웹 취약점 진단을 위한
Fiddler 사용법

2016. 07.

호레

제•개정 이력

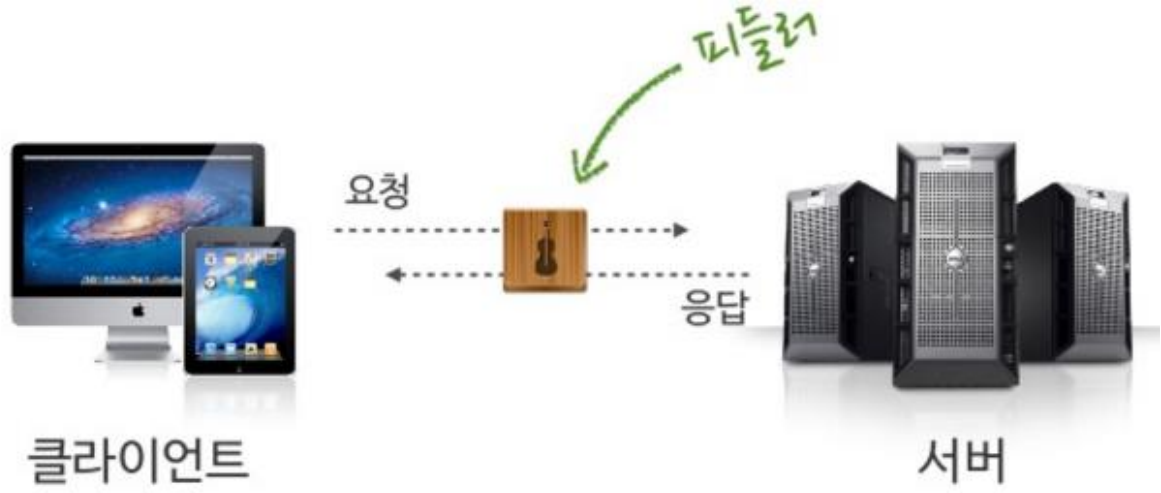
개정	제•개정 페이지 및 내용	제•개정 일자
1.0	최초 작성	2016. 07. 24

목 차

1. 개요

1.1. 도구 개요

Fiddler 는 개인 PC 의 웹브라우저와 Internet 사이에서 발생하는 모든 HTTP(S)트래픽을 로깅하는 HTTP Debugging Proxy 이다.

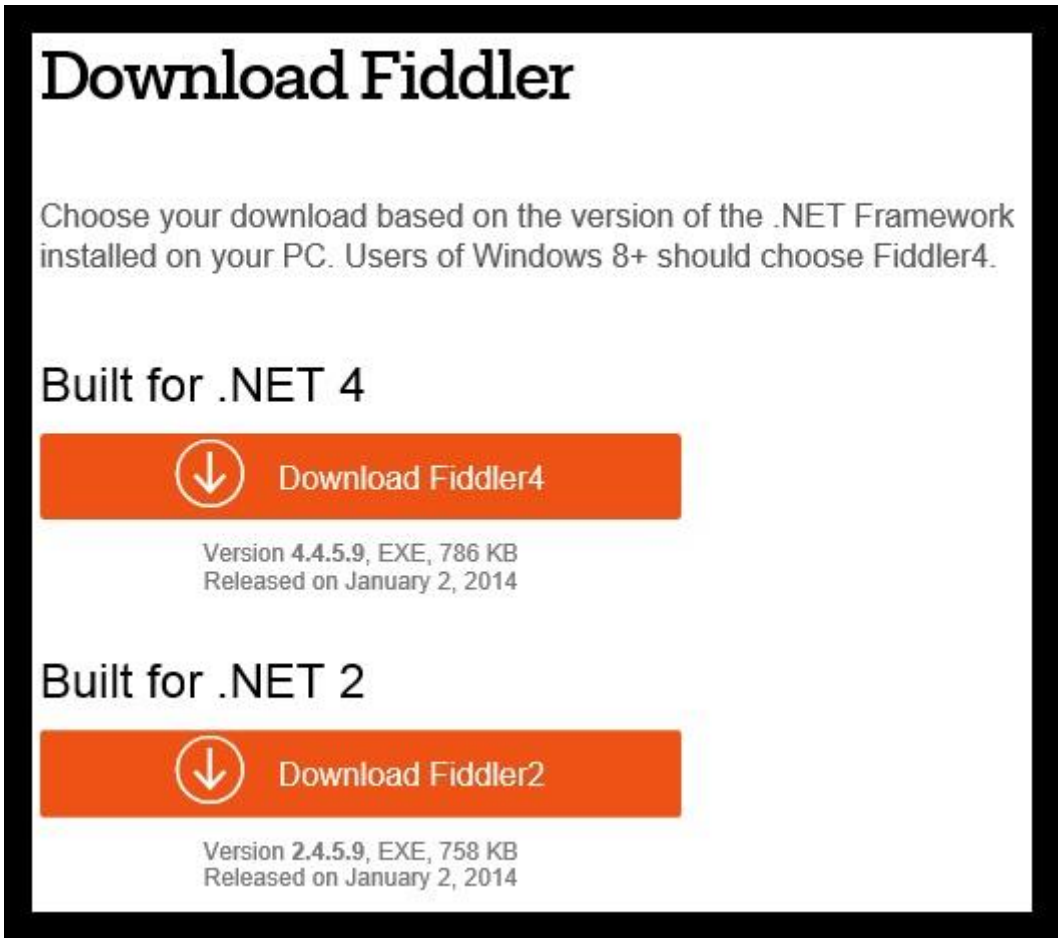


1.2. 도구 설치

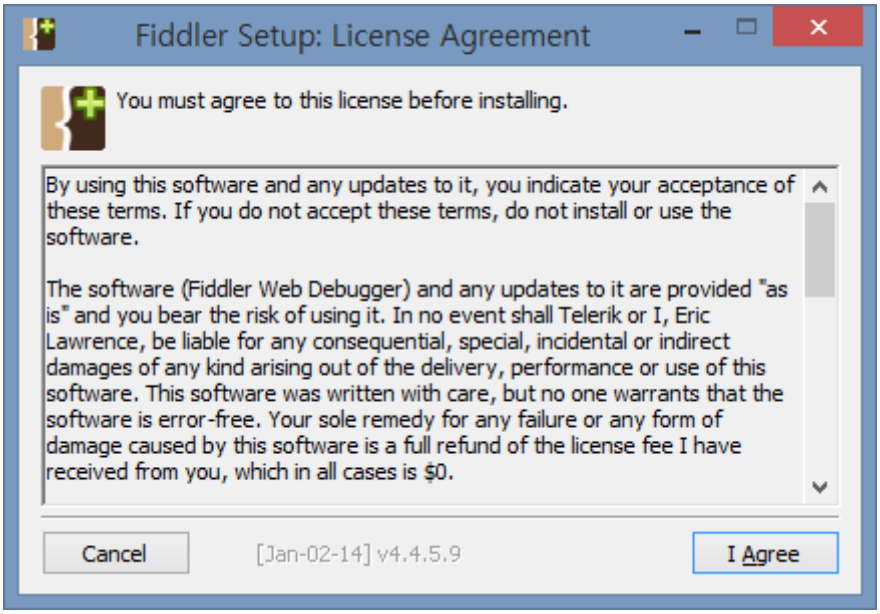
클릭 : **http://www.telerik.com/fiddler**

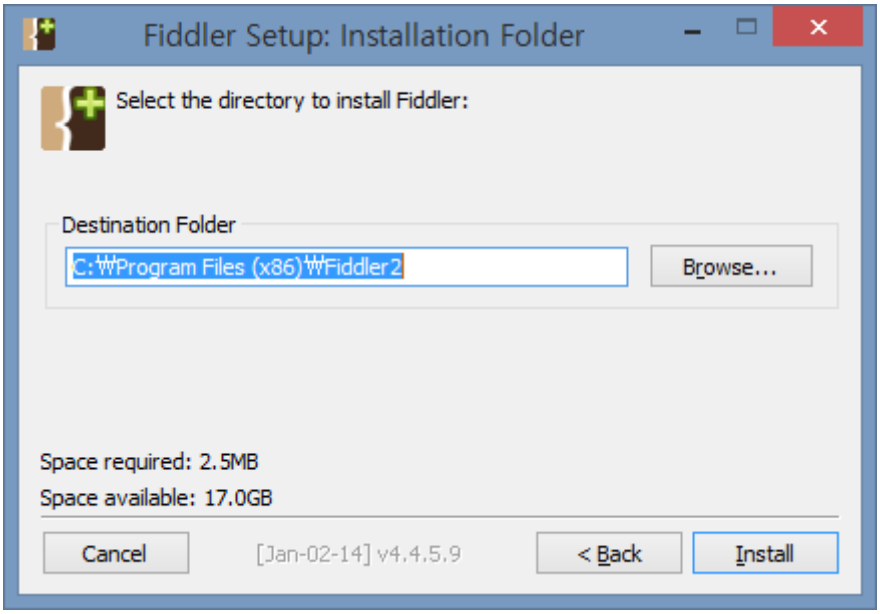
위 주소로 접속하셔서 오른쪽에 **Free Download** 로 들어 갑니다. **"Built for .NET 4", "Built for .NET 2"**의 다운로드가 2 가지 존재합니다. 본인의 컴퓨터에 **.Net Framework 4.x** 이 설치되어 있으면 **"Download Fiddler4"**를 클릭, **.Net Framework 2.x** 가 설치되어 있으면 **"Download Fiddler2"**를 클릭하여 설치하시면 됩니다.

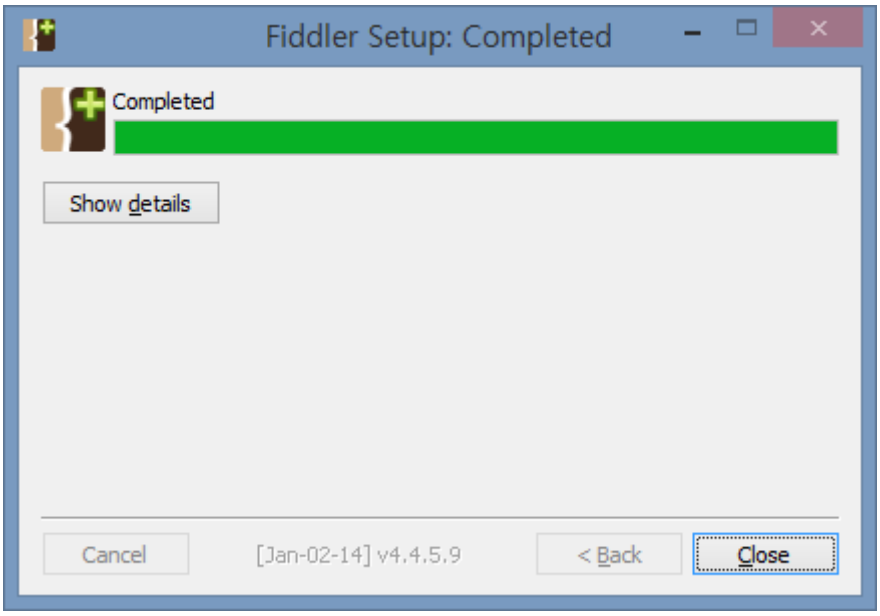
윈도우 7 이상이 설치되어 있으시면 .Net Framework 4.x 이 설치가 되어 있으실 껍니다. 확인이 힘드시면 "**%systemroot%₩Microsoft.NET₩Framework"** 폴더에서 철치된 버전을 확일 할 수 있습니다(%systemroot%는 윈도우 폴더).



다운로드를 하셔서 설치를 진행하시면 됩니다.

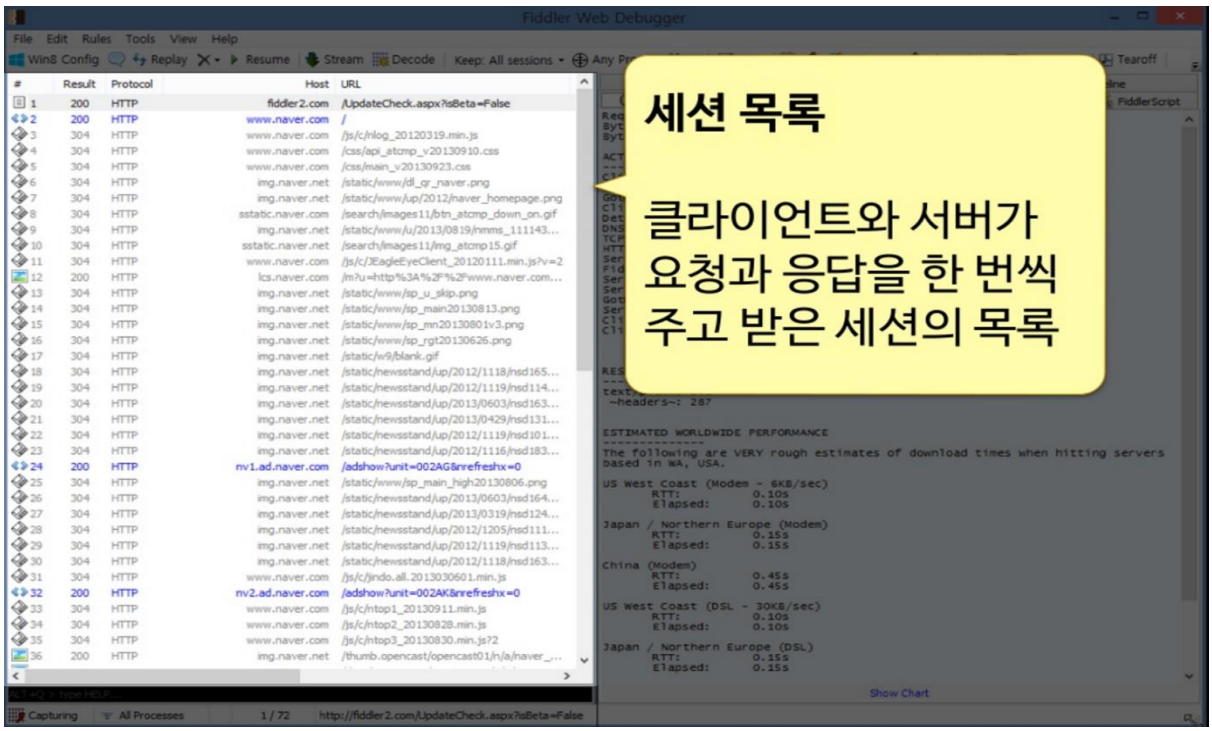


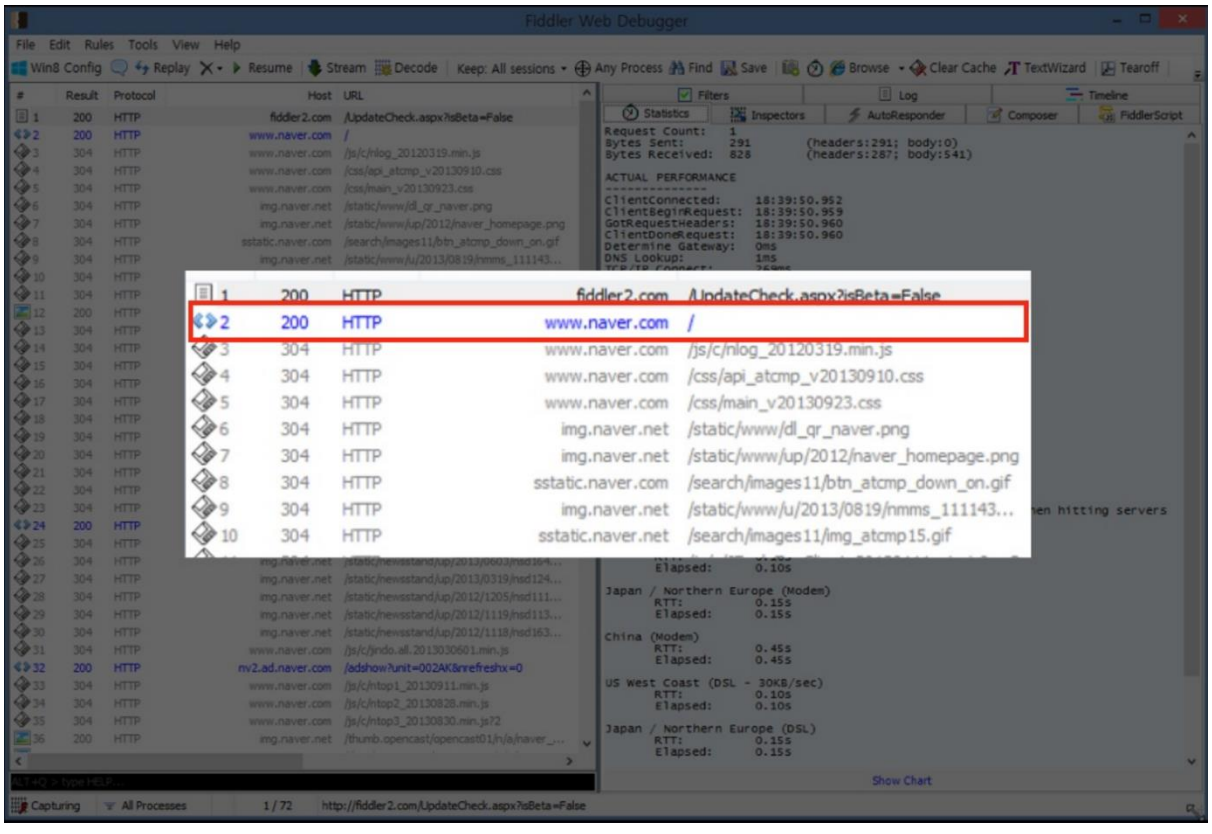

Fiddler Setup: Installation Folder
Select the directory to install Fiddler:
Destination Folder
C:₩Program Files (x86)₩Fiddler2
Browse...
Space required: 2.5MB
Space available: 17.0GB
Cancel
[Jan-02-14] v4.4.5.9
< Back
Install


Fiddler Setup: Completed
Completed
Show details
Cancel
[Jan-02-14] v4.4.5.9
< Back
Close

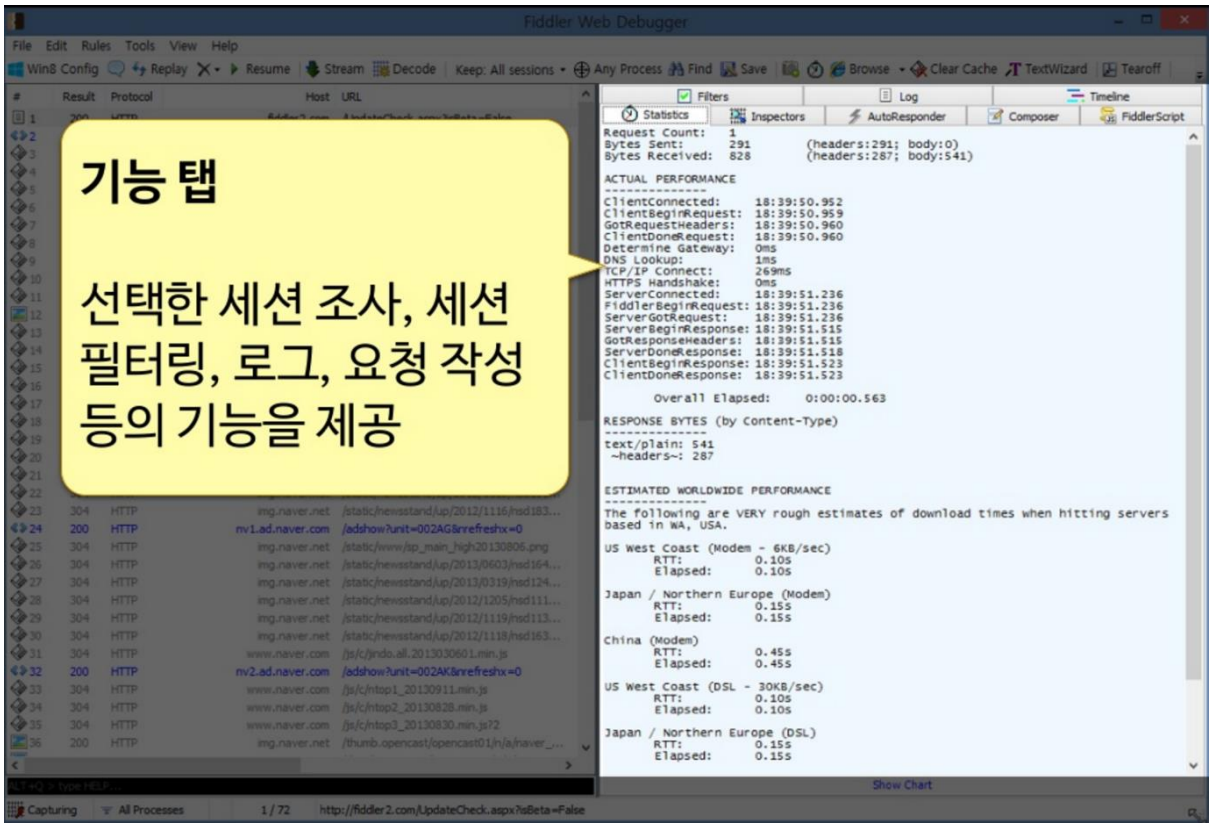
1.3. 도구 구성

세션목록

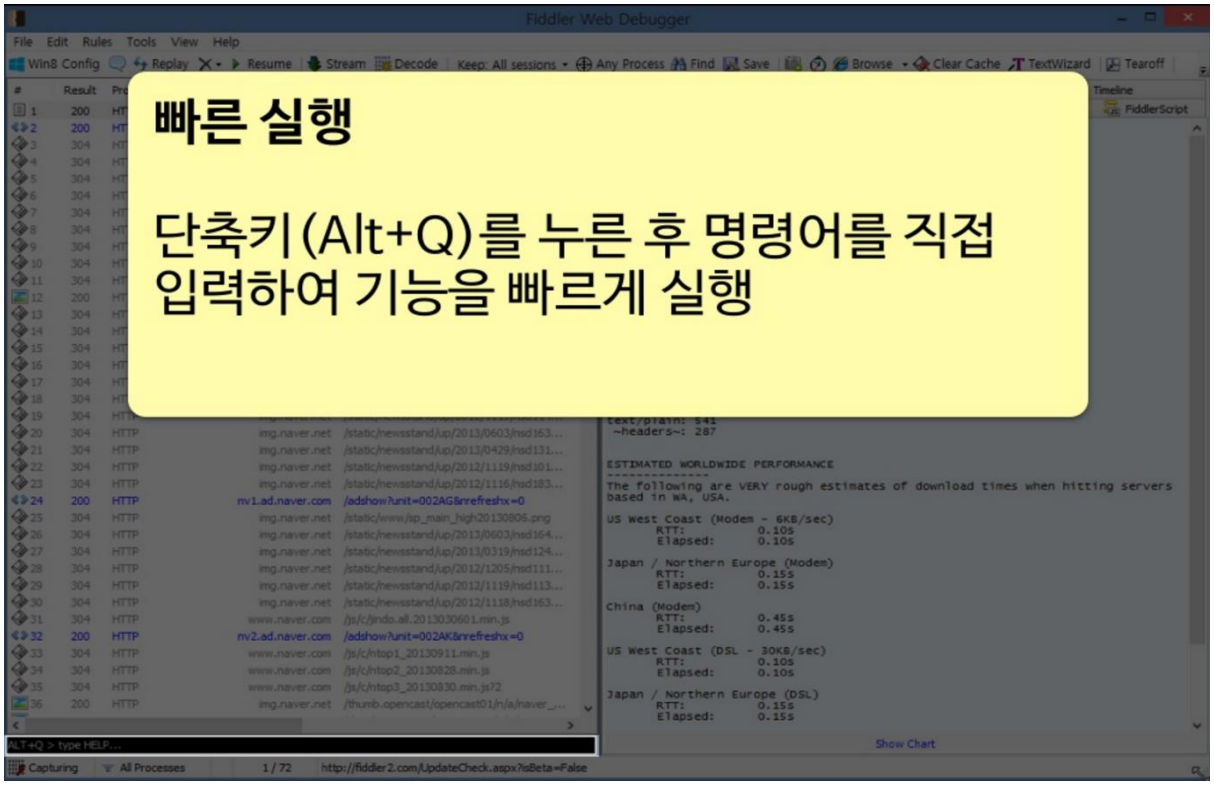




기능탭



빠른 실행



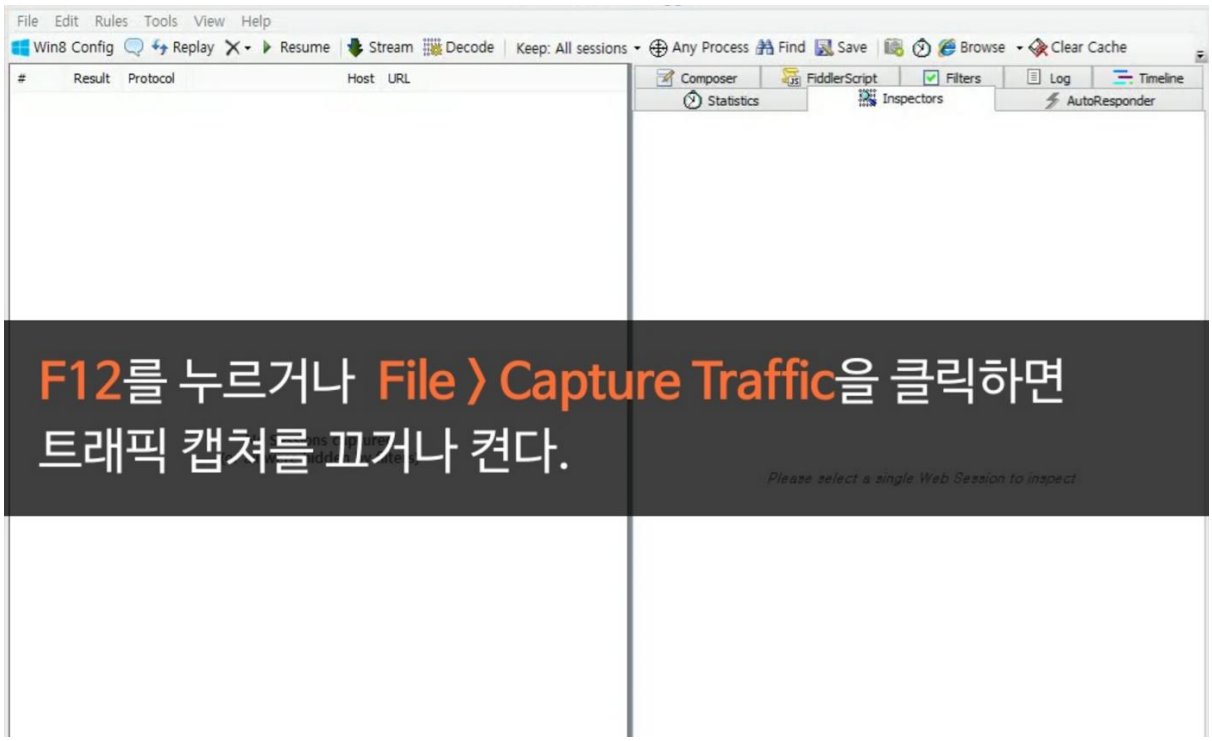
피들러 주요 기능

- 웹 디버깅
  - 웹 세션의 복호화화 압축해제
  - 다양한 시스템(윈도우 PC, Mac, 모바일 기기 등)에서 디버깅 트래픽
  - 다양한 클라이언트와 브라우저에서 디버깅 트래픽
  - 세션 데이터 분석
- HTTP/HTTPS 트래픽 기록
  - 트래픽 기록 보관과 리플레이
  - 모든 HTTP(s) 트래픽 캡쳐
  - 세션의 고급 정보
  - 캡쳐된 트래픽 필터
- 보안 테스팅
  - SSL 자동 복호화
  - 피들러 보안 애드온들
- 커스터마이징 피들러
  - 룰셋을 제작
  - 피들러 확장과 애드온들
  - 추가 감시자
  - .NET 코드와 피들러 스크립트로 피들러 확장
- 성능 테스팅
  - 병목현상 표시
  - 네 웹 어플리케이션의 성능 프로파일링
  - 성능분석 타임라인
  - HTTP 압축 시뮬레이션
  - HTTP 캐싱 장점 확인
- 웹 세션 조작
  - HTTP(s) 요청구성
  - 빠른 실행
  - 중단점 설정
  - HTTP(s) 요청이나 응답 조작
  - 원래 HTTP(s) 트래픽의 시뮬레이션

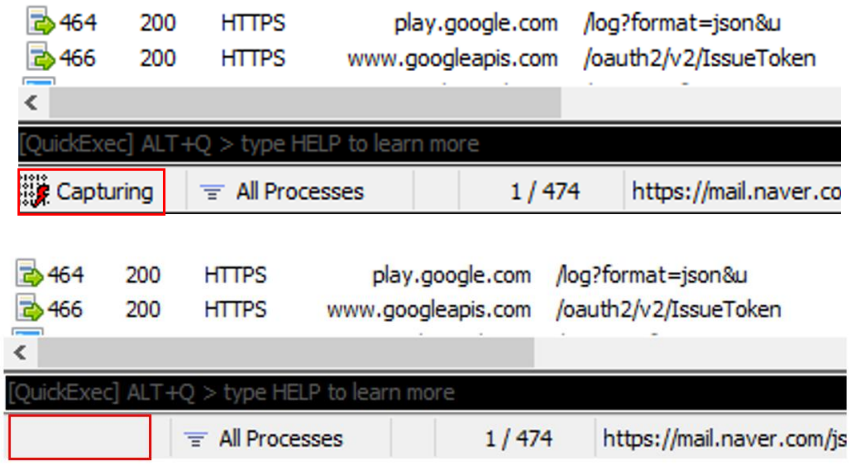
2. 모니터링 및 트래픽 조사

2.1. 모니터링

피들러를 실행후 F12 를 누르거나 File > Capture Traffic 을 클릭하면 트래픽 캡쳐를 끄거나 실행할 수 있다.

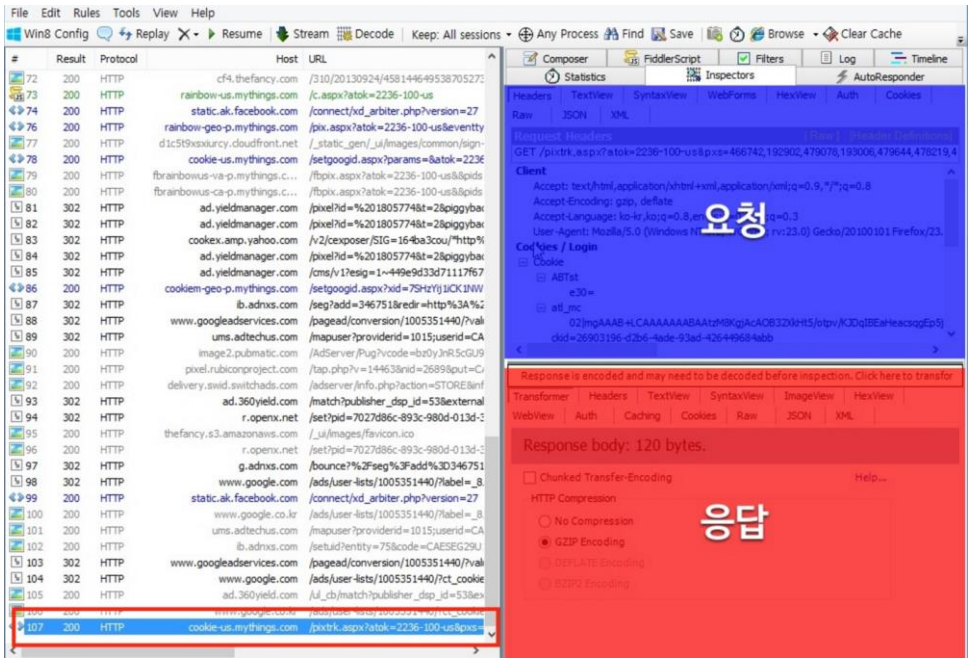


피들러 좌측 하단을 보면 캡쳐 상태를 확인할 수 있다.

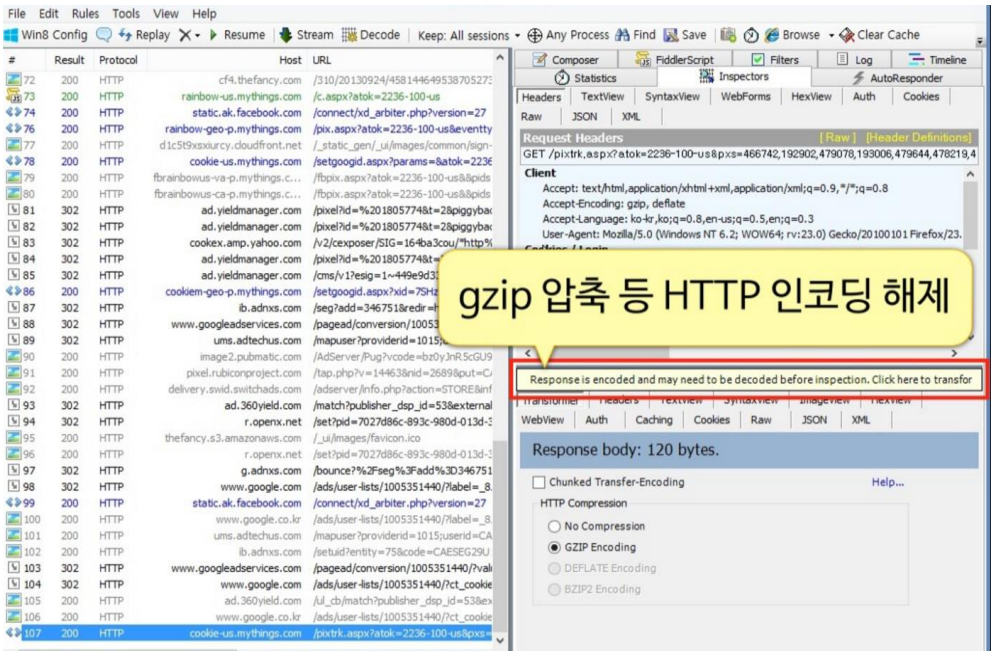


2.2. 트래픽 확인

트래픽 수집 후 좌측에 표시된 URL을 클릭하게 되면 해당 URL에 대한 자세한 결과를 볼 수 있다. 아래 그림의 파란색 박스는 해당 URL에 대한 요청 내용이 들어 있으며, 빨간 박스는 해당 URL에 대한 응답 받은 값이 나와있다.

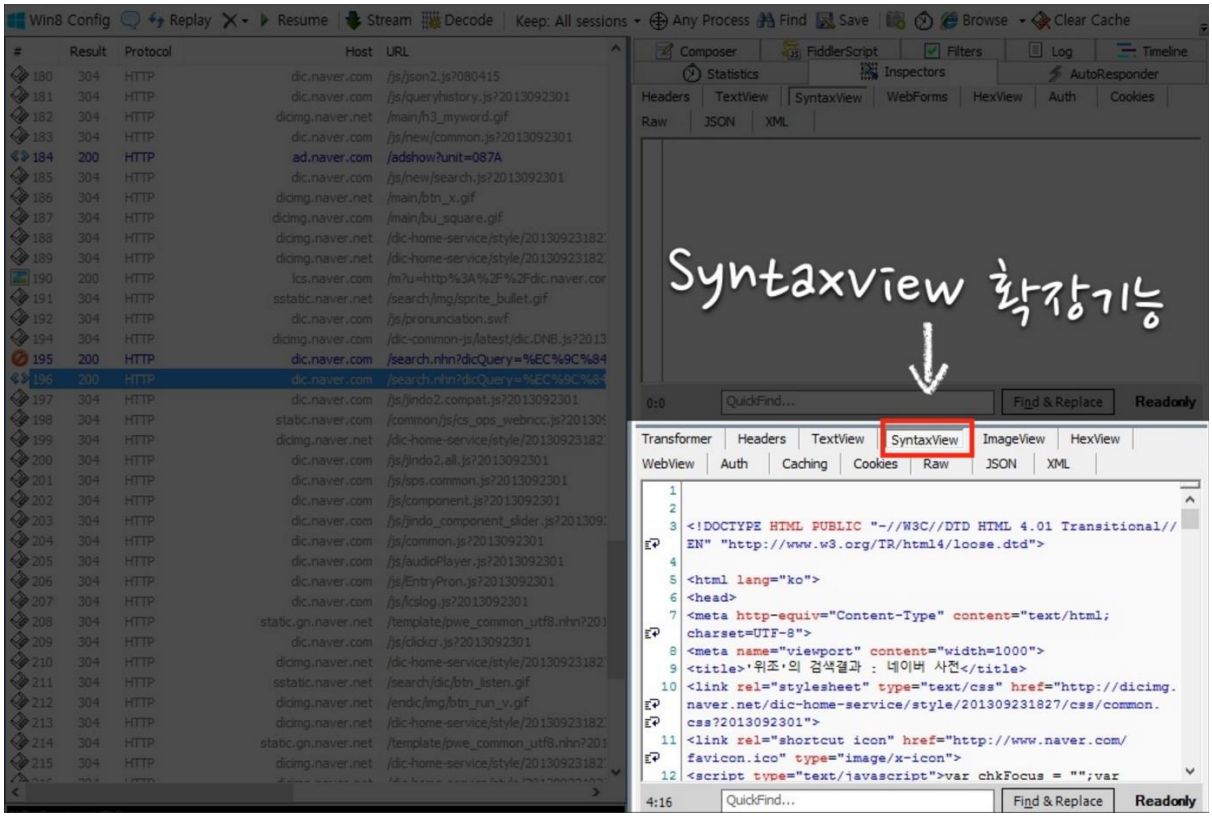


gzip 압축 등 HTTP 인코딩이 되어 있는 경우 아래 그림의 빨간 박스를 클릭하면 인코딩이 해제된다.

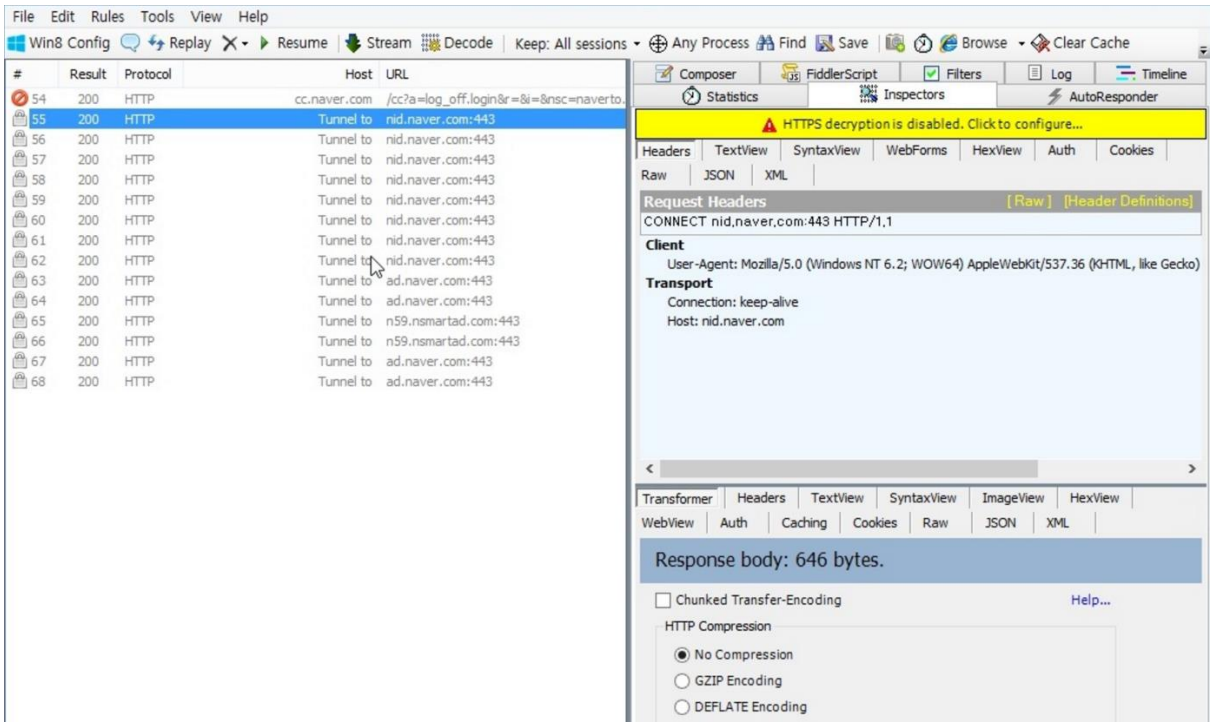


SyntaxView 기능은 HTML, Script, CSS, 그리고 XML 등을 Highlighted 해준다. SyntaxView 기능을 사용하기 위해서는 아래 그림과 같이 Download 후 설치해주면 된다.

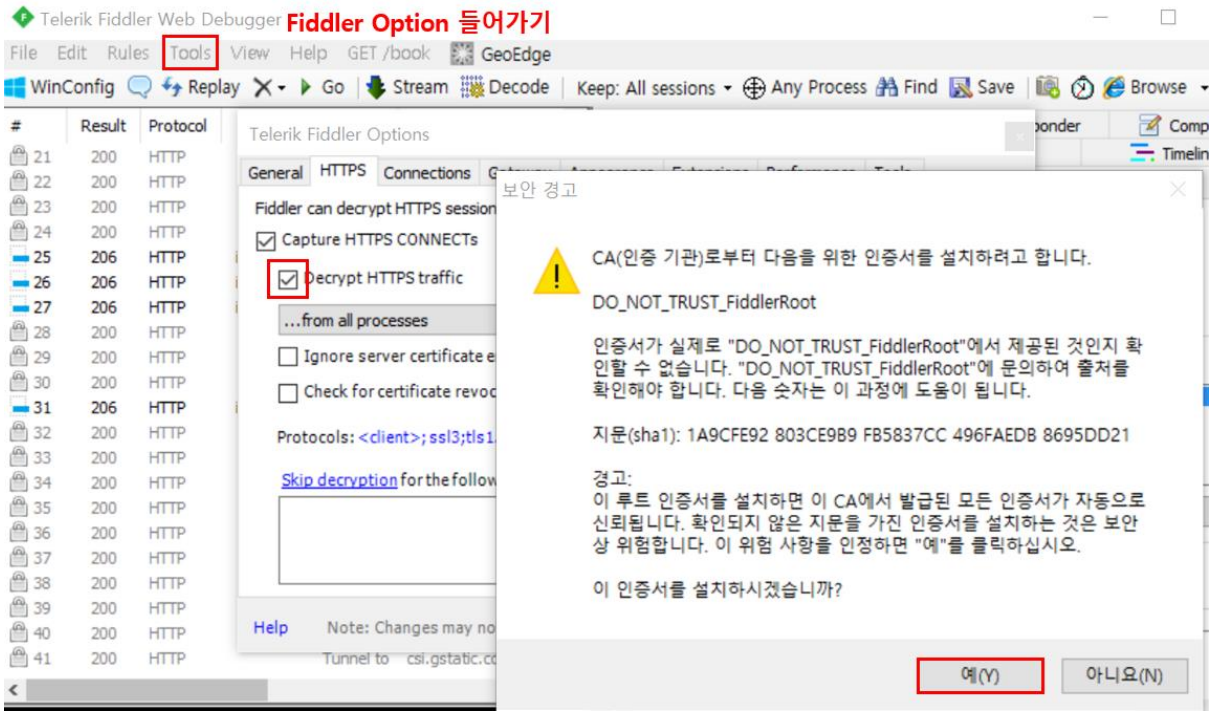
아래 그림은 Inspections 탭에 있는 SyntaxView 확장 기능을 선택한 화면이다.



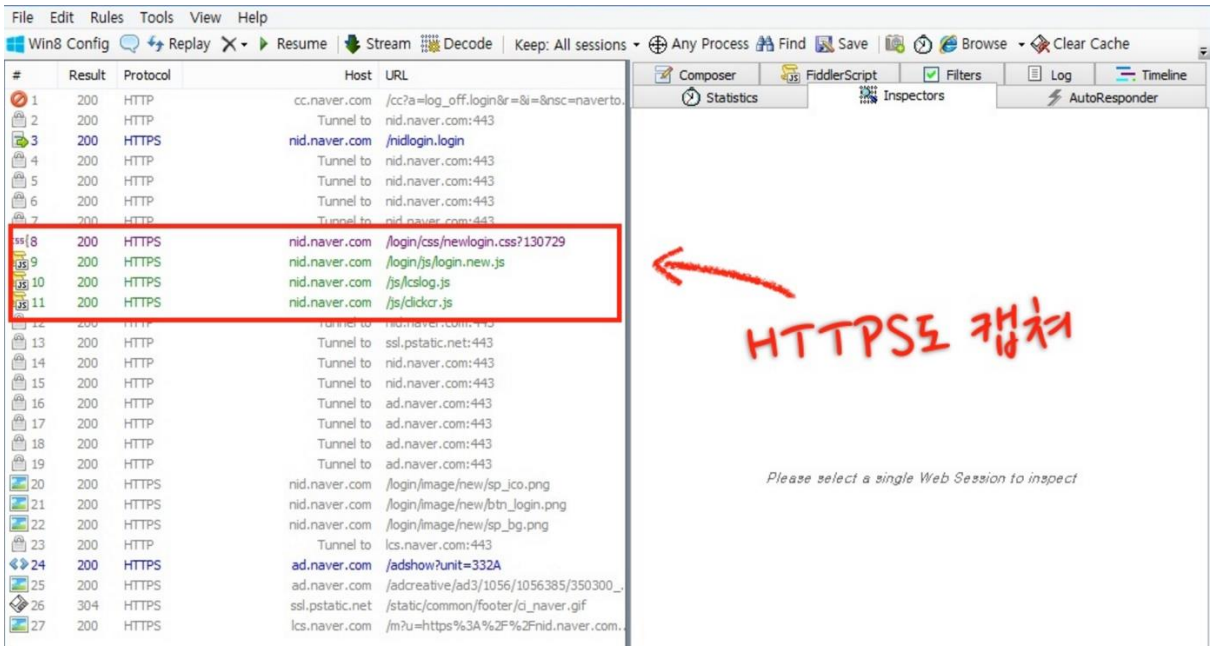
아래 그림을 보면 HTTPS 전송을 하는 트래픽에 대해서 캡쳐 하지 못한다.



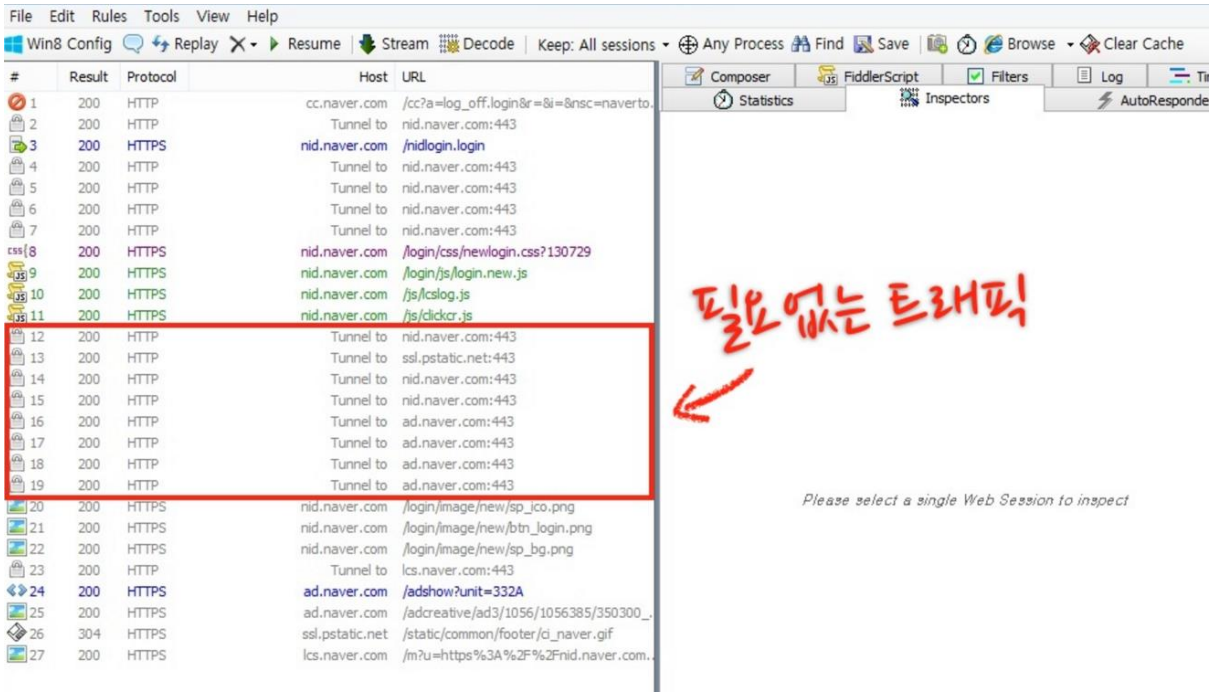
이와 같은 문제를 해결하기 위해서는 Tools > Fiddler Options >HTTPS 탭에 들어간 후 Decrypt HTTPS traffic 을 설정해주어야 한다.



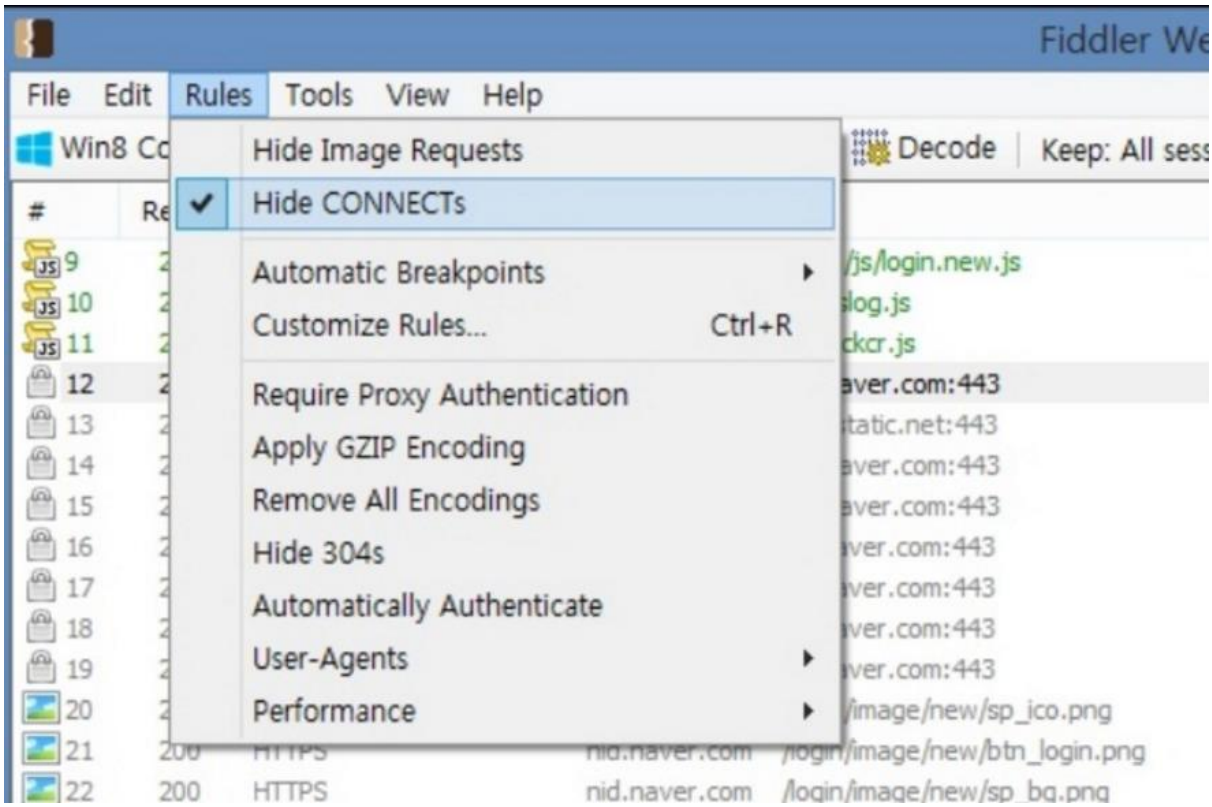
설정 후 HTTPS 패킷도 캡쳐가 되는 것을 확인 할 수 있다.



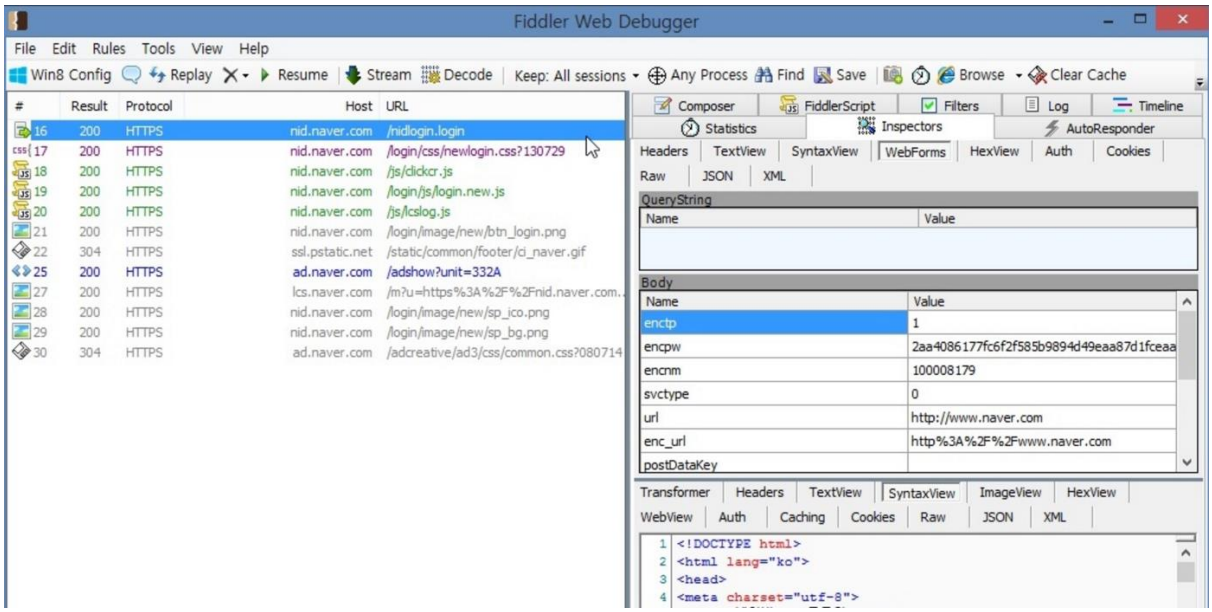
아래 그림을 보면 불필요한 트래픽도 나오는 것을 확인할 수 있다.



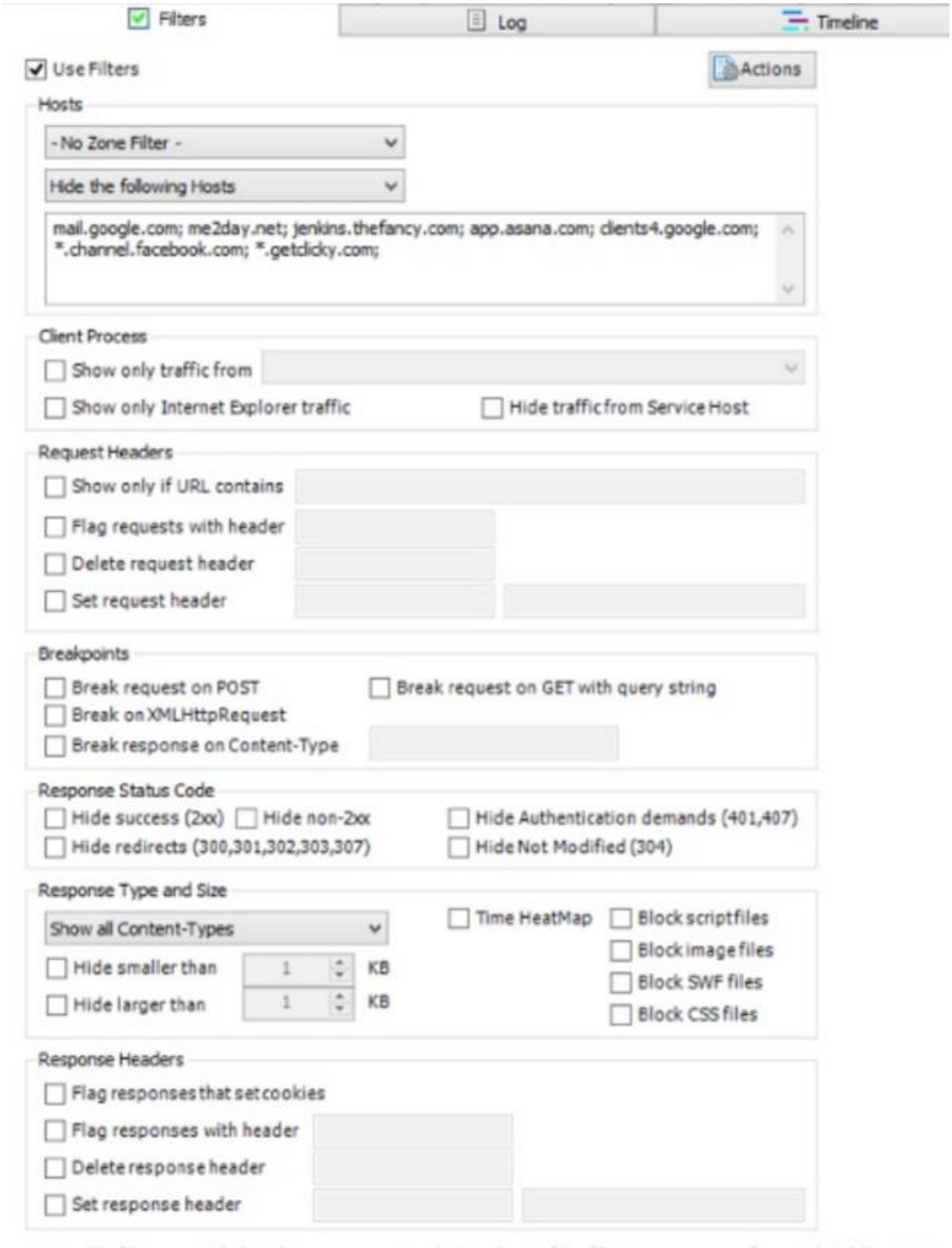
불필요한 트래픽을 제거 하기 위해서는 아래 그림과 같이 Rules > Hide Connects 를 체크해주면 된다.



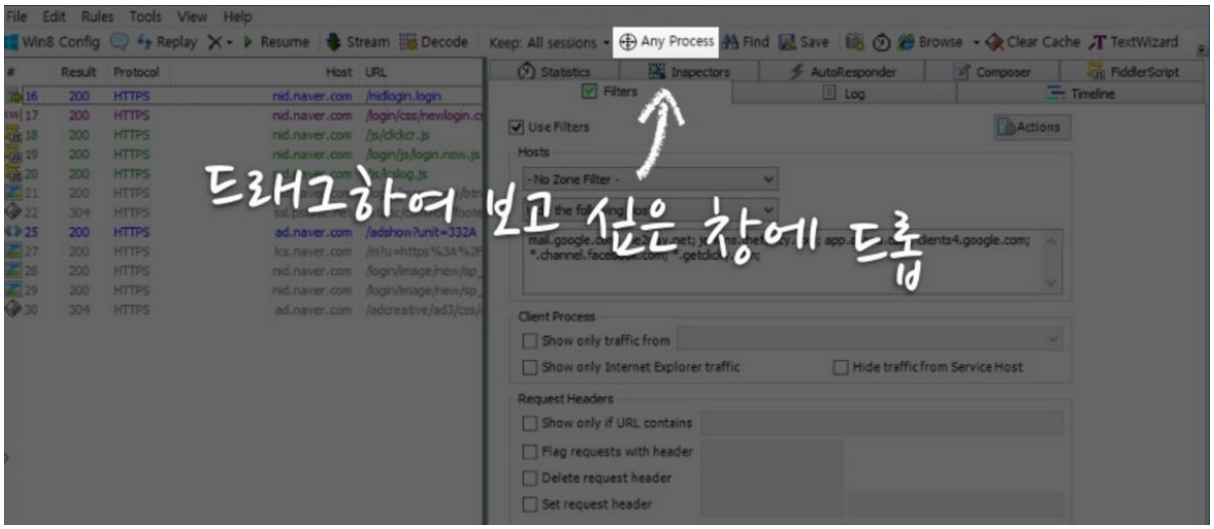
아래 그림은 불필요한 트래픽을 제거한 화면이다.



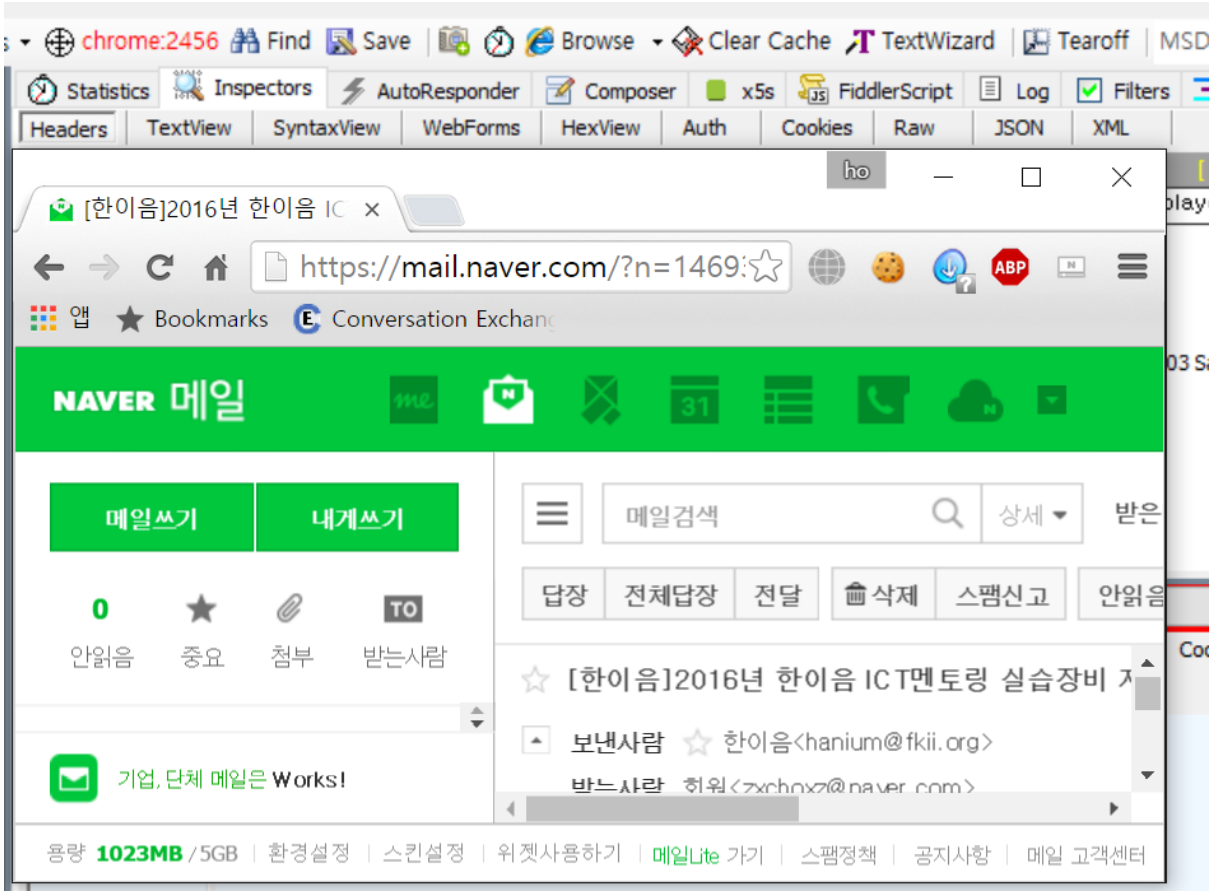
Filters 탭을 사용하면 트래픽을 손쉽게 필터링 할 수 있다. 예) JS/ CSS 차단, 플래시차단 등



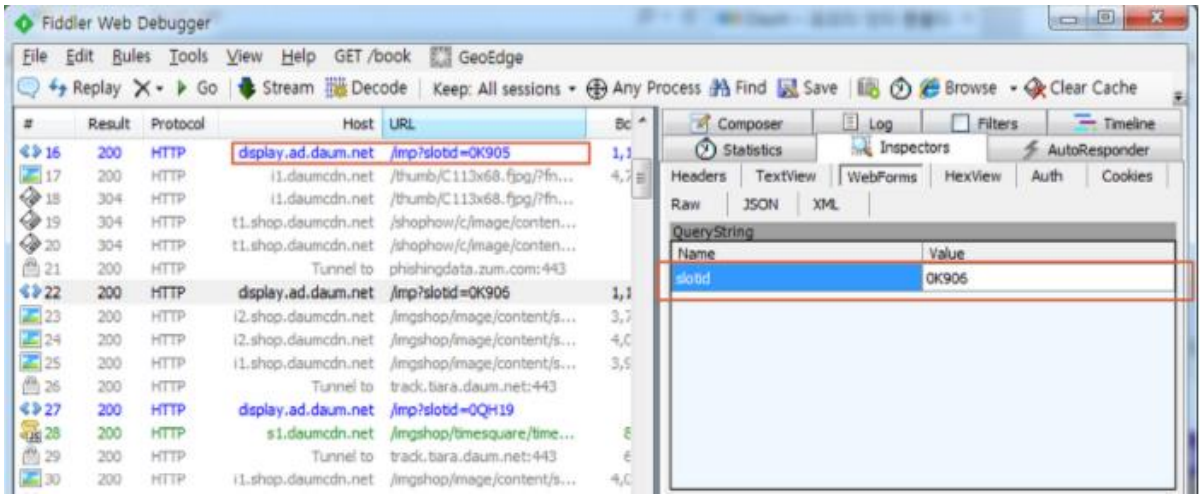
특정 프로세스(브라우저)에 대해서만 트래픽 감시를 하고 싶을 때는 아래 그림과 같이 AnyProcess 버튼을 드래그 하여 보고 싶은 브라우저에 드롭하면 된다.



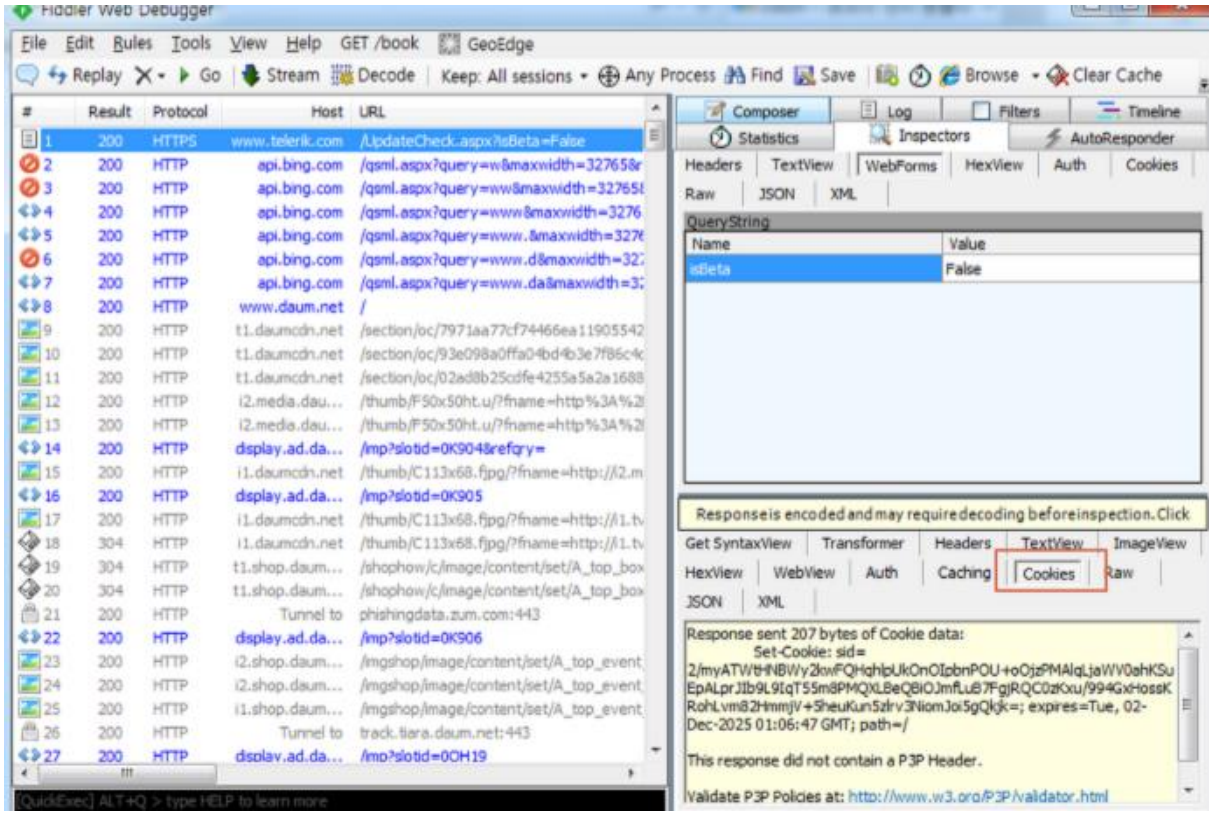
아래 그림은 특정 크롬(pid:2456)에 대해서만 트래픽 감시중인 화면이다.



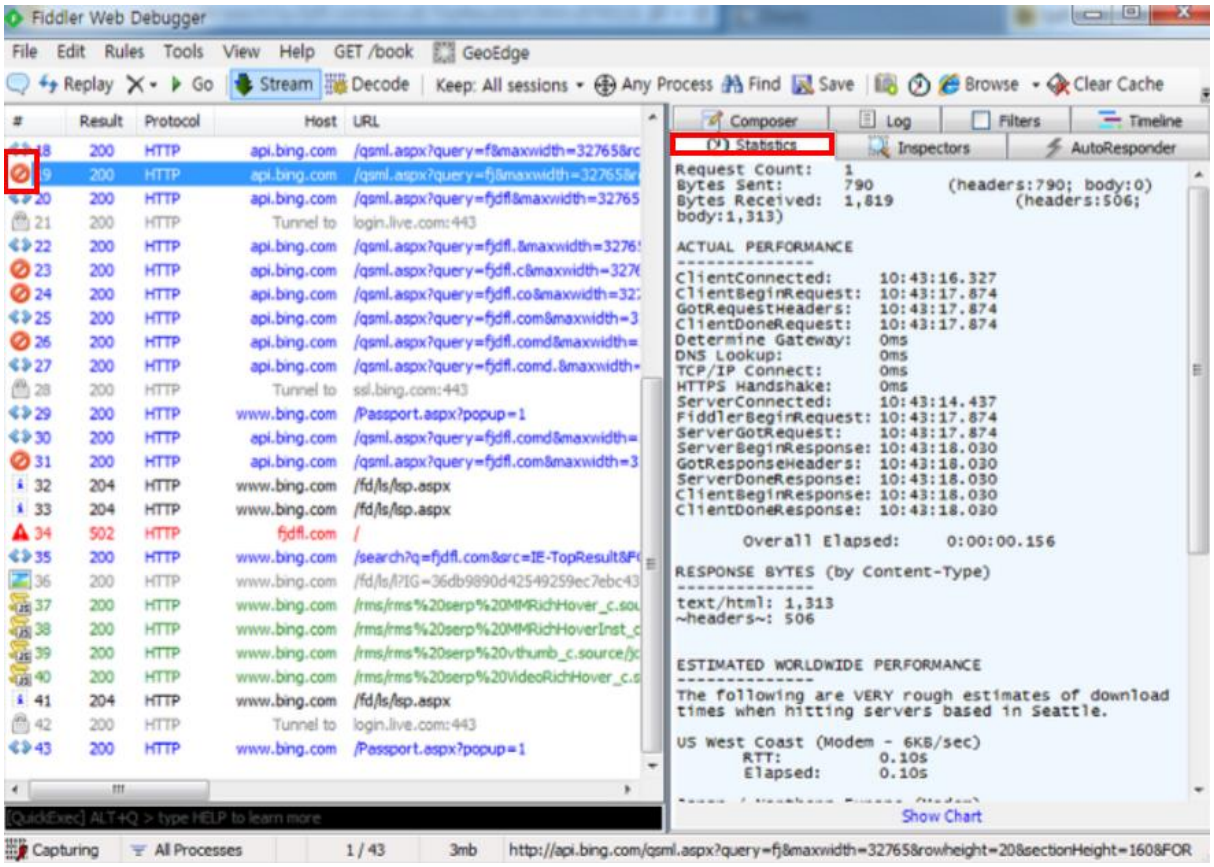
Fiddler 의 Inspectors > WebForms 탭을 통하여 파라미터로 입력된 값을 확인 할 수 있다.
이는 POST / GET 모두 확인 가능



Fiddler 의 Inspectors > Cookies 탭을 통하여 쿠키 값도 확인 가능하다.



좌측에 빨간색 금지 아이콘이 나와있는 URL 은 응답 실패한 경우나 비정상 페이지이다.
우측에 Statistics 에는 접속 시간, 응답시간 등이 표시가 된다.


Fiddler Web Debugger
File Edit Rules Tools View Help GET /book GeoEdge
Replay Go Stream Decode Keep: All sessions Any Process Find Save Browse Clear Cache
Result Protocol Host URL
18 200 HTTP api.bing.com /qsml.aspx?query=f&maxwidth=32765&rc
19 200 HTTP api.bing.com /qsml.aspx?query=fj&maxwidth=32765&r
20 200 HTTP api.bing.com /qsml.aspx?query=fjdfl&maxwidth=32765
21 200 HTTP Tunnel to login.live.com:443
22 200 HTTP api.bing.com /qsml.aspx?query=fjdfl.&maxwidth=3276
23 200 HTTP api.bing.com /qsml.aspx?query=fjdfl.c&maxwidth=327
24 200 HTTP api.bing.com /qsml.aspx?query=fjdfl.co&maxwidth=32
25 200 HTTP api.bing.com /qsml.aspx?query=fjdfl.com&maxwidth=3
26 200 HTTP api.bing.com /qsml.aspx?query=fjdfl.comd&maxwidth=
27 200 HTTP api.bing.com /qsml.aspx?query=fjdfl.comd.&maxwidth=
28 200 HTTP Tunnel to ssl.bing.com:443
29 200 HTTP www.bing.com /Passport.aspx?popup=1
30 200 HTTP api.bing.com /qsml.aspx?query=fjdfl.comd&maxwidth=
31 200 HTTP api.bing.com /qsml.aspx?query=fjdfl.com&maxwidth=3
32 204 HTTP www.bing.com /fd/ls/lsp.aspx
33 204 HTTP www.bing.com /fd/ls/lsp.aspx
34 502 HTTP fjdfl.com /
35 200 HTTP www.bing.com /search?q=fjdfl.com&src=IE-TopResult&F
36 200 HTTP www.bing.com /fd/ls/l?IG=36db9890d42549259ec7ebc43
37 200 HTTP www.bing.com /rms/rms%20serp%20MMRichHover_c.sou
38 200 HTTP www.bing.com /rms/rms%20serp%20MMRichHoverInst_c
39 200 HTTP www.bing.com /rms/rms%20serp%20vthumb_c.source/jc
40 200 HTTP www.bing.com /rms/rms%20serp%20VideoRichHover_c.s
41 204 HTTP www.bing.com /fd/ls/lsp.aspx
42 200 HTTP Tunnel to login.live.com:443
43 200 HTTP www.bing.com /Passport.aspx?popup=1
[QuickExec] ALT+Q > type HELP to learn more
Composer Log Filters Timeline
Statistics Inspectors AutoResponder
Request Count: 1
Bytes Sent: 790 (headers:790; body:0)
Bytes Received: 1,819 (headers:506; body:1,313)
ACTUAL PERFORMANCE
ClientConnected: 10:43:16.327
ClientBeginRequest: 10:43:17.874
GotRequestHeaders: 10:43:17.874
ClientDoneRequest: 10:43:17.874
Determine Gateway: 0ms
DNS Lookup: 0ms
TCP/IP Connect: 0ms
HTTPS Handshake: 0ms
ServerConnected: 10:43:14.437
FiddlerBeginRequest: 10:43:17.874
ServerGotRequest: 10:43:17.874
ServerBeginResponse: 10:43:18.030
GotResponseHeaders: 10:43:18.030
ServerDoneResponse: 10:43:18.030
ClientBeginResponse: 10:43:18.030
ClientDoneResponse: 10:43:18.030
Overall Elapsed: 0:00:00.156
RESPONSE BYTES (by Content-Type)
text/html: 1,313
~headers~: 506
ESTIMATED WORLDWIDE PERFORMANCE
The following are VERY rough estimates of download times when hitting servers based in Seattle.
US West Coast (Modem - 6KB/sec)
RTT: 0.10s
Elapsed: 0.10s
Show Chart
Capturing All Processes 1 / 43 3mb http://api.bing.com/qsml.aspx?query=fj&maxwidth=32765&rowheight=20&sectionHeight=160&FOR

3. 필터링

3.1. Break Point 란?

➔ 요청 또는 응답을 조작할 수 있도록 세션을 일시적으로 중단시키는 지점.

Breakpoint 방법으로 5 가지가 있다.

1. Rules > Automatic breakpoints 메뉴
2. 빠른 실행 상자에 bpu 또는 bpa 명령
3. Filters 탭
4. AutoResponder 탭
5. 피들러 스크립트/확장 기능 사용

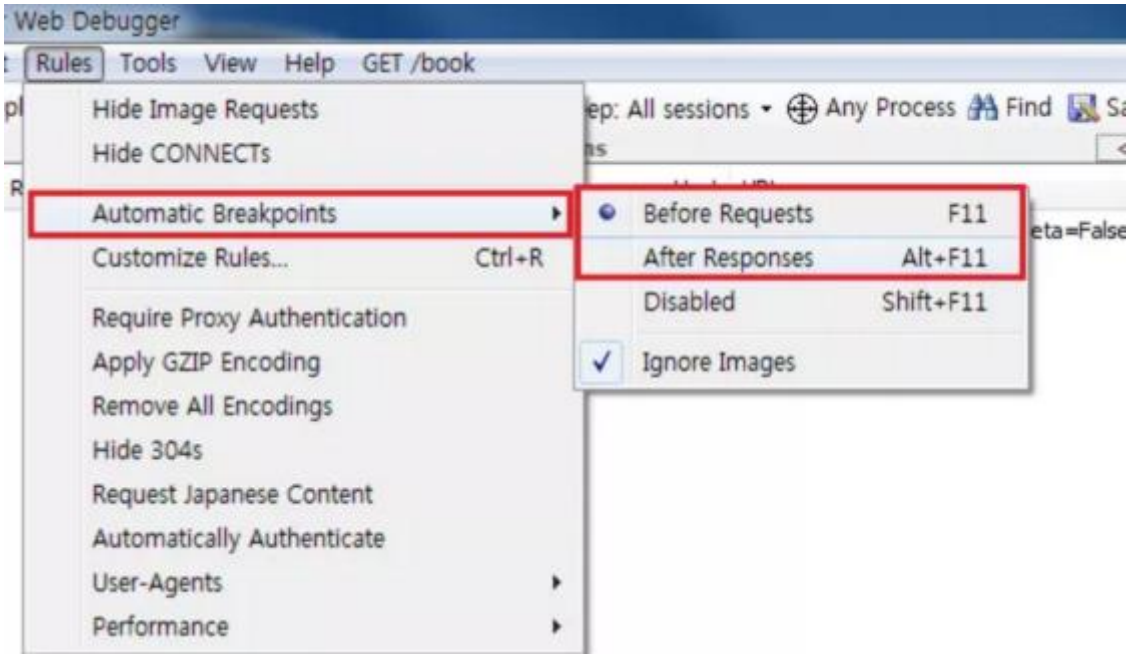
3.2. Rules > Automatic breakpoints 메뉴

Before Requests → 패킷 Requests 전에 breakpoints 을 한다. 파라미터 변조 등을 할 때 사용

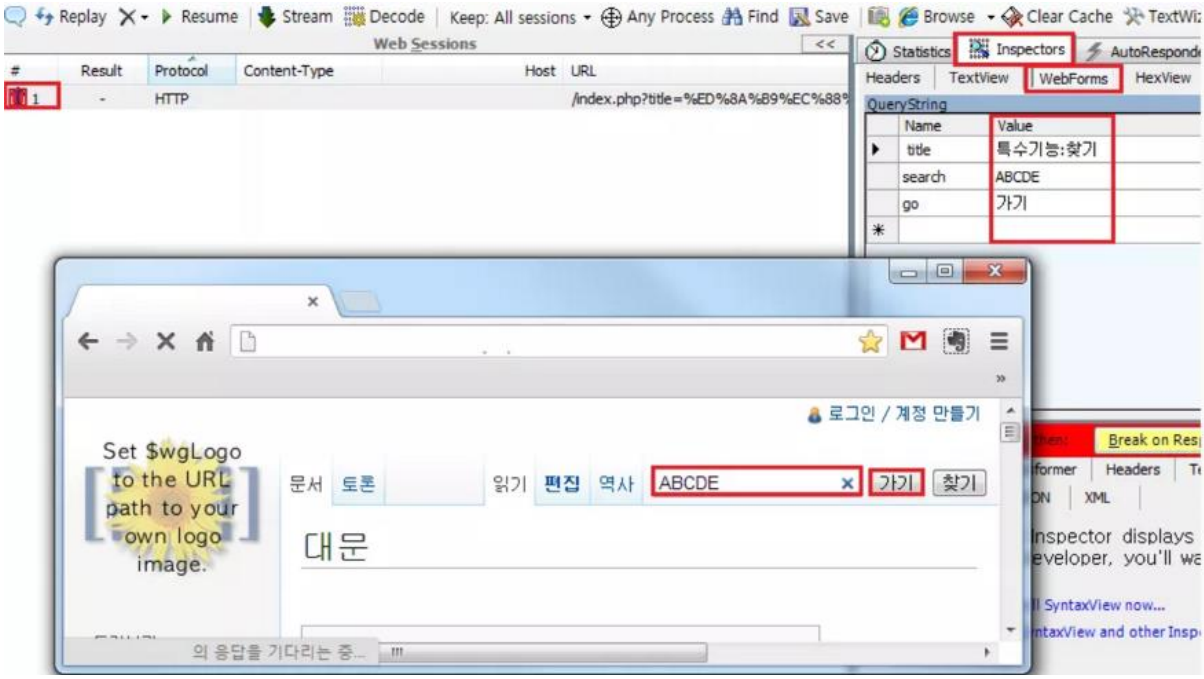
After Response → 응답을 받은 직 후

Disabled → 사용안함

Ignore Images → 이미지 요청은 무시



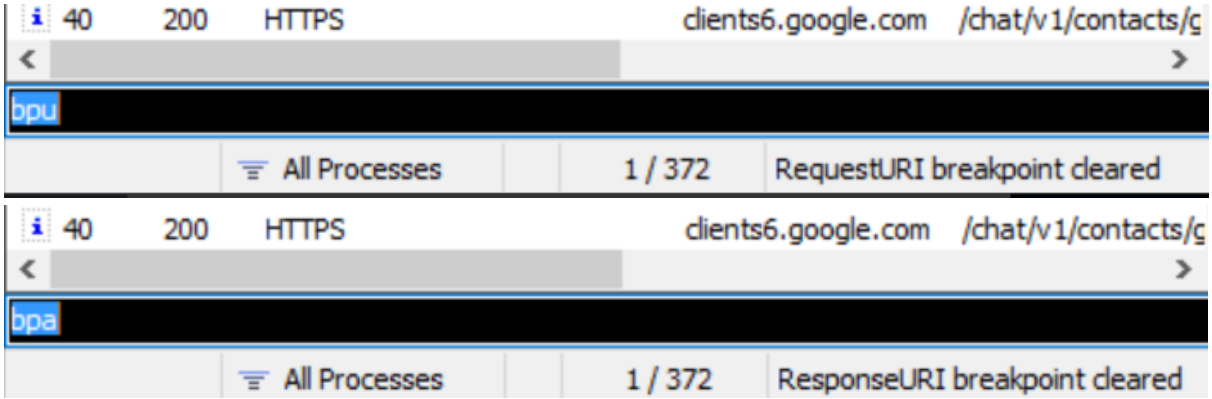
옵션을 설정한 후 어떠한 요청을 날리게 되면 피들러에 요청이 break 되며 표시된다. 해당 row 를 들어가 Inspectors > WebForms 를 확인해보면 어떠한 요청으로 보내려고 하는지가 테이블 형태로 확인할 수 있다. 중요한 점은 이 요청에 대한 Value 들을 변결 할 수 있다는 것이다. 추가도 가능하며, 지우기도 가능하다.



3.3. 빠른 실행 상자에 bpu 또는 bpa 명령

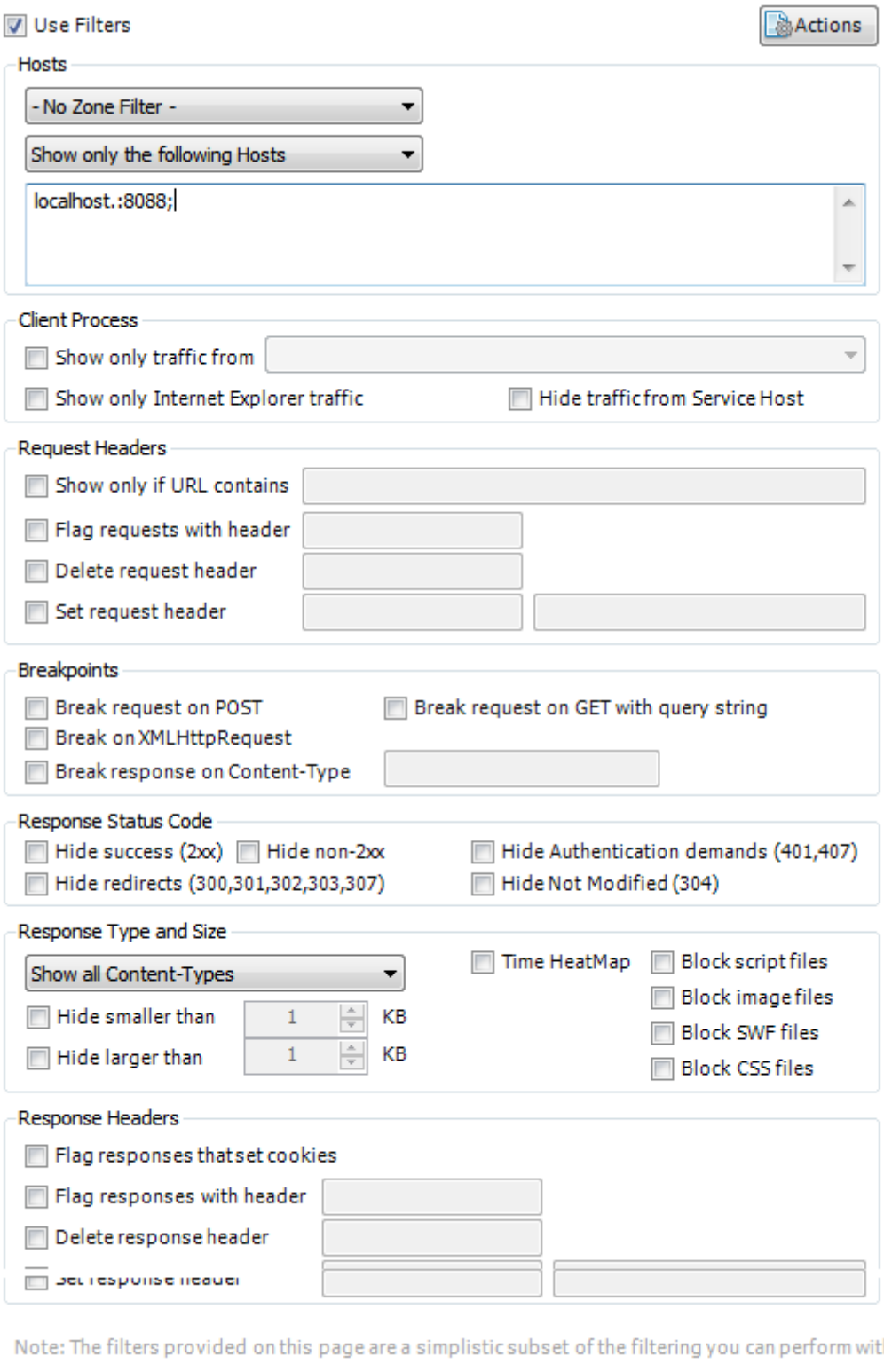
Bpu → RequestURI breakpoint

Bpa → ResponseURI breakpoint



3.4. Filters 탭

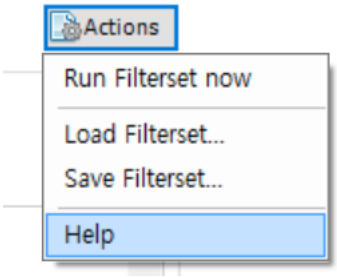
Filters 탭에 Use Filters 를 체크 후 세부 항목을 설정한다.



Note: The filters provided on this page are a simplistic subset of the filtering you can perform with Fic

3.4.1. Filters > Actions 속성

Use Filters 옆에 있는 Actions 를 클릭하면 Filterset 을 가지고 오거나 저장, 실행할 수 있다.

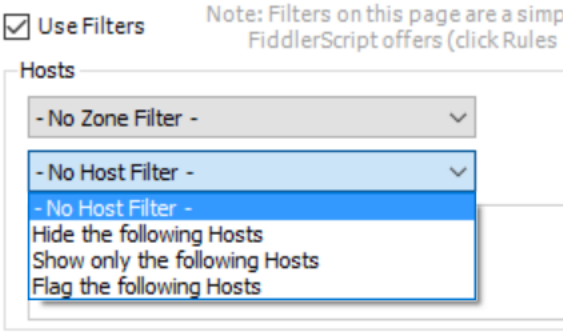


Host Filter 의 경우 3 가지로 나뉘어진다.

Hide the following hosts → 해당 Hosts 를 보여주지 않는다.

Show only the following Hosts → 해당 Hosts 만 보여준다.

Flag the following Hosts → 해당 Hosts 에 대해 강조하며, 다른 Hosts 도 같이 나온다.

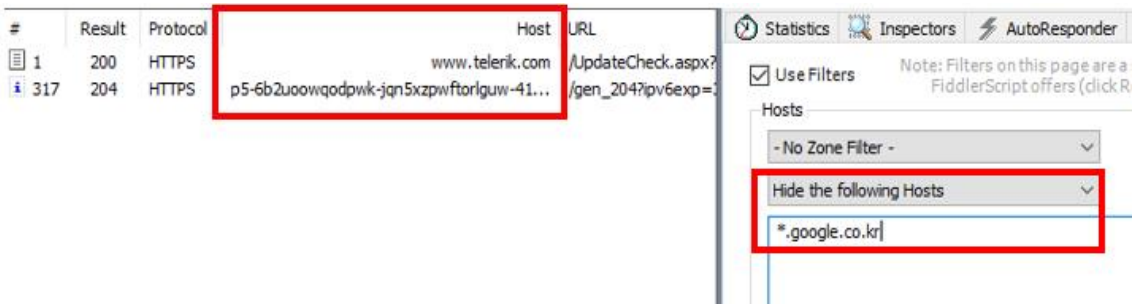


아래의 Host 항목을 통해 필터링 된다.

3.4.2. Filters > Hosts 속성

Hide the following hosts

➔ *.google.kr 이 포함되어 있는 URI 에 대하여 숨김 필터링

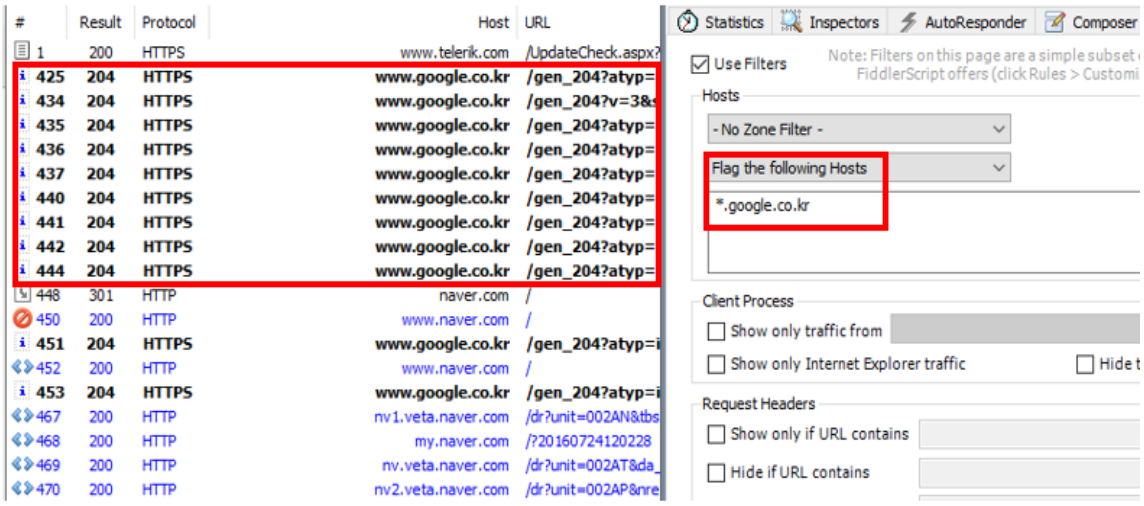


Show only the following Hosts

➔ *.google.kr 이 포함되어 있는 URI 에 대하여만 필터링

Flag the following Hosts

➔ *.google.kr 이 포함되어 있는 URI 에 대해 강조 표시하며 *.google.kr 이외에 URI 도 표시된다.(www.naver.com 등)



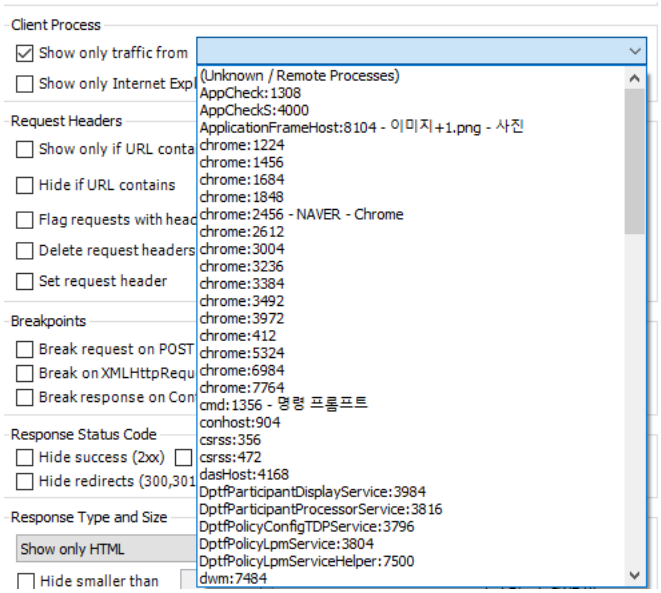
3.4.3. Client Process 속성

Client Process 는 아래 그림과 같이 세가지로 분류 된다.

Show only traffic from

➔ 자신이 원하는 프로세스에 대하여 선택하여 보이도록 할 수 있다.

이 기능을 응용하면 악성코드 분석 시에 악성코드 프로세스를 설정하면 C&C 서버등으로 요청하는 정보들을 필터링 할 수 있다.



Show only Internet Explorer traffic

➔ IE 브라우저에 대해서만 보이도록 한다.

Hide traffic from Service Host

➔ svchost.exe 에 관한 Rss Feeds 나 백그라운드로 실행되는 네트워크 패킷을 안보이게 한다.

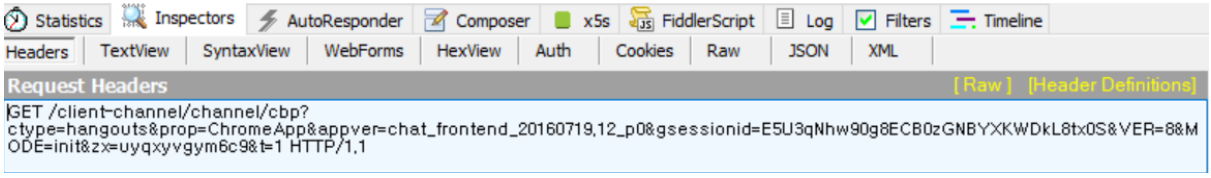
3.4.4. Request Headers 속성

Request Headers 는 아래 그림과 같이 5 가지로 분류 된다.

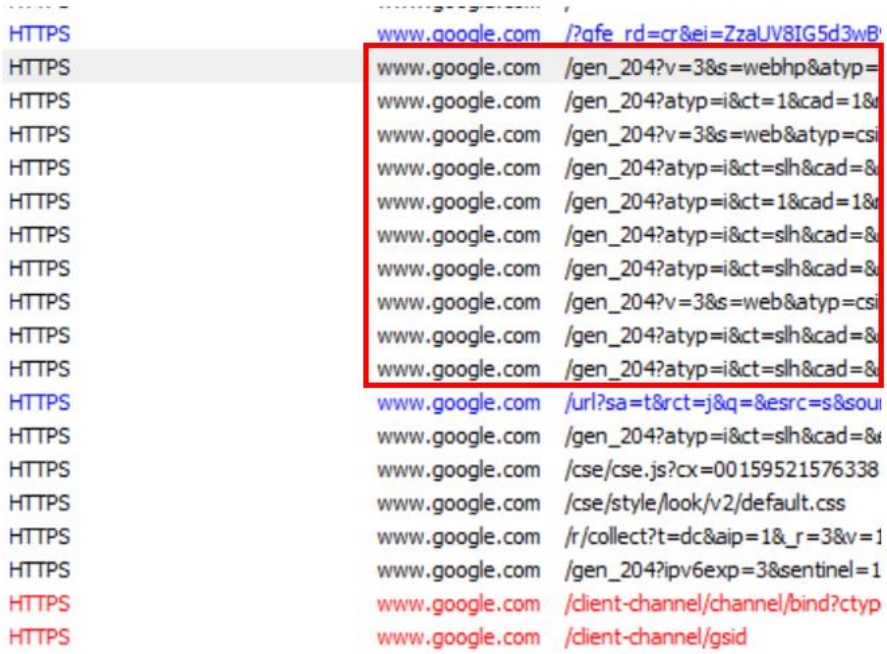
1~4 번까지는 Request Headers 의 내용을 입력하여 Show, Hide, Flag, Delete 등을 할 수 있다.

내용은 아래 그림에 나와있는 Request Headers 정보의 일부분을 입력하거나 정규표현식을 입력하여 해당 내용을 필터링 할 수 있다.

5 번은 1~4 번과는 다르게 URI 요청시 setting 된 해더값으로 바뀌어서 전송이 된다.

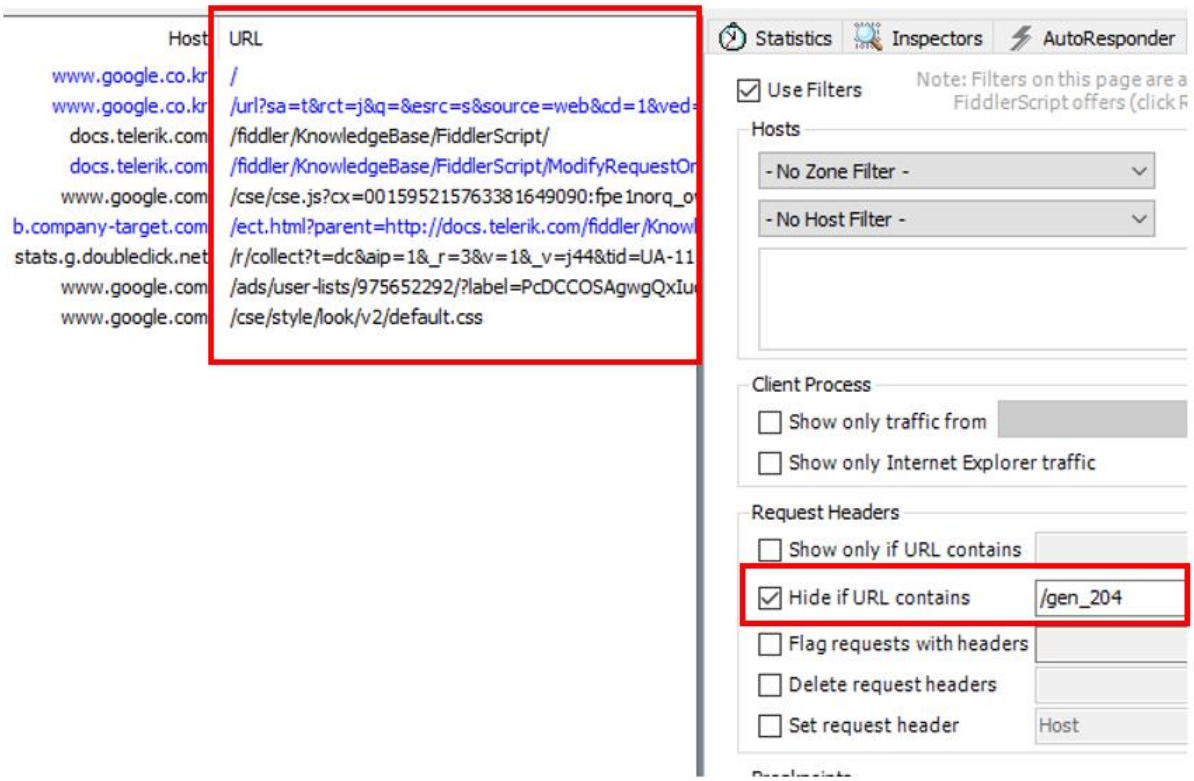


아래 그림에 나와 있는 /gen_204 에 대한 항목을 필터링 해보자.

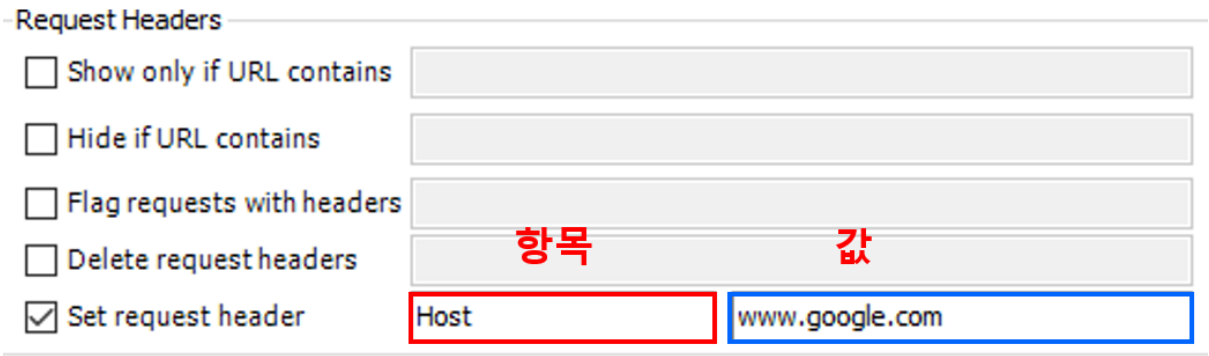


해당 항목 클릭후 Request Headers 내용을 파악한다.

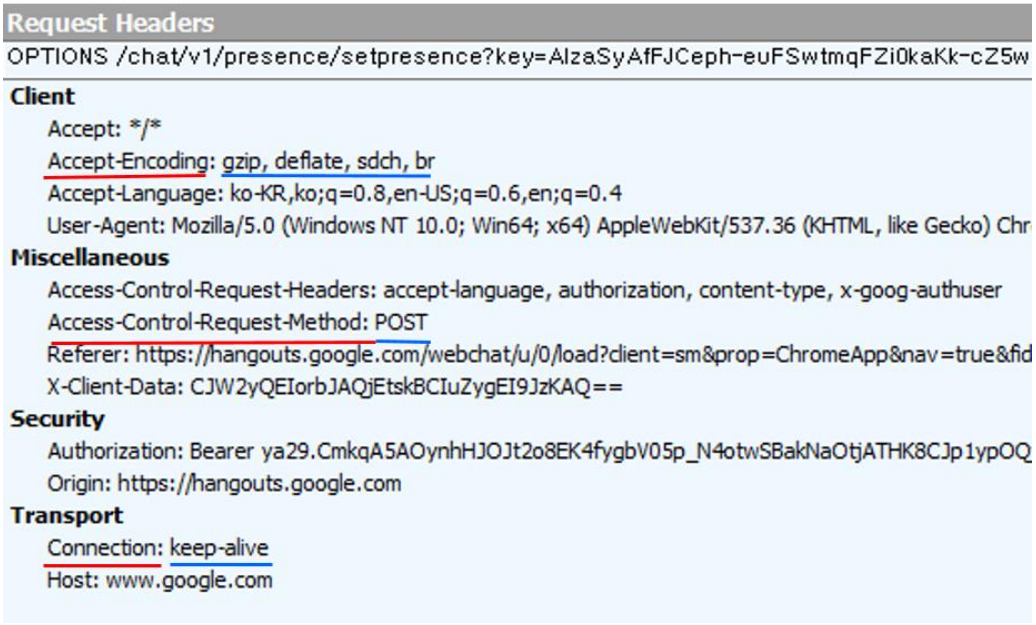
Hide if URL contains 체크 후 /gen_204 입력 후 필터링 시작하면 아래 그림과 같이 /gen_204 가 포함된 패킷은 안보이게 된다.



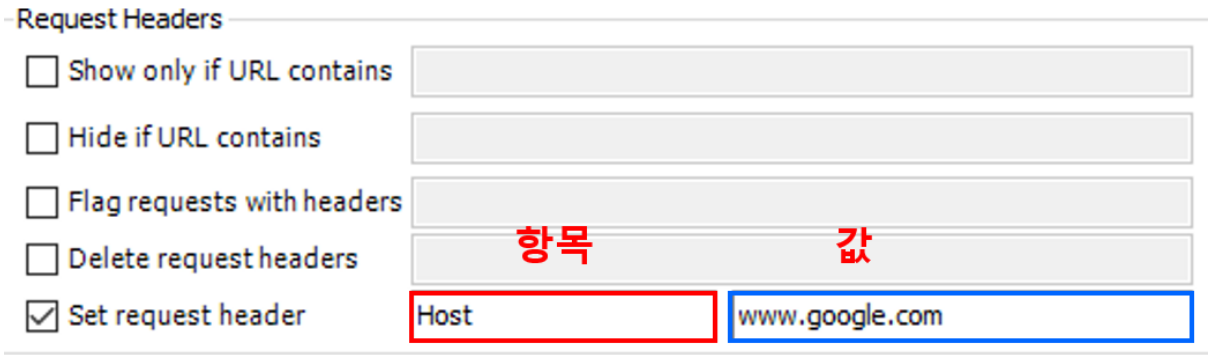
Set request header 는 왼쪽 박스가 항목이고, 오른쪽 박스가 값이다.



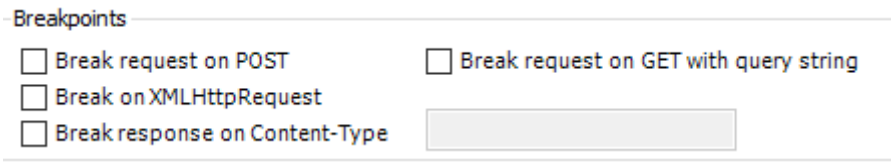
항목과 값은 request header 에 있는 것을 의미한다. 아래 그림을 보면 OOO : OOO 형식으로 나와 있다. 왼쪽 OOO 이 항목을 나타내며 오른쪽 OOO 이 값을 나타낸다. 헤더에 나와 있는 모든 OOO : OOO 형식에 대해 적용이 가능하다.



따라서 아래 항목과 같이 설정하면, 어떤 request 든 Host:www.google.com 으로 설정하게 되고 URL 에 네이버등을 입력하여도 항상 www.google.com 으로 접속하게 된다.



3.4.5. Breakpoints 속성



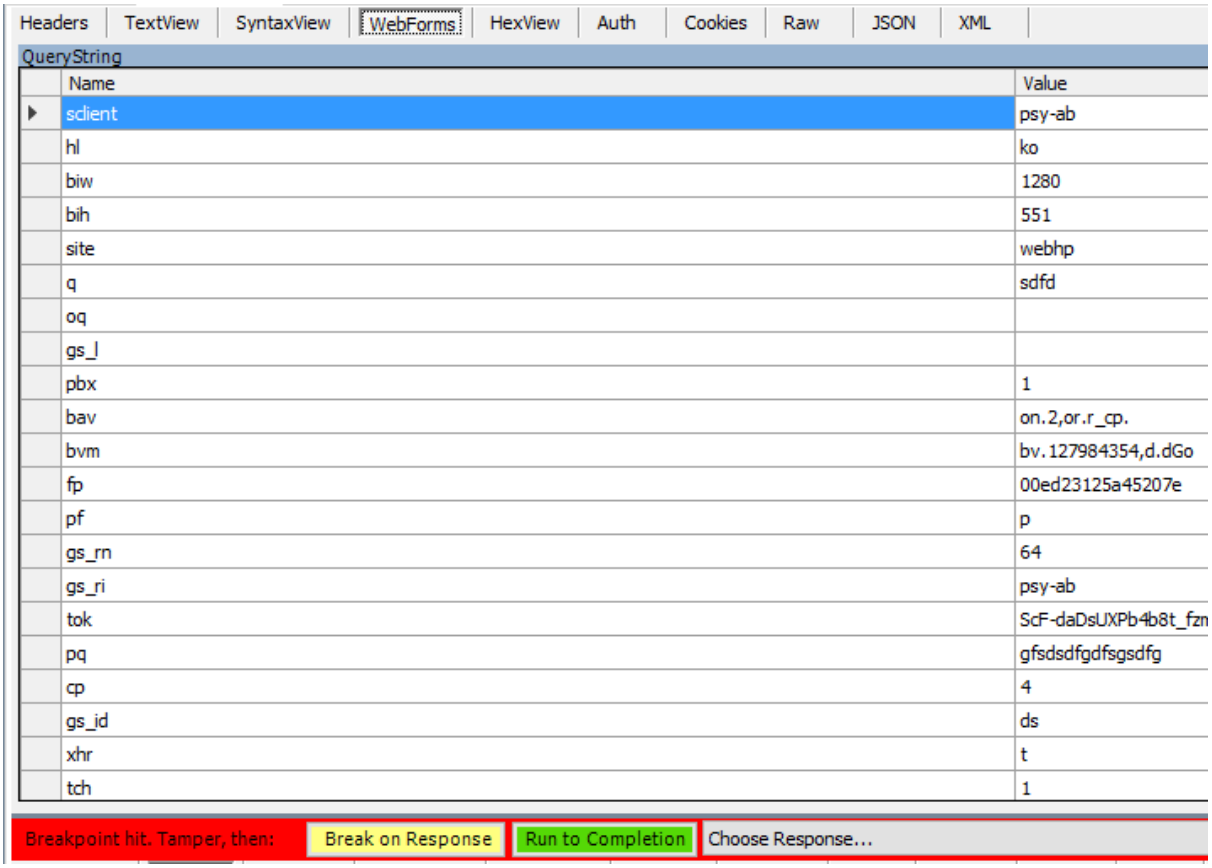
Break request on Post

➔ POST 요청 전송 전에 break 된다.

Break request on GET with query string

➔ GET with 파라마미터 요청 전송 전에 break 된다.

Break request on GET with query string 필터링을 설정하면 아래 그림과 같이 break 가 되고 파라미터 값들을 변경하여 전송할 수 있다.



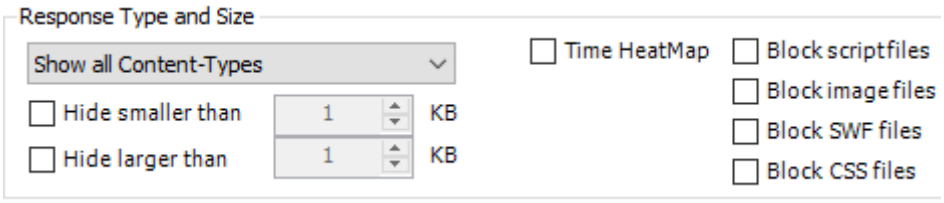
Break response on Content-Type

➔ Content-Type 에 입력된 값들이 response 패킷에 포함되어 있는 경우 break 한다.

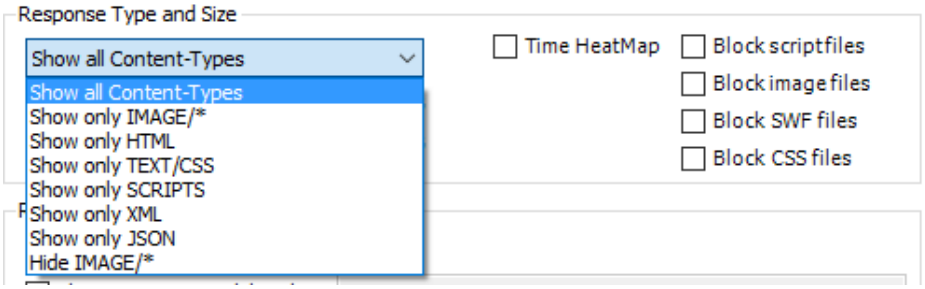
3.4.6. Response Status Code 속성

Break status code 에 대하여 숨김 설정 가능

3.4.7. Response Type and Size 속성

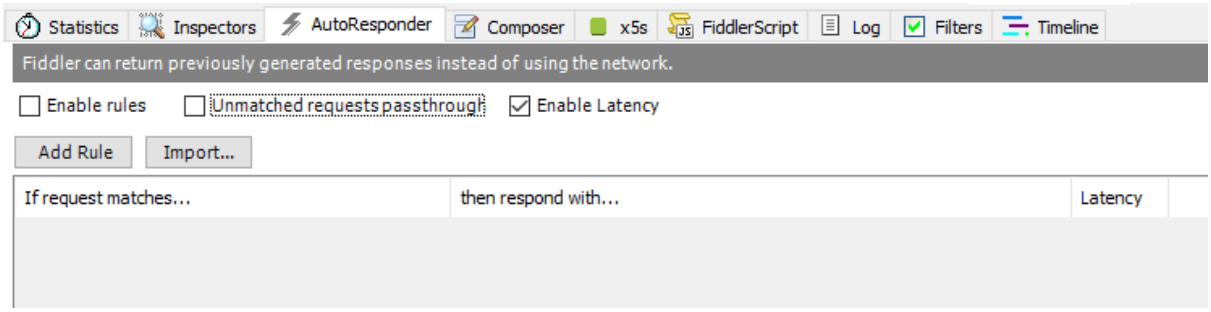


Response Type 및 size 에 대한 다양한 패킷 필터링 가능



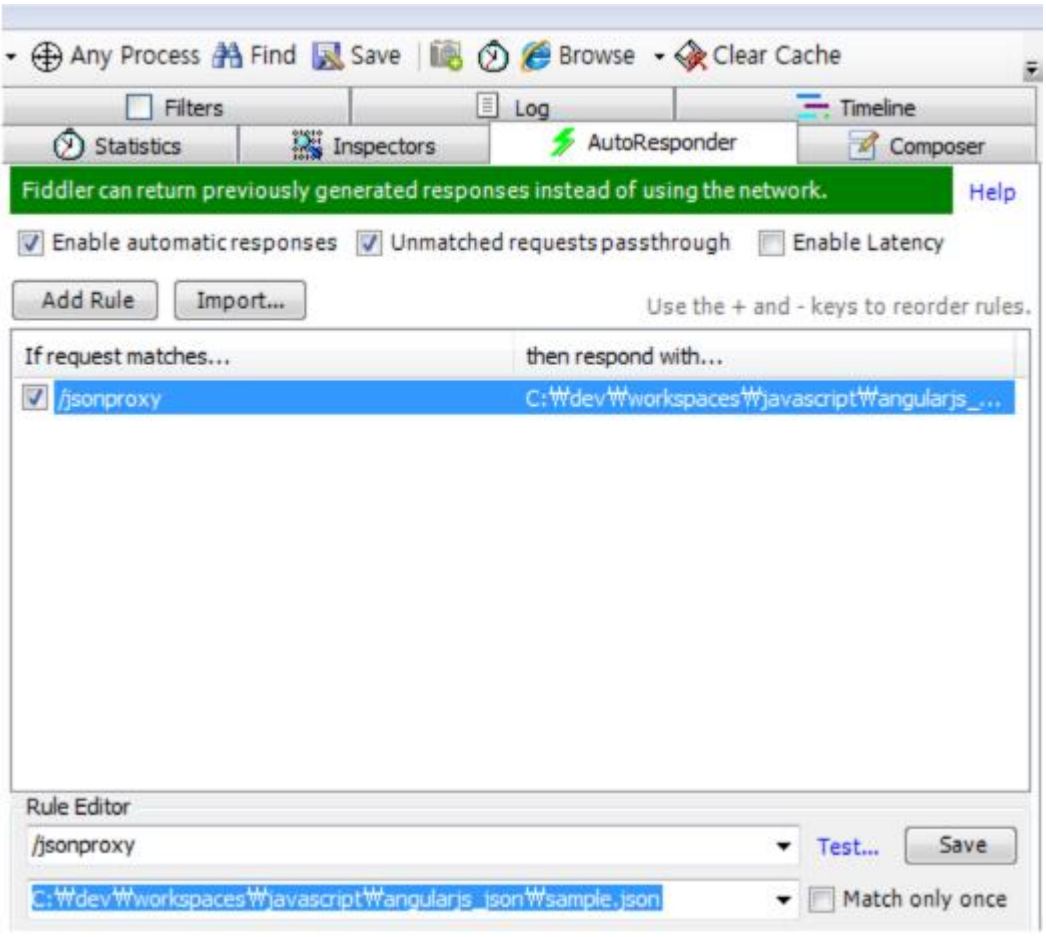
3.5. AutoResponder 탭

Response 특정 URL 패턴으로 들어오는 Request 를 가로채서 특정 Response 를 보내게 만드는 기능이다.



설정에 따라 다양한 기능을 할 수 있으며, 이번에 소개할 것은 이 기능을 이용하여 간단한 JSON 클라이언트를 세팅하는 과정을 소개하겠다.

아래 그림과 같이 /jsonproxy 가 포함된 URI 에 대하여 c:₩dev₩..json 의 응답을 하도록 설정하였다.



RuleEditor 에서 /jsonproxy 라는 URL 로 설정해놓으면, 모든 사이트로 가는 Request 에 대해 /jsonproxy 라는 URI 를 가지면 모두 이 Rule 을 적용 받는다. 그리고, 아래에, Response 를 선택할 수 있는데, 여기서는 특정 JSON 파일을 만들어서 선택하였다.

sample.json 파일의 내용은 아래와 같다.

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
Content-Length: 57
{
        "data":{
                "name":"Terry",
                "city":"Seoul"
                }
}
```

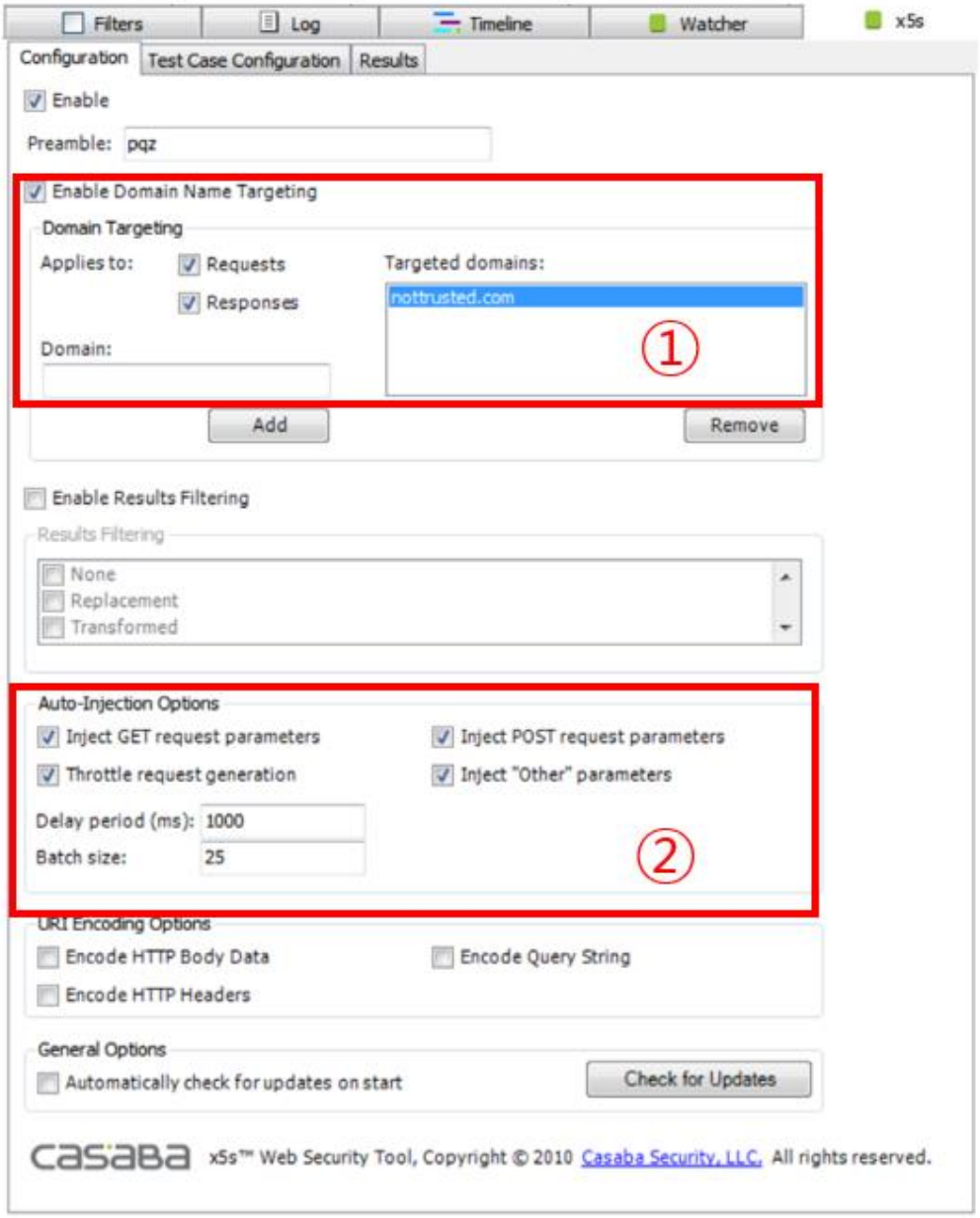
마지막으로 위의 AutoResponder 메뉴에서 "Enable automatic response" 체크 박스를 선택해주면. /jsonproxy 로 가는 모든 Request 에 대해서, 위의 파일에 저장된 값을 리턴해준다.

4. 활용 방안

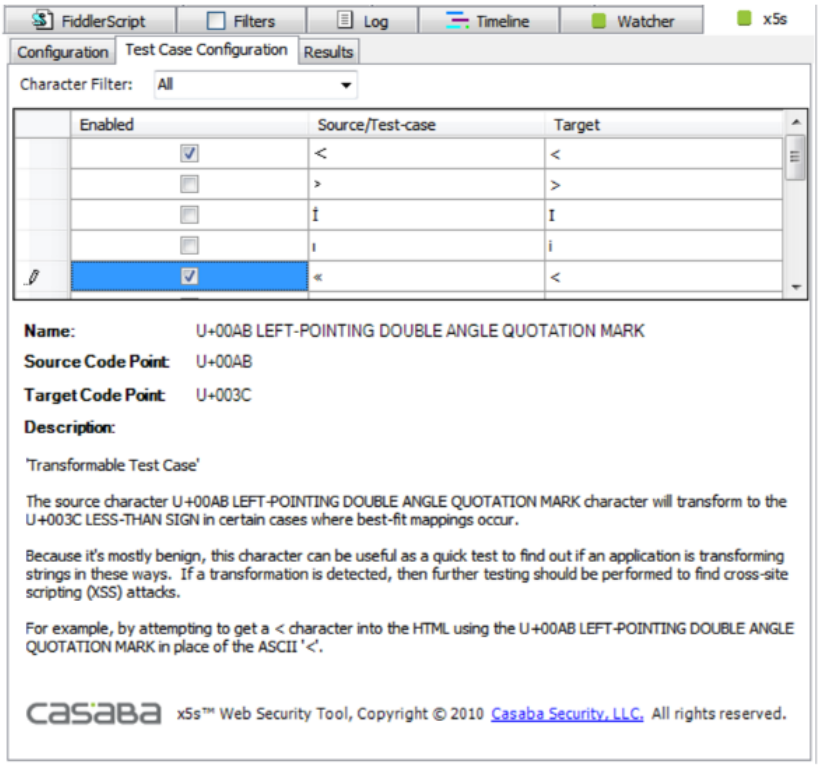
4.1. X5S 를 활용한 XSS 취약점진단

x5s 는 casaba 에서 제작한 fiddler 플러그인이다. https://xss.codeplex.com/를 통해 다운로드 및 설치가 가능하다.

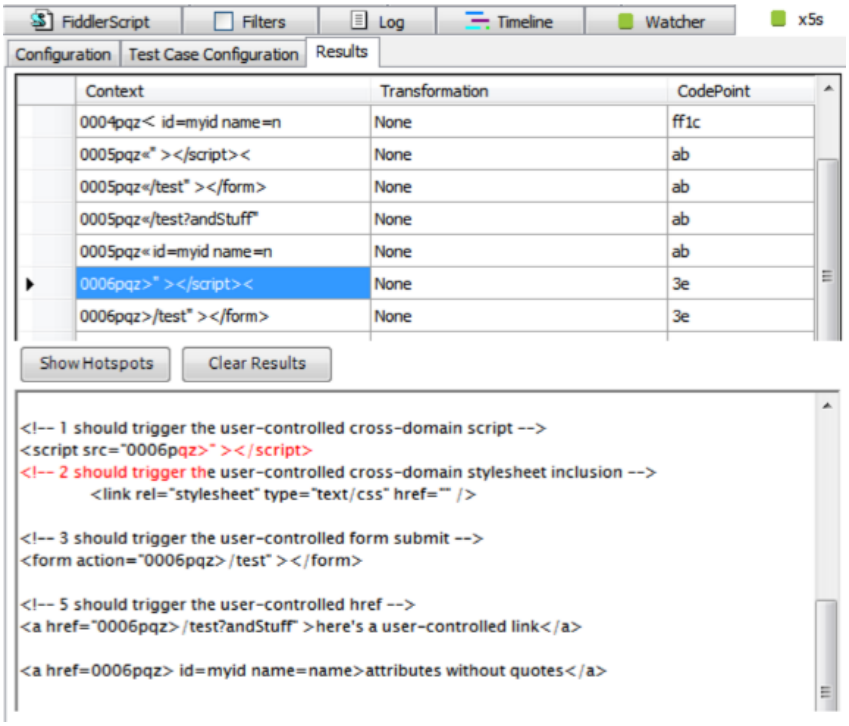
아래 그림은 x5s 탭 선택화면이다. 1 번 박스에는 공격할 Domain 을 입력해주면 된다. 2 번 박스에는 Auto Injection 옵션을 체크 해주면 된다. 1 번 박스 밑에 있는 enable Results Filtering 을 통해 결과값에 대하여 필터링을 할 수 있다.



아래 그림은 Test case 선택 화면으로 테스트를 할 구문을 체크해주어야 한다. XSS 테스트를 위해서는 < , > , ' , " 항목이 포함되어 있어야 한다.



아래 그림은 Test 후 결과 화면이며, 오탐 진탐 여부를 확인해봐야 한다.



4.2. 웹 보안 프로그램 회피

ActiveX 를 사용하고(별도의 프로그램을 설치), 이 ActiveX 를 이용하여 개별 컴퓨터를 조작 하여 원하는 동작을 수행하는 웹 보안 프로그램에 대하여 회피하는 방안입니다.

4.2.1. 웹 보안 프로그램의 일반적인 패턴

Web 이라는 기술에는 script 기술이 허용이 되어, 매우 능동적인 동작이 가능하게 되어있습니다. 즉 프로그래밍 기법이 적용 되고 있습니다.

웹(Web) 보안 프로그램은 이 script 를 이용해, 개별 ActiveX 를 설치, 설치 여부 확인, 설치한 기능 실행을 수행 합니다.

자 그렇다면, 보안 프로그램의 설치, 설치여부 확인, 설치 기능 실행을 막으면 우리가 원하는 목적을 달성 할 수 있습니다. 테스트 하는 웹 보안 프로그램은 하나의 함수만 실행을 막아 버리면, 설치확인, 설치까지 막혔습니다.
왜냐하면 script 가 있고, 결국 이 script 가 실행 되어야 하는데, 이 실행 기회를 막아 버리면 아주 쉽게 목적이 달성 된다는 것입니다.

**중요 포인트

HTML 은 code 가 위에서, 아래로 쭈욱~ 실행이 됩니다.
즉 Script 가 위치하고, 호출 하는 부분이 있다면, 해당 부분의 스크립트가 실행이 되고 난 뒤에 아래에 나오는 HTML 혹은 추가 스크립트가 실행이 됩니다.
웹 보안 프로그램은 보통 BODY 가 처리 되기 전 단계인 HEAD 혹은 BODY 맨 윗 부분에 위치하여 해당 page 가 보이기 전에, 막음 처리 등등을 하게 됩니다.

4.2.2. 공략 대상의 관련 코드와 구성

아래 그림은 비보안 페이지의 소스 코드 구성입니다.

```
<HTML>
<HEAD>
<SCRIPT> ~ </SCRIPT>
</HEAD>
<BODY>
~
~
</BODY>
</HTML>
```

아래 그림은 보안 페이지의 소스 코드 구성입니다.

```
<HTML>
<HEAD>
<SCRIPT> ~ </SCRIPT>
</HEAD>
<BODY>
~
<SCRIPT>
  웹 기반 보안 프로그램을 위한 기괴한 스크립트들이 와장창 위치 합니다.
  괴물용-변수선언
  괴물용-변수선언들 ...
  ......
  괴물용-함수선언
  괴물용-최초동작을위한 함수
  괴물용-함수선언들 ...
  ......
  괴물용-최초동작을위한 함수호출!      [--> 위에 등장한 '괴물용-함수선언' 중의 하나로 호출 합니다.]
</SCRIPT>


~
</BODY>
</HTML>
```

웹기반 보안프로그램이 포함된다면 옆 그림과 유사하게, HEAD section 혹은 BODY section 에 '괴물 code'가 위치하게 됩니다. (은행 등등의 솔루션은 별도 script URL 에서 읽어 옵니다. 결국 구성은 비슷 합니다)

4.2.3. Fiddler 를 이용한 실시간 HTML 변경 처리하기

결국 웹보안 회피는 웹보안 기동 함수를 찾아서 해당 함수 호출을 막으면 됩니다. 즉 앞서 설명 드렸던 "괴물용-최초동작을 위한 함수호출 1" 이 부분만 확인하고 Fiddler 에 "Rule"을 추가하여 '없는 것으로' 처리 하면, 아주 깔끔하게 마무리가 됩니다.

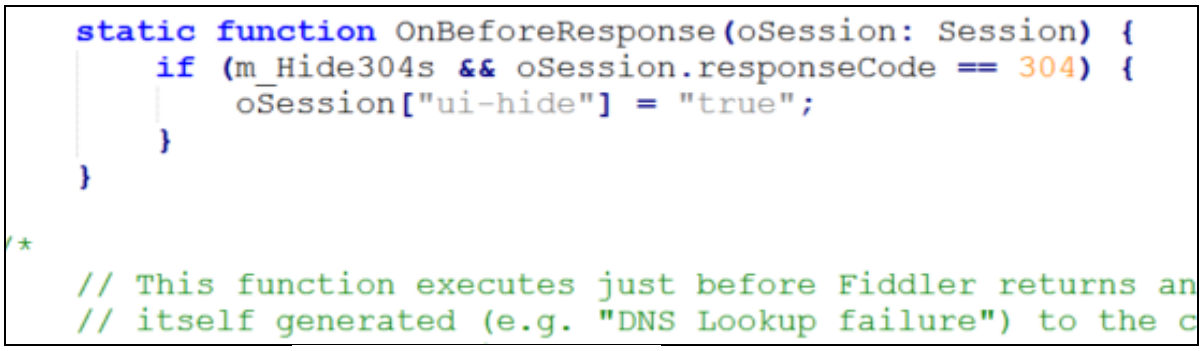
공략 대상 보안프로그램은 아래와 '유사'하게 구성되어 있습니다.

SecurePage();

즉 위 함수가 <BODY> 아래 부터 나오는 요상한 스크립트에 정의 되어 있고, 해당 스크립터 덩어리의 맨 아래 부분에 보면 호출 하는 것으로 끝을 맺게 됩니다.

최신 IE 등 에서 제공하는 **스크립터 디버깅** 기능을 이용해서 의심가는 함수에 break 를 걸고, skip 하여 동작을 사전 확인도 가능 합니다.

1) ctrl + R 입력을 통해 CustomRules page 접속



```
    static function OnBeforeResponse(oSession: Session) {
        if (m_Hide304s && oSession.responseCode == 304) {
            oSession["ui-hide"] = "true";
        }
    }

/*
    // This function executes just before Fiddler returns an
    // itself generated (e.g. "DNS Lookup failure") to the c
```

2) OnBeforeResponse 함수 안에 에 아래 코드를 입력

```
if (oSession.HostnameIs("www.공략대상.사이트") ) {
     oSession.utilDecodeResponse();
     oSession.utilReplaceInResponse('SecurePage();',' ');
}
```

코드 설명

1) HostnameIs() → www.공략대상.사이트 인 패킷이 존재하면 함수 내부로 들어감
2) utilDecodeResponse() → Response packet 을 디코딩함
3) utilReplaceInRespone() → Response packet 안에 SecurePage() 함수를 공백으로 대체

4.3. CutomRules 로 특정 패킷 저장

```
import System;
import System.Windows.Forms;
import Fiddler;
import System.IO;
```

< 헤더 선언 >

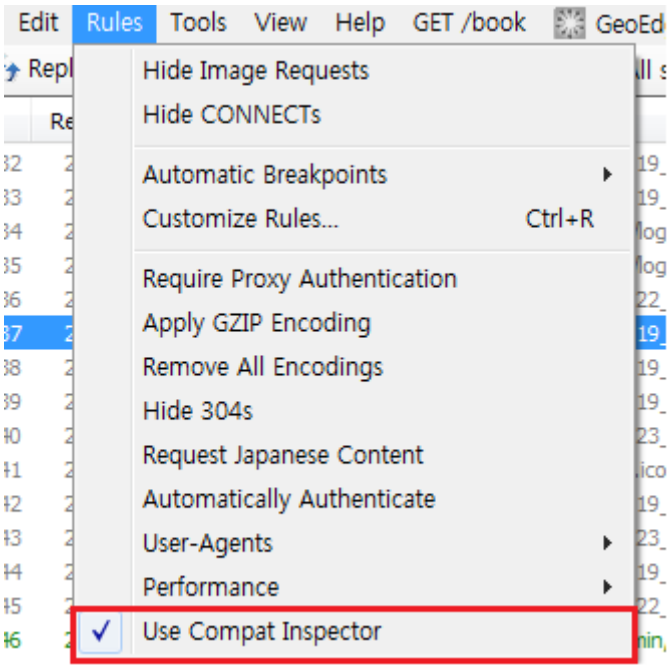
아래 코드는 OnBeforeResponse 응답 패킷을 받기 전이며, 응답 패킷의 상태코드가 error 코드일 경우, Naver.com 하위 사이트의 세션이서 발생한 경우 해당 내용을 파일로 저장해주는 코드 입니다.

```
static function OnBeforeResponse(oSession: Session) {
        if (m_Hide304s && oSession.responseCode == 304) {
            oSession["ui-hide"] = "true";
        }

        if (oSession.responseCode >= 400) {
            if (oSession.hostname.indexOf("naver.com") != -1) {
                var sw = File.AppendText("C:₩₩400Error₩₩response_error.txt");
                sw.WriteLine('[' + now_time() + '] ' + oSession.responseCode + ' ' + oSession.url);
                sw.Close();
            }
        }
 //now_time 함수 추가
        static function now_time() {
                var time_t = new Date();
                var s = set_standard(time_t.getFullYear(), 4) + '-' +
                        set_standard(time_t.getMonth() + 1, 2) + '-' +
                        set_standard(time_t.getDate(), 2) + ' ' +
                        set_standard(time_t.getHours(), 2) + ':' +
                        set_standard(time_t.getMinutes(), 2) + ':' +
                        set_standard(time_t.getSeconds(), 2);
                return s;
        }
        static function set_standard(time, digits) {
                var zero = '';
                time = time.toString();

                if (time.length < digits) {
                        for (var i = 0; i < digits - time.length; i++)
                                zero += '0';
                }
                return zero + time;
        }
```

4.4. CustomRules 로 Script Injection 하기

4.4.1. 피들러 Rules 메뉴에 항목 추가 하기

아래 사진과 같이 항목을 추가해서 체크되어 있을 때는 Script injection 이 되고, 체크해제 되어 있을 때는 안되도록 만들려고 합니다.



UI 메뉴를 넣는 코드 입니다.

"Use Compat Inspector" 라는 메뉴를 생성하고 디폴트 체크해제 상태로 둡니다.

```
        public static RulesOption("Use Compat Inspector")
        var m_UseCompatInspector : boolean = false;
```

4.4.2 스크립트 삽입하기

본인이 원하는 타이밍에 함수를 삽입합니다. 제 경우엔 Request 를 받기 전 스크립트 삽입하기 위해 아래와 같이 작성했습니다.

```
static function OnBeforeRequest(oSession : Session) {
        InjectInspectorScript(oSession);
```

그리고 InjectInspectorScript 함수를 작성해 줍니다.

```
static function InjectInspectorScript(oSession : Session) {
        if (!m_UseCompatInspector)
                return;
        // Ensure we only inject into HTML
        if (oSession.url.EndsWith(".js"))
                return;
        if (oSession.url.EndsWith(".css"))
                return;
        if (!oSession.oResponse.MIMEType.Contains("text/html"))
                return;
        // Retrieve the response body
        var responseBody = oSession.GetResponseBodyAsString();
        // One final check to ensure the content looks like HTML
        if (!/^₩uFEFF?₩s*</.exec(responseBody ))
                return;
        // Prepare to inject
        var pos = 0; // Initial position is start of document
        // Locate important elements in the page
        var doctype = responseBody .IndexOf("<!doctype",
StringComparison.OrdinalIgnoreCase);
        var meta = responseBody .IndexOf("X-UA-Compatible",
StringComparison.OrdinalIgnoreCase);
        var script = responseBody .IndexOf("<script", StringComparison.OrdinalIgnoreCase);
        // Place after doctype (if present)
        if (doctype != -1)
```

```
                        doctype = responseBody.IndexOf(">", doctype + 1);
                if (doctype != -1 && doctype != responseBody .Length - 1)
                        pos = doctype + 1;
                // Place after first X-UA-Compatible meta tag (if present)
                if (meta != -1)
                        meta = responseBody.IndexOf(">", meta + 1);
                if (meta != -1 && meta != responseBody.Length - 1)
                        pos = meta + 1;
                // Place before any script tags that occur before the current position (if present)
                if (script != -1 && script < pos)
                        pos = script;
                // Perform the injection at the detected location
                oSession.utilSetResponseBody(
                        responseBody.Insert(pos, "<script
src='http://ie.microsoft.com/testdrive/HTML5/CompatInspector/inspector.js'></script>"));
        }
}
```

제일 처음 boolean 체크하는 부분과,

제일 아랫부분 responseBody.Insert 부분만 보시면 됩니다.

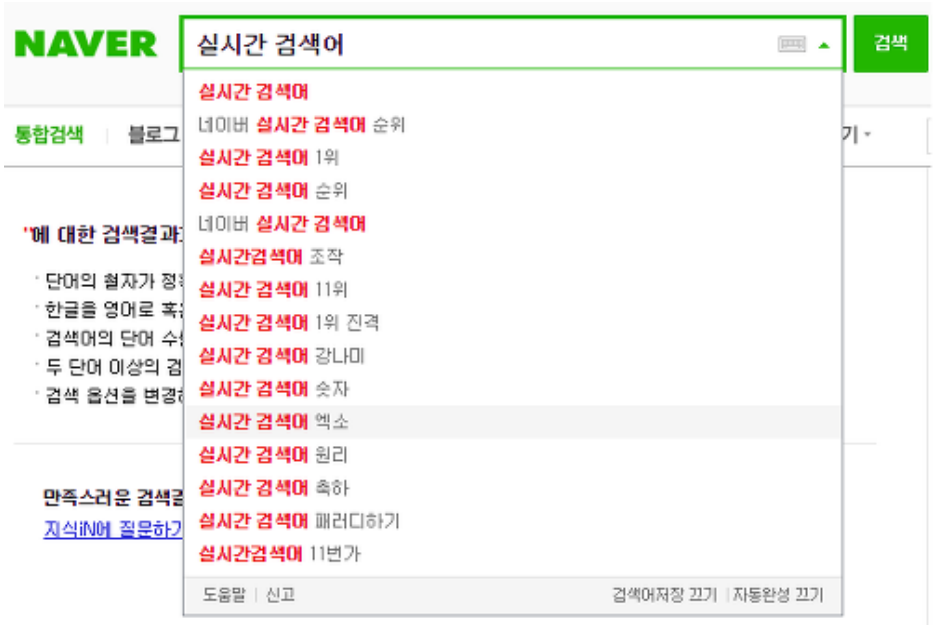
이후 Use Compat Inspector 메뉴에 체크하면 응답 패킷의 바디에 js 가 적용되어 그에 따라 동작하게 됩니다.

4.5. 패킷 바꿔치기

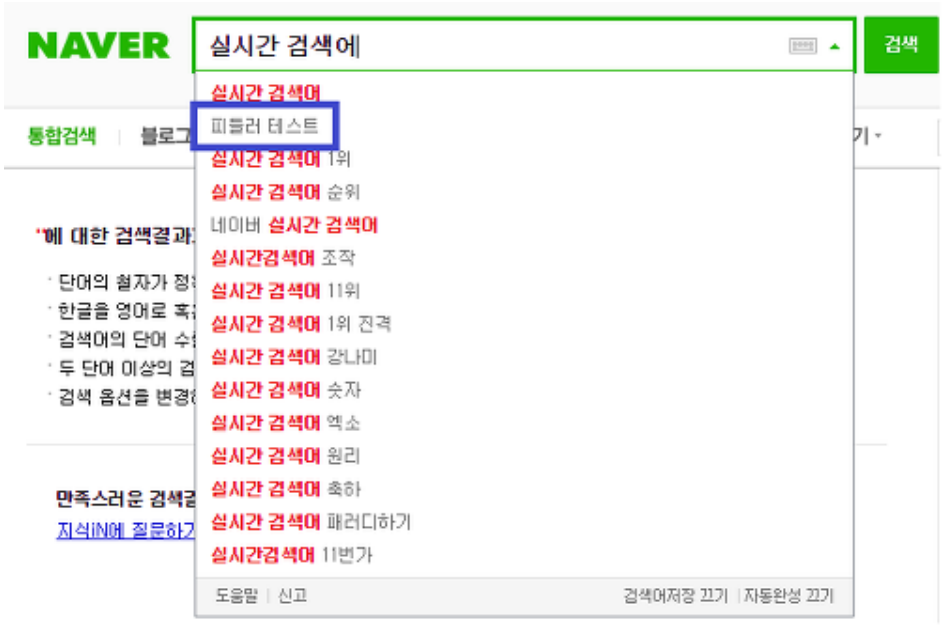
피들러의 AutoResponder 탭을 이용합니다.

특정 요청을 했을 때 실제 서버에서 내려온 응답 패킷으로 웹 브라우저를 구성해서 보여주는게 아니라 AutoResponder 탭에서 변경한 패킷으로 웹 브라우저에 표시되도록 하려고 합니다.

현재 네이버에서 "실시간 검색어"를 입력하면 아래와 같이 보이는데요.

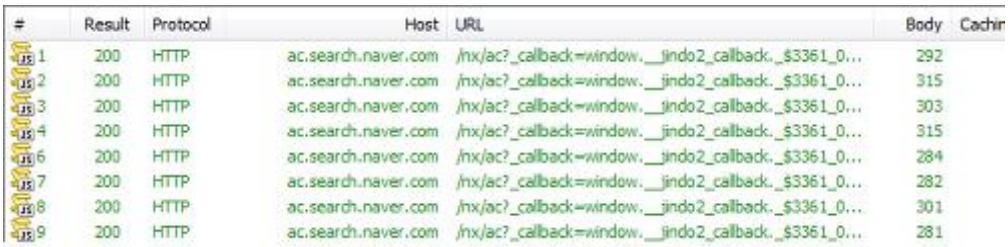


이것을 제가 바꿔치기 한 데이터로 아래와 같이 보이게 하겠습니다.



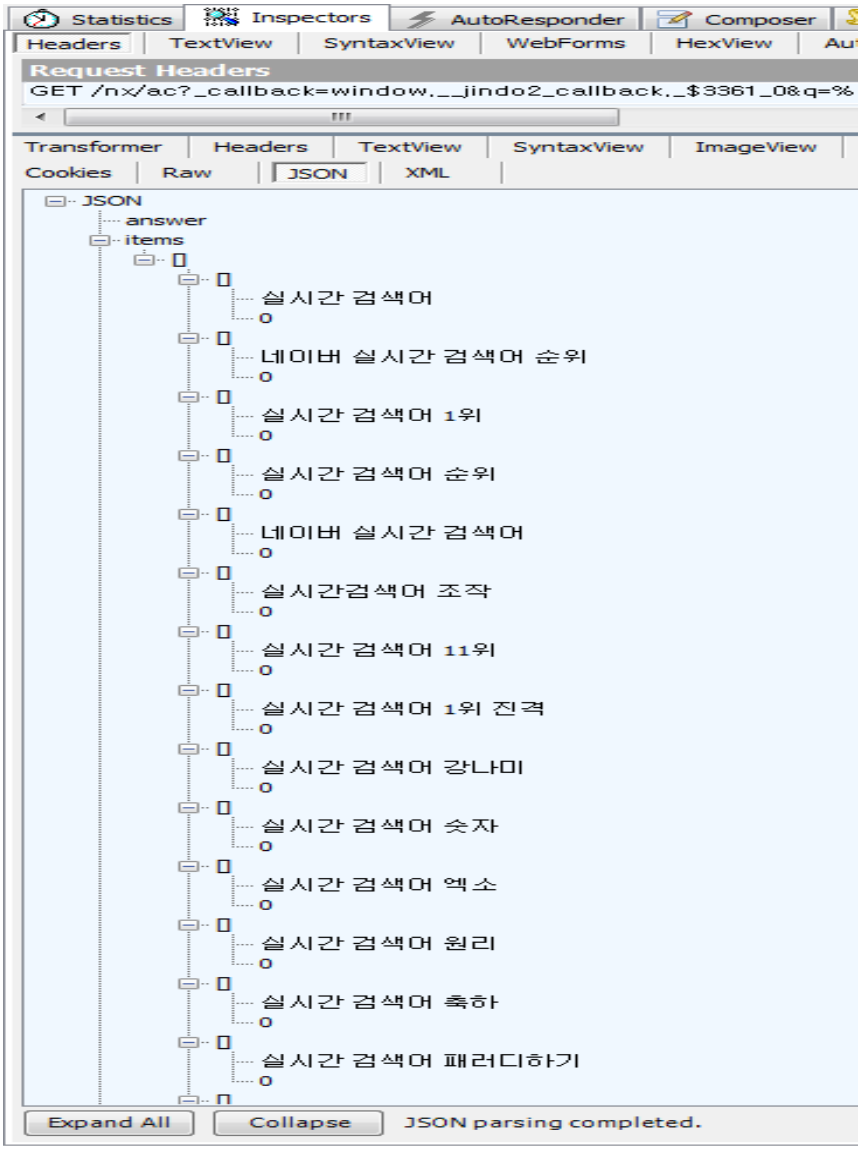
4.5.1 원본 데이터가 어떻게 넘어오는지 확인

네이버에서 검색창에 검색어를 입력하면 입력한 문자가 바뀔 때(?) 마다 패킷이 전송되는 것을 볼 수 있습니다.



#	Result	Protocol	Host	URL	Body	Cachir
1	200	HTTP	ac.search.naver.com	/nx/ac?_callback=window.__jindo2_callback._$3361_0...	292	
2	200	HTTP	ac.search.naver.com	/nx/ac?_callback=window.__jindo2_callback._$3361_0...	315	
3	200	HTTP	ac.search.naver.com	/nx/ac?_callback=window.__jindo2_callback._$3361_0...	303	
4	200	HTTP	ac.search.naver.com	/nx/ac?_callback=window.__jindo2_callback._$3361_0...	315	
6	200	HTTP	ac.search.naver.com	/nx/ac?_callback=window.__jindo2_callback._$3361_0...	284	
7	200	HTTP	ac.search.naver.com	/nx/ac?_callback=window.__jindo2_callback._$3361_0...	282	
8	200	HTTP	ac.search.naver.com	/nx/ac?_callback=window.__jindo2_callback._$3361_0...	301	
9	200	HTTP	ac.search.naver.com	/nx/ac?_callback=window.__jindo2_callback._$3361_0...	281	

이걸 Inspectors 에서 보면 Json 데이터를 응답으로 받고 있는 것을 확인할 수 있습니다.



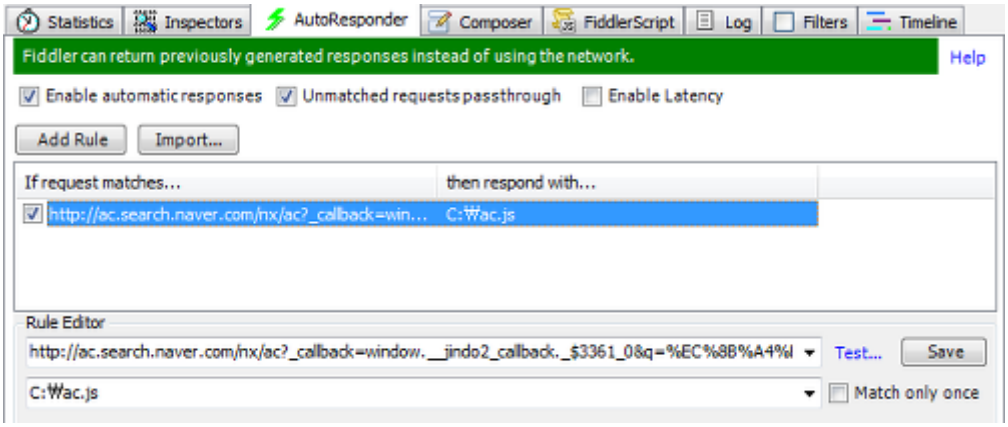
4.5.2 원본 데이터를 변경해서 로컬에 저장합니다.

원하는 데이터를 변경해서 아래 내용을 "ac.js"라는 파일로 저장해둡니다.

```
window.__jindo2_callback._$3361_0({
"query" : ["실시간 검색어", "실시간 검색어"],
"answer" : [],
"nature" : [],
"items" : [
[["실시간 검색어","0"],["피들러 테스트","0"],["실시간 검색어 1 위","0"],["실시간 검색어 순위","0"],["네이버 실시간 검색어","0"],["실시간검색어 조작","0"],["실시간 검색어 11 위","0"],["실시간 검색어 1 위 진격","0"],["실시간 검색어 강나미","0"],["실시간 검색어 숫자","0"],["실시간 검색어 엑소","0"],["실시간 검색어 원리","0"],["실시간 검색어 축하","0"],["실시간 검색어 패러디하기","0"],["실시간검색어 11 번가","0"]],
[],
[],
[],
[],
[]
]
})
```

4.5.3 AutoResponder 탭에 Rule 을 추가합니다.

세션 리스트에서 해당 세션을 드래그앤 드랍으로 항목에 끌어다 놓고 respond with 를 수정한 뒤 저장합니다.

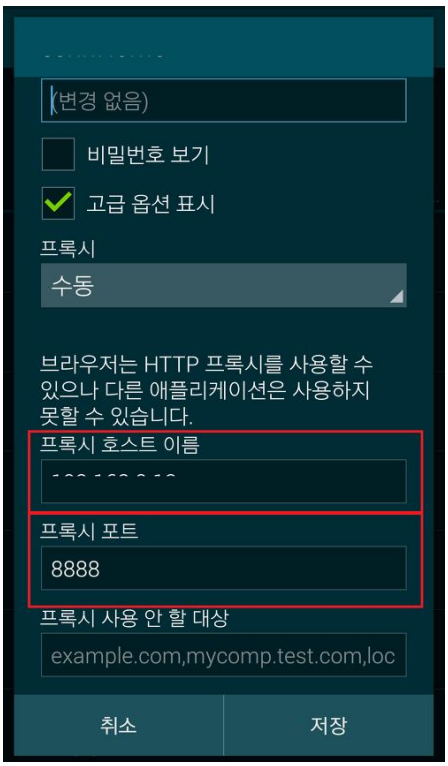


4.5.4 변경된 데이터로 웹 브라우저가 표시되는지 확인

4.6. 모바일 패킷 확인하기

4.6.1 같은 AP 대역을 사용하고 있을 때 – 프록시 설정

모바일에서 wifi 프록시 설정을 해줍니다. 프록시 호스트 이름은 PC 에서 ipconfig 로 조회한 IP 주소를 입력합니다. 프록시 포트는 피들러에서 설정한 포트를 입력합니다.

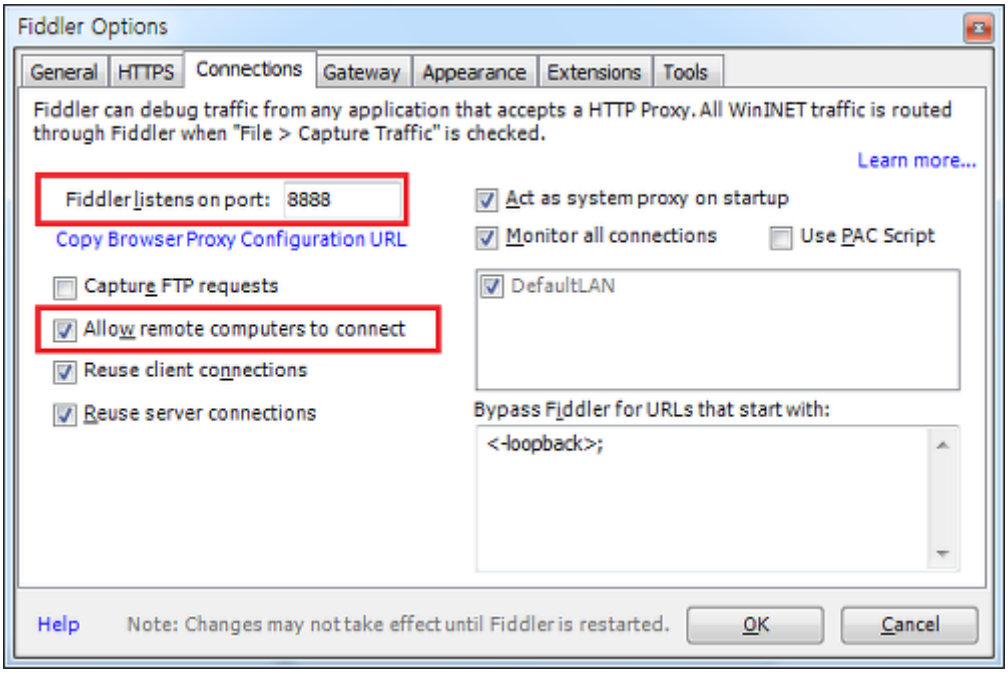


피들러에서 설정을 해줍니다.

상단 Tools > Fiddler Option.. 에서 Connections 탭 입니다.

포트 번호는 모바일의 포트번호와 동일하게 기입해주시고,

Allow remote computers to connect 옵션에 체크해줍니다.



이후 모바일의 패킷을 PC 의 피들러에서도 확인하실 수 있습니다.

4.6.2 USB 를 이용해서 모바일 패킷을 확인할 때

사내 네트워크 같은 경우 모바일과 PC 를 같은 AP 대역을 사용할 수 없을 때가 있습니다.
그럴 때 크롬의 port 포워딩을 써서 fiddler 에서 패킷을 받도록 합니다.

모바일과 PC 를 USB 로 연결합니다.

PC 에서 크롬을 열고 chrome://inspect/#devices 화면으로 들어갑니다.

모바일에서도 chrome 을 열면, chrome://inspect/#devices 화면에 디바이스 정보가 보입니다.

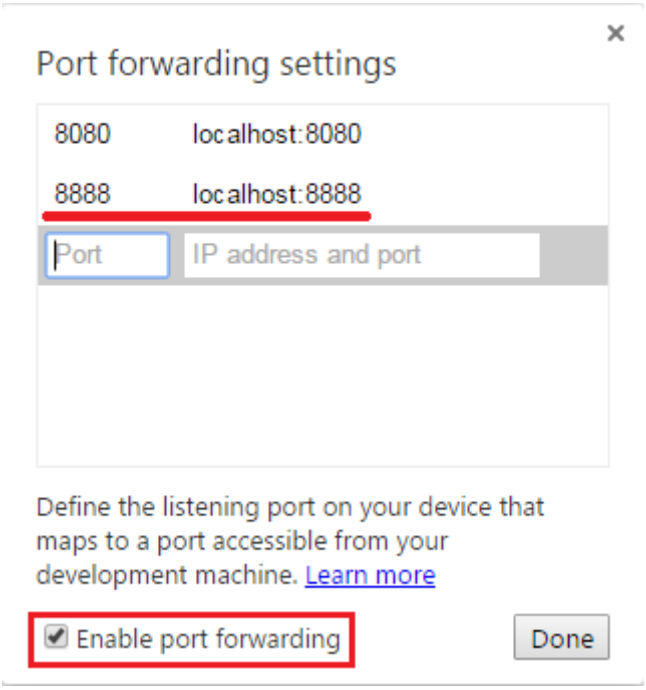
만약 크롬에서 모바일 정보가 보이지 않을 경우,

1. sdk 를 설치하셨다면 adb devices 로 디바이스가 PC 와 제대로 연결됐는지 확인합니다.

2. PC 에서 크롬을 시크릿 모드로 실행한 건 아닌지 확인합니다.

chrome://inspect/#devices 페이지에서

상단 "Port forwarding..." 버튼을 눌러 포워딩할 포트를 설정해줍니다.



4.6.1.와 같이 모바일과 프록시 설정과 PC 피들러 설정을 하되,

모바일 프록시 호스트 이름을 크롬에서 지정한 localhost 로 저장합니다.

이후 모바일의 패킷을 PC 의 피들러에서도 확인하실 수 있습니다.

위와 같이 지정할 경우, 모바일 패킷이 PC 를 거치기 때문에

PC 의 호스트 설정이나 피들러 설정을 따릅니다.

5. 부록

5.1. 피들러로 할 수 있는 일 & 없는 일

▶ 피들러로 할 수 있는 일

⊙ 브라우저, 클라이언트 프로그램, 서비스 등에서 오고 가는 웹 트래픽을 볼 수 있다.

⊙ 자동 또는 수동적인 방법으로 어떠한 응답이나 요청도 수정할 수 있다.

⊙ HTTPS 트래픽을 복호화하여 살펴보거나 수정하는 것도 가능하다.

⊙ 캡쳐한 트래픽을 저장하여 보관해 두고 나중에 불러올 수도 있다. 다른 컴퓨터의 트래픽도 이런 식으로 다룰 수 있다.

⊙ 클라이언트 프로그램으로 가는 응답을 이전에 캡쳐해 두었다면 서버가 오프라인 상태라 해도 응답을 재현할 수 있다.

⊙ 맥/리눅스 시스템, 스마트폰, 태블릿 컴퓨터 등을 포함한 대부분의 PC 와 모바일 기기에서 발생한 웹 트래픽을 디버깅할 수 있다.

⊙ TOR 네트워크 등 프록시 서버에 업스트림(upstream)을 연결(chain)할 수 있다.

⊙ 클라이언트 컴퓨터나 모바일 기기를 다시 설정하지 않고도 역 프록시(reverse proxy)로 동작하여 트래픽을 캡쳐할 수 있다.

⊙ 피들러 스크립트나 .NET 기반 확장 모듈을 통해 새로운 기능을 추가할 수 있어 더 강력해질 수 있다.

▶ **피들러로 할 수 없는 일**

⊙ 웹 프로토콜 트래픽이 아닌 경우에는 디버깅할 수 없다.

→ 피들러는 HTTP, HTTPS, FTP 트래픽 그리고 HTML5 웹소켓(Web-Socket)이나 ICY 스트림과 같은 관련 프로토콜에 대해서만 작동한다.

→ 피들러는 SMTP, POP3, 텔넷, IRC 등의 다른 프로토콜의 트래픽은 볼 수 없다.

⊙ 너무 큰 요청이나 응답은 다룰 수 없다.

→ 피들러는 크기가 2 기가바이트 이상인 요청은 다루지 못한다.

→ 피들러는 크기가 2 기가바이트 이하의 응답만 다룰 수 있다.

→ 피들러는 시스템의 메모리와 페이지 파일을 사용하여 세션 데이터를 저장한다. 목록에 저장한 세션 수가 너무 많거나 요청이나 응답이 너무 크면 성능이 저하될 수 있다.

⊙ "마법처럼" 웹 사이트에 있는 버그를 없애지는 못한다.

5.2. 참고사이트

http://jamesku.tistory.com/entry/%EC%9D%B8%ED%84%B0%EB%84%B7%EC%9C%A0%ED%8B%B8-Fiddler-%EC%82%AC%EC%9A%A9%EB%B2%95-%EC%82%B4%ED%8E%B4%EB%B3%B4%EA%B8%B0HTTP-%ED%8A%B8%EB%9E%98%ED%94%BD
http://www.slideshare.net/taggon/fiddler-27055698
http://www.dsun.kr/23
http://blog.kinesis.kr/99
http://nuts84.tistory.com/28
http://www.codekin.com/?p=390
http://bcho.tistory.com/849
http://www.codekin.com/?p=390
http://egs41.tistory.com/entry/%ED%94%BC%EB%93%A4%EB%9F%AC%EB%A1%9C-%ED%95%A0-%EC%88%98-%EC%9E%88%EB%8A%94-%EC%9D%BC-%EC%97%86%EB%8A%94-%EC%9D%BC
http://whoisit.tistory.com/378
http://nuts84.tistory.com/
https://www.casaba.com/products/x5s/