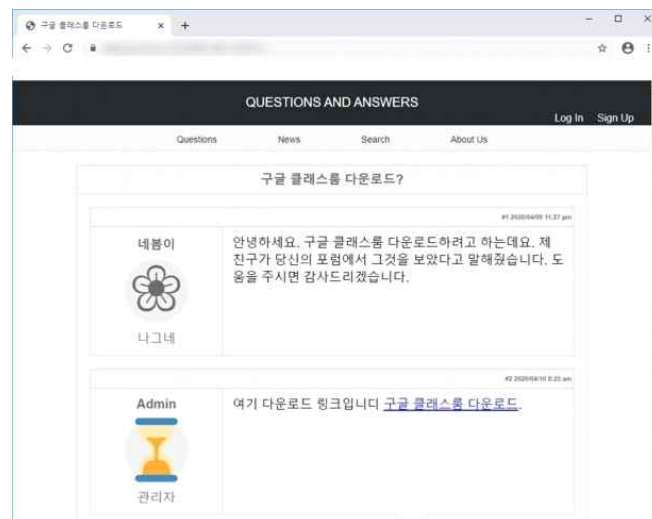
작성일: 2020-10-15

1. 시작하기 전에

- * ConEmu: <https://conemu.github.io/>
- * Node.js 다운로드: <https://nodejs.org/ko/download/>
- * Chrome 다운로드: <https://www.google.com/intl/ko/chrome/>
- * 실습파일: <https://github.com/gnh1201/malware-analysis>
- * 네뵐이 웹 사이트: <https://github.com/gnh1201/nebomi>

2. 개요

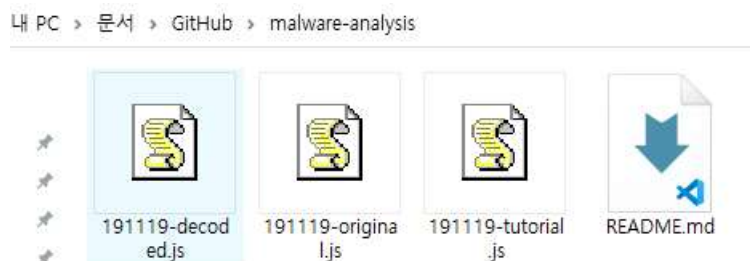
갠드크랩(GandCrab) 계열 랜섬웨어 유포에 사용된 스크립트 유형이다. '네뵐이'라는 가짜 다운로드 웹 사이트를 통해 주로 유포되었다.



[그림 1] 네뵐이 웹 사이트

2. 실습파일 확인

실습파일을 열면 original, tutorial, decoded 파일을 확인할 수 있다. 'original'은 암호화된 원본, 'tutorial'은 복호화 중간 과정, 'decoded'는 완전히 복호화된 파일이다.



[그림 2] 실습 파일

3. Node.js 설치

Node.js 다운로드 웹 사이트 (<https://nodejs.org/ko/download/>)에 들어가면 유형에 맞는 다운로드 옵션을 제공한다. 다운로드 후 설치를 진행한다.

다운로드

최신 LTS 버전: 12.19.0 (includes npm 6.14.8)

플랫폼에 맞게 미리 빌드된 Node.js 인스톨러나 소스코드를 다운받아서 바로 개발을 시작하세요.

LTS
대다수 사용자에게 추천

Windows Installer
node-v12.19.0-win94.msi

Windows Binary (.zip)
macOS Installer (.pkg)
macOS Binary (.tar.gz)
Linux Binaries (x64)
Linux Binaries (ARM)
Source Code

현재 버전
최신 기능

macOS Installer
node-v12.19.0.pkg

Source Code
node-v12.19.0.tar.gz

32-bit	64-bit
32-bit	64-bit
64-bit	
64-bit	
64-bit	
ARMv7	ARMv8
node-v12.19.0.tar.gz	

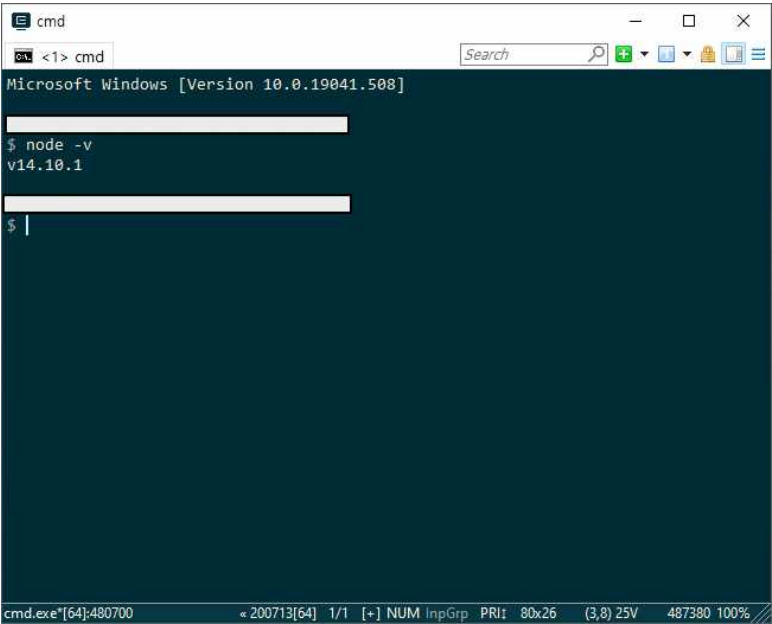
그 밖의 플랫폼

SmartOS Binaries	64-bit
Docker Image	Official Node.js Docker Image
Linux on Power LE Systems	64-bit
Linux on System z	64-bit
AIX on Power Systems	64-bit

[그림 3] Node.js 다운로드 웹 사이트

4. Node.js 버전 확인

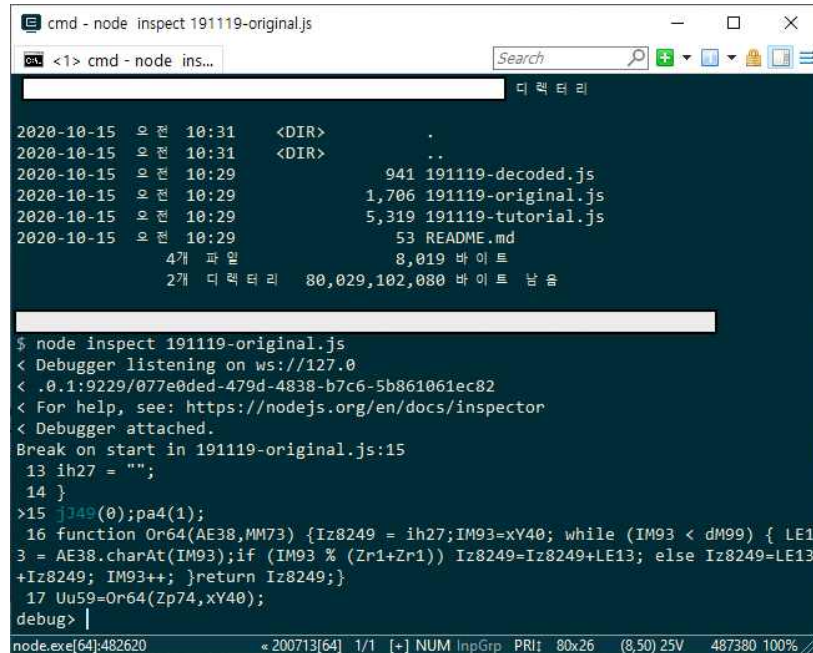
명령어 `node -v`로 설치 여부 및 버전을 확인할 수 있다. 이후 원활한 터미널 작업을 위해 별도의 명령 프롬프트(cmd)를 보조하는 프로그램(예: conemu)을 같이 이용하면 좋다.



[그림 4] Node.js 버전 확인

5. 분석 대상 스크립트를 Node inspect에 붙이기(attach)

실습 파일 중 `original` 파일의 이름을 알아둔 뒤, `node inspect [파일이름]` 명령으로 디버깅을 시작할 수 있다.



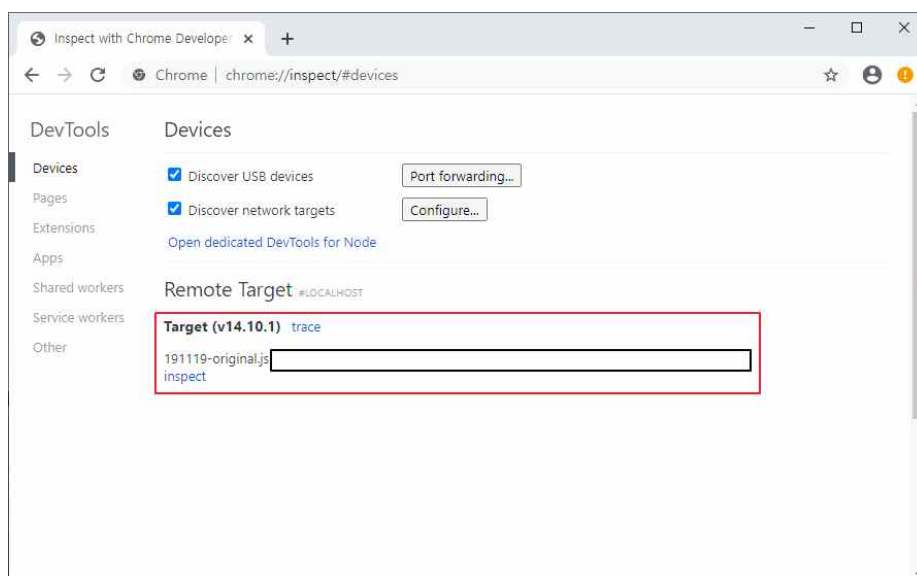
```
cmd - node inspect 191119-original.js
<1> cmd - node ins...
2020-10-15 오전 10:31 <DIR> .
2020-10-15 오전 10:31 <DIR> ..
2020-10-15 오전 10:29          941 191119-decoded.js
2020-10-15 오전 10:29        1,706 191119-original.js
2020-10-15 오전 10:29        5,319 191119-tutorial.js
2020-10-15 오전 10:29          53 README.md
          4개 파일          8,019 바이트
          2개 디렉터리 80,029,102,080 바이트 남음

$ node inspect 191119-original.js
< Debugger listening on ws://127.0.0.1:9229/077e0ded-479d-4838-b7c6-5b861061ec82
< For help, see: https://nodejs.org/en/docs/inspector
< Debugger attached.
Break on start in 191119-original.js:15
 13 ih27 = "";
 14 }
>15 j349(0);pa4(1);
 16 function Or64(AE38,MM73) {Iz8249 = ih27;IM93=xY40; while (IM93 < dM99) { LE1
3 = AE38.charAt(IM93);if (IM93 % (Zr1+Zr1)) Iz8249=Iz8249+LE13; else Iz8249=LE13
+Iz8249; IM93++; }return Iz8249;}
 17 Uu59=Or64(Zp74,xY40);
debug>
```

[그림 5] Node inspect 사용 화면

6. 크롬 웹 브라우저 `DevTools(개발도구)` 접속

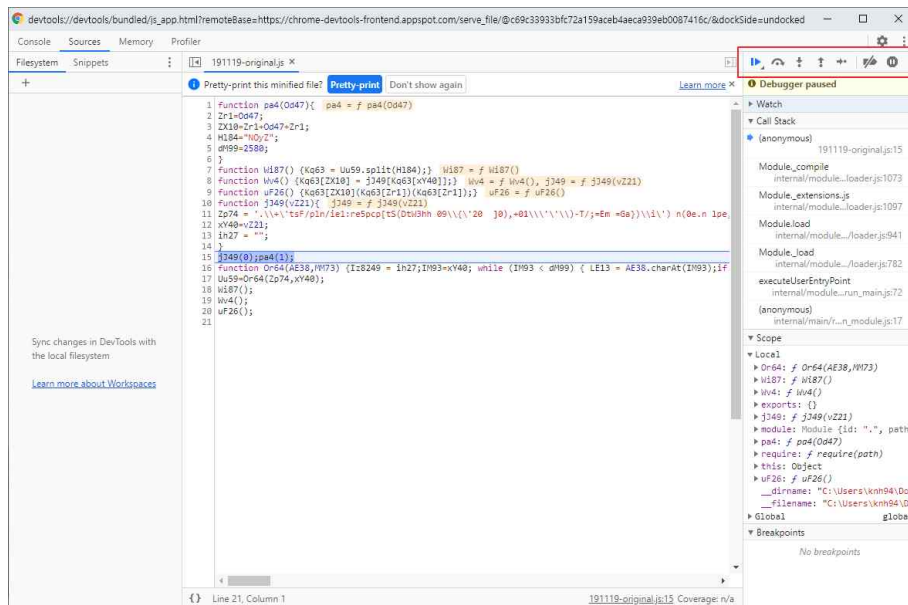
크롬 웹 브라우저를 열고 주소창에 `chrome://inspect`로 접속하면 아래와 같은 화면을 볼 수 있다. `Remote Target` 섹션에 Node inspect에 붙은 스크립트가 표시된다. 디버깅을 위해 `inspect`를 눌러준다.



[그림 6] 크롬 웹 브라우저 `DevTools` (개발도구)

7. 크롬 디버거 사용

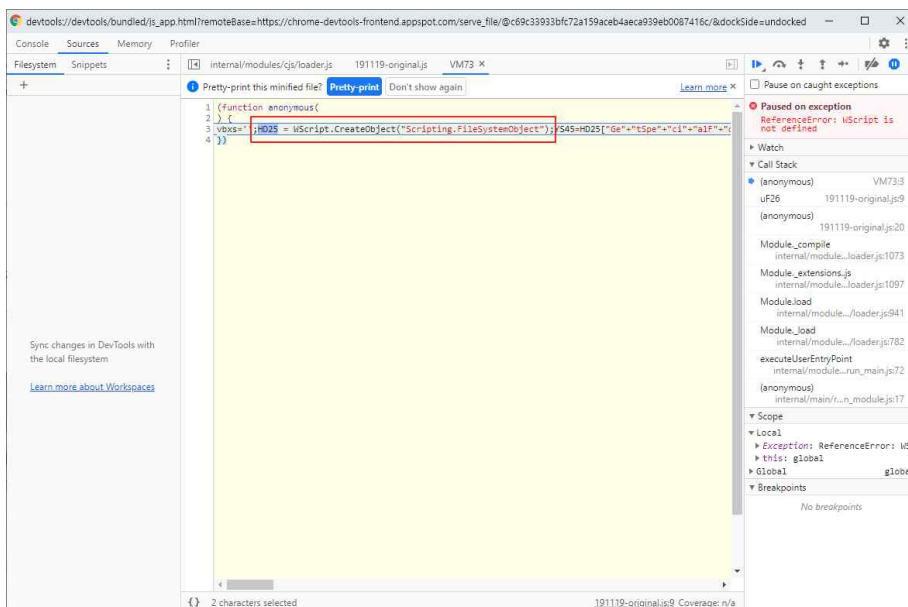
`inspect`를 누르면 새로운 창이 하나 뜨는데 디버깅을 할 수 있는 창이다. 아래에 표시된 버튼을 클릭하거나 지정 단축키로도 디버깅이 가능하다.



[그림 7] 크롬 디버거 화면

7. 디버깅 시작

이번 실습에 사용되는 예제는 예외가 발생하는 예제이므로 디버깅 컨트롤 버튼의 맨 마지막에 있는 `예외 발생 시 중단`을 눌러주면 된다. 이후 실행하면 아래와 같이 실행이 중단된 모습을 볼 수 있다.

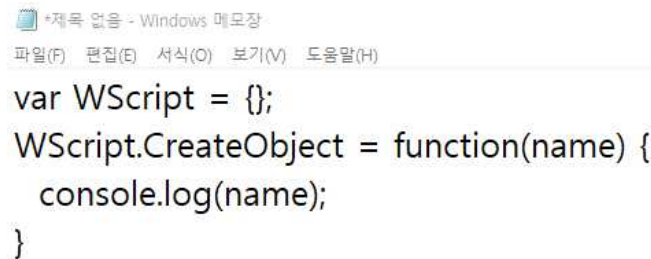


[그림 8] 예외 발생하여 중단된 화면

디버깅을 Node.js를 이용하여 시도 했기 때문에 `WScript` 객체와 하위 메소드의 정의가 없기 때문에 발생하는 예외이다. 이것을 복원해주는 과정이 필요하다.

8. 메소드 복원

`WScript`에서 객체를 생성하는 주요 메소드인 `CreateObject()`부터 아래와 같은 방식으로 복원을 시도할 수 있다.



```
*제목 없음 - Windows 메모장
파일(F)  편집(E)  서식(O)  보기(V)  도움말(H)

var WScript = {};
WScript.CreateObject = function(name) {
    console.log(name);
}
```

[그림 9] 메소드 복원 예시

이러한 방식으로 예외가 발생하는 메소드에 대한 복원 시도를 계속하다보면 최종적으로는 아래와 유사한 코드를 얻을 수 있다.



```
var WScript = {
  "CreateObject": function(arg1) {
    return function(arg1) {
      return {
        "FileExists": function(_arg1) {
          console.log(arg1 + " -> FileExists(arg1): " + _arg1);
          return false;
        },
        "CreateTextFile": function(_arg1, _arg2) {
          console.log(arg1 + " -> CreateTextFile(arg1): " + _arg1);
          console.log(arg1 + " -> CreateTextFile(arg2): " + _arg2);
          return {
            "Close": function() {
              console.log(arg1 + " -> CreateTextFile -> Close");
            }
          }
        },
        // https://docs.microsoft.com/en-us/office/vba/language/reference/user-in
        "GetSpecialFolder": function(_arg1) {
          switch(_arg1) {
            case 0:
              console.log(arg1 + " -> GetSpecialFolder(WindowsFolder)");
              break;
            case 1:
              console.log(arg1 + " -> GetSpecialFolder(SystemFolder)");
              break;
            case 2:
              console.log(arg1 + " -> GetSpecialFolder(TemporaryFolder)");
            default:
              console.log(arg1 + " -> GetSpecialFolder(Nothing)");
          }
        }
      }
    }
  },
  "Quit": function() {
    console.log("Quit");
  }
};
```

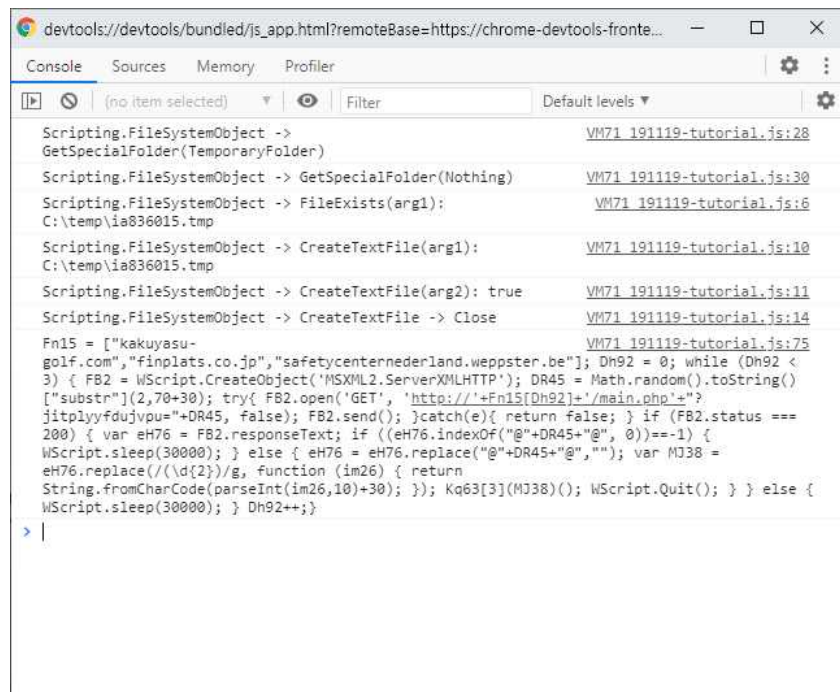
[그림 10] 임의 복원된 메소드 예시

9. 복호화 확인

사실 본 예제에서는 여기까지만 해도 스크립트를 단순 실행하는 것만으로도 복호화 결과를 볼 수 있다. (사실 디버거까지 안써도 된다.)

본 내용에서는 다루지 못했지만, '레이스 컨디션'(race condition)을 모방한 개념을 사용한 자바스크립트 난독화 사례에서는 디버거를 활용하는게 매우 도움이 될 수 있다.

그래도 취지에 맞게 복호화 결과를 확인해보자.

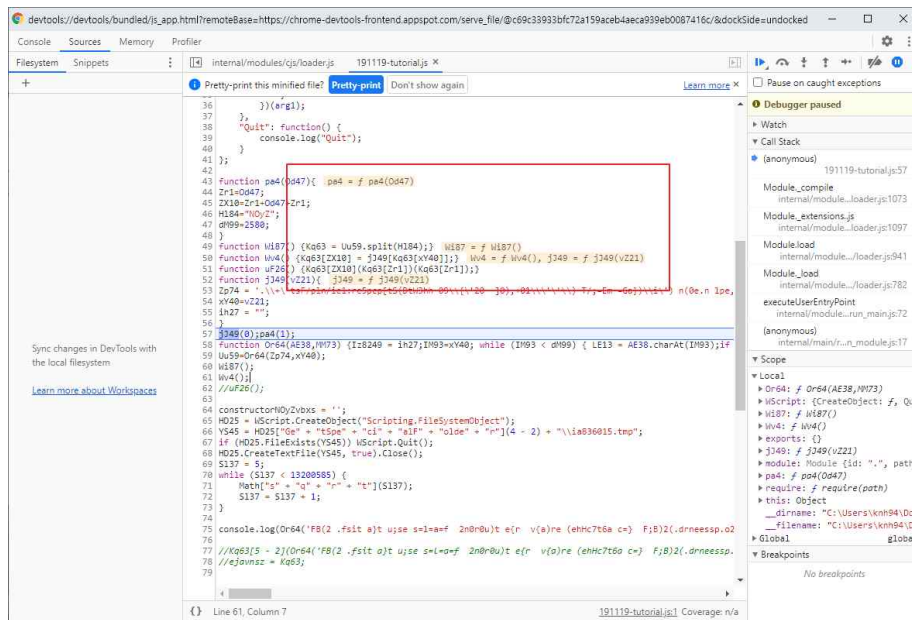


[그림 11] 디버거 콘솔 창

복호화 결과를 콘솔로 보내도록 해놓았기 때문에, 콘솔에서 그 결과를 확인할 수 있다.

10. 보너스

디버깅을 진행하다보면 형광펜 표시로 코드 옆에 작은 정보들이 올라오는데 스크립트에 따라선 아예 복호화된 내용이 바로 보이거나, 복호화가 잘 안되는 경우 원인에 대한 힌트를 얻을 수 있어 잘 활용하면 유용하다.



[그림 12] 코드 정보 확인

여기까지!