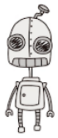
2021 Incognito CTF Write-up

해킹 멈춰!

step 1. 봇 탐지 우회



검사중입니다. 잠시만 기다려 주십시오..

처음 마주하게 되는 화면

귀여운 최첨단 로봇이 나를 반기며 검사중이라고 합니다.

잠시 기다리면 `Detected Result` 가 뜨며 null이라고 뜨게됩니다.

잠시 후 아래와 같은 화면이 출력됩니다.

봇이 탐지되었습니다

User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64; ; NCLIENT50_AAP5051A320C23) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36
Bot Name : None

봇 데이터베이스 검색 결과 봇으로 탐지되었습니다.
해당 서비스를 이용하실 수 없습니다.

봇으로 탐지되어 넘어가게 되는 화면

분명히 `Detected Result` 는 `null` 이라고 떴었습니다.

그런데 문제에서 의뢰인이 말했던 것처럼 접속을 하면 봇으로 탐지되게 됩니다.

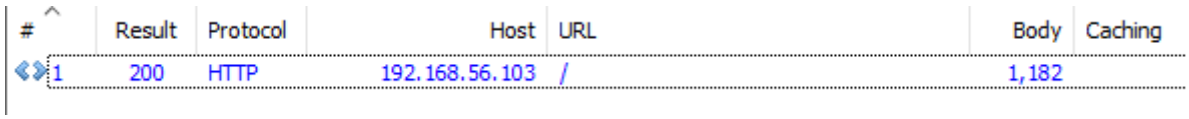
여기서 출력되는 문구를 잘 보면 현재 접속한 `User-Agent`를 보여줌을 알 수 있고, `Bot Name`을 `None`으로 출력하는 것으로 보아 탐지된 봇 이름을 보여줌을 알 수 있습니다. 또한, 그 아래 `데이터베이스 검색`과 같은 문장으로 다음과 같은 결과를 도출해낼 수 있습니다.

> 봇은 `User-Agent` 헤더를 이용해 데이터베이스 질의를 하고, 그 결과인 `Bot Name`에 따라 차단을 판단한다!

또한, 탐지 결과가 None으로 설정되는 것으로 보아 내 User-Agent가 데이터베이스에는 없는 건 맞는것 같고, 근데 봇으로 탐지한다? 문제에서는 해커가 소스코드를 바꾼 것 같다고 했으니, 이에 대한 원인은 다음과 같이 생각해볼 수 있습니다.

- 아예 반대로 DB에 데이터가 없을 때 봇으로 탐지하도록 장난침.
- 그냥 싹다 봇으로 탐지되게 장난침.

해커도 접속을 해야하지 않을까? 라고 생각했다면 첫 번째로 좁혀질 수 있을 것 같습니다.

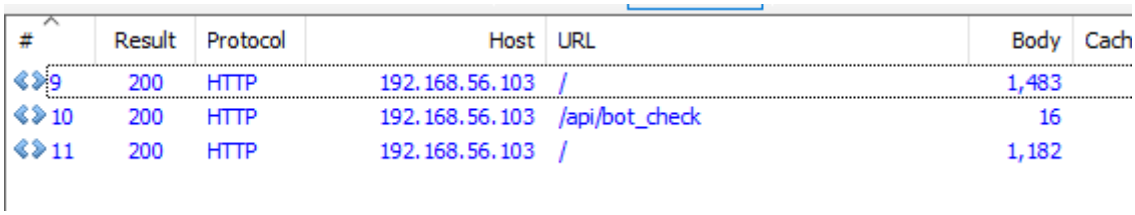


#	Result	Protocol	Host	URL	Body	Caching
1	200	HTTP	192.168.56.103	/	1,182	

탐지된 이후 패킷을 확인하기 위해 재시도한 결과.

우선 어떤 패킷이 발생하는지 알아보고자 다시 실행해보면 뭔가 달라진 점이 있을겁니다. 처음 접속할 때는 대기 시간이 존재했고, 초록색 팝업도 뜨면서 친절하게 탐지 결과도 알려준 후에 최종 `차단 페이지`로 넘어갔습니다.

그런데 지금은 그런 과정이 하나도 없고 바로 차단 페이지로 넘어갔습니다. 동일한 세션으로 한번 탐지되었기 때문인데요, 세션을 지우면 다시 처음과 같은 순서대로 동작합니다. 다시 패킷을 확인해보겠습니다.



#	Result	Protocol	Host	URL	Body	Cach
9	200	HTTP	192.168.56.103	/	1,483	
10	200	HTTP	192.168.56.103	/api/bot_check	16	
11	200	HTTP	192.168.56.103	/	1,182	

세션을 비운뒤 요청했을 때의 결과

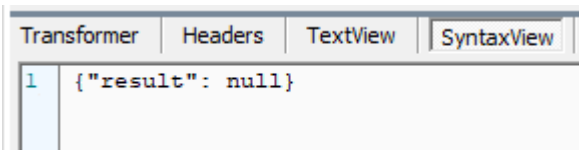
총 3번의 패킷이 발생합니다. 처음 메인페이지에 접근했을 때, 그리고 딱봐도 봇을 체크할 것 같은 url, 마지막으로 차단페이지.

▼ 몰라도 되지만 조금만 더 TMI,,!

`처음 접근한 페이지`와 최종적으로 출력되는 `차단페이지`는 동일한 URL에 접근한 결과입니다. 동일한 URL에 동일한 패킷을 보냈는데, 다른 결과를 내었다는 것은 그 사이에 어떠한 일련의 과정을 통해 서버단에 저장된 데이터가 있다는 뜻이겠죠.

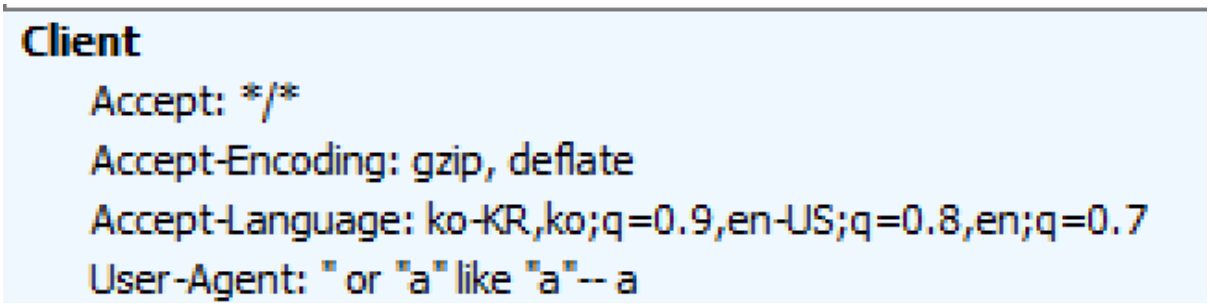
예를들면 지금 이 문제에서는 그 사이에 `bot_check` 라는 과정이 존재하기 때문에 봇을 체크하면서 그 결과를 어딘가에 저장해두었다고 추측해볼 수 있습니다. 그래서 앞서 다시 접속하면 바로 차단 페이지가 떴던 이유도 이와 같지요! 그렇기 때문에 마치 처음 접속하는 것처럼 세션을 비우고 접속을하면 다시 원래대로 동작하는 것입니다.

이 부분은 다양한 원인이 있을 수 있어서 꼭 이렇다! 는 아니고 지금 문제가 그렇다~ 정도로만 참고하시면 될 것 같습니다. 몰라도 문제는 풀어요!



/api/bot_check 요청의 응답데이터

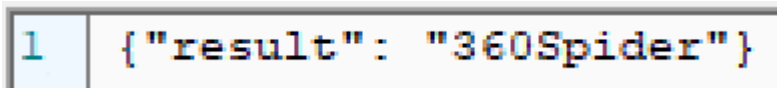
이와 같이 응답데이터를 확인해보면 화면에 출력되었던 질의 결과를 반환하는 것을 알 수 있습니다. 실질적으로 봇을 체크하는 로직으로 볼 수 있겠네요! 응답 결과도 바로바로 오니 맛있어 보입니다!



단간한 공격 페이로드가 들어간 패킷

더욱 간단한 " or 1=1— a로 하시거나 " or 1=1#로 하신다면 result가 null로 뜨게 됩니다. 네, 필터링입니다. '=', '#', '\\', 'tbl_name', 'union select' 가 필터링되어 있기 때문에 조금 피곤한 문제라고 생각하실 수도 있겠지만 큰 막힘 없이 푸실 수 있으실 겁니다.

우선 데이터베이스를 사용하며, `User-Agent` 헤더로 비교를 한다고 하니 여기에 SQL Injection을 시도해볼 수 있겠네요! 결과는 아래와 같습니다.

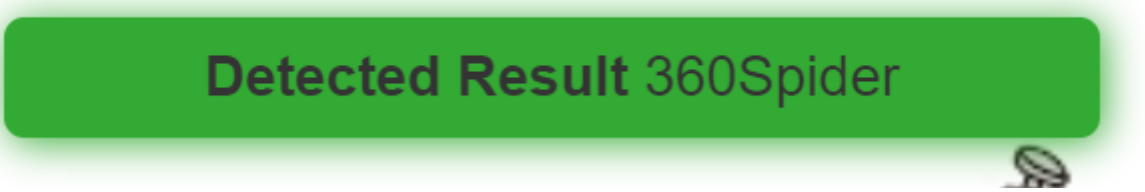


공격 결과

짜잔!

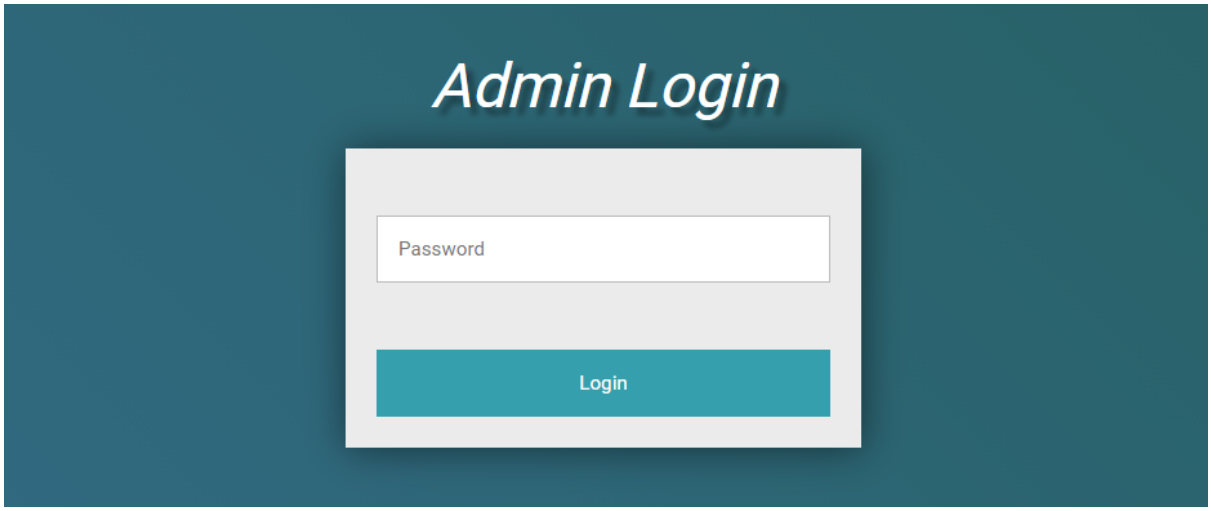
여기서 저는 서버가 `User-Agent` 를 input으로 사용하기 때문에 input으로 사용하는 모든 데이터가 공격 벡터가 될 수 있다! 라는 점을 문제를 통해서 전달하고 싶었습니다.

위와 같이 데이터베이스에 있는 봇을 알아내 보았습니다. 그렇다면 이렇게 봇이 실제 있는 경우는 서버가 어떻게 반응할까요?



프록시로 잡아서 조작된 User-Agent로 요청한 결과

성공적으로 Detect 되는 것을 확인할 수 있습니다. 그렇다면 과연 이 경우에는 어떻게 될까요?

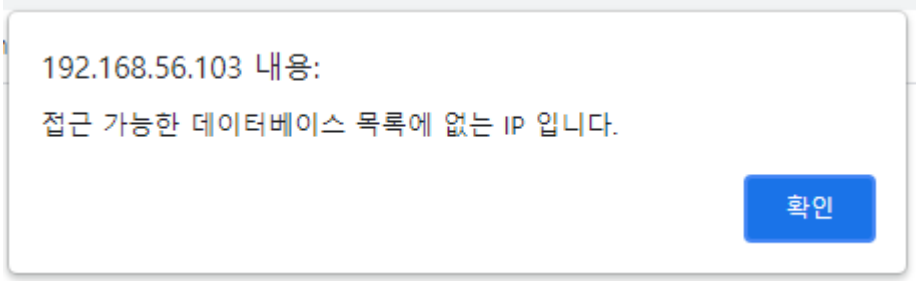


등장해버린 관리자 로그인 페이지

드디어 봇 관문을 통과했습니다!

step 2. IP 탐지 우회

그러나 로그인이 남아있죠. 걱정할 필요가 없는게, 문제에서 비밀번호는 다 제공되었습니다. `djenadls~?` 이 비밀번호로 로그인 해보겠습니다.



로그인 결과

로그인을 시도했지만 받게되는 메세지는 차단 메세지입니다. 여기서 알 수 있는 점은 접근 가능한 IP를 설정하는 기능이 있고, 그 IP는 데이터베이스를 통해 관리한다는 점입니다.

그렇다면 이 IP는 무엇으로 설정되어있을까요?

- 관리자가 주로 접속할 때 사용하는 IP
- 해커가 추후 접속을 위해 설정해놓은 해커의 IP

이런 식으로 생각해볼 수 있는데, 중요한건 접속할 수 있는 IP를 알아내야 한다는 점입니다. 데이터베이스에 있는 IP를 어떻게 알아낼 수 있을까요?

사실 저희는 이미 데이터베이스에 한번 공격을 했던 이력이 있습니다. DB에 공격이 가능한 것을 알고 있으니 이를 통해 DB를 탈탈 털어보는건 어떨까요?

#4. 본격 DB털이

DB를 털기위한 공격은 `/api/bot_check` 페이지에 계속 다양한 쿼리를 날려보면서 진행할 수 있습니다.

`/api/bot_check` 에는 필터링이 적용되어 있습니다. '=', '#', '\\', 'tbl_name', 'union select' 그리고, 20글자 이하의 user-agent가 필터링되어 있습니다.

```
tbl_name -> name

union select -> union/**/selec
```

그 중 키워드의 필터링은 위와 같이 간단하게 우회하여 줄 수 있습니다.

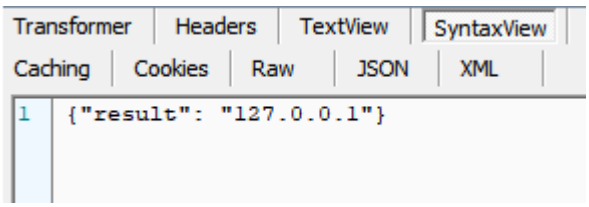
```
hi" union select version() -- 오류발생

hi" union select @@version -- 오류발생

hi" union select sqlite_version() -- 정상적으로 출력됨
```

위 쿼리문을 필터링에 걸리지 않게 후회하여 공격한 결과를 통해 sqlite DB를 사용함을 알 수 있고, sqlite 문법에 맞게 페이로드를 작성해나가면 됩니다.

```
hi" union/**/select name from sqlite_master --

hi" union/**/select name from sqlite_master limit 3,1 --

hi" union/**/select sql from sqlite_master where name like "white_list" --

hi" union/**/select ip from white_list --
```

주요한 쿼리들은 위와 같습니다. 존재하는 table을 찾다보면 `white_list` 라는 테이블이 있는 것과, `ip` 컬럼이 존재함을 알 수 있습니다. 최종적으로 화이트리스트 테이블을 출력해보

면 아래와 같이 결과가 잘 나옵니다.



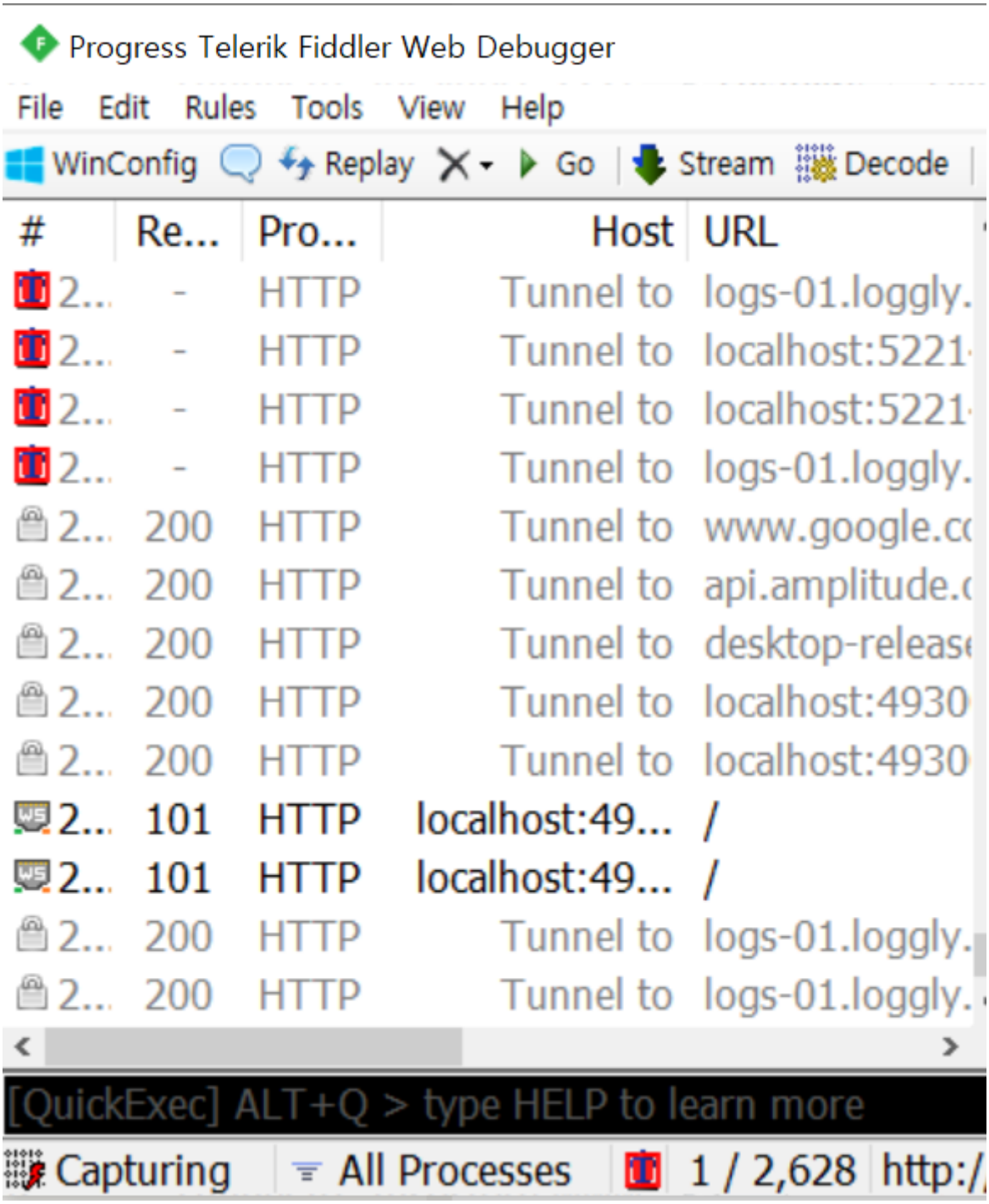
ip 조회 결과

#5. 마무리

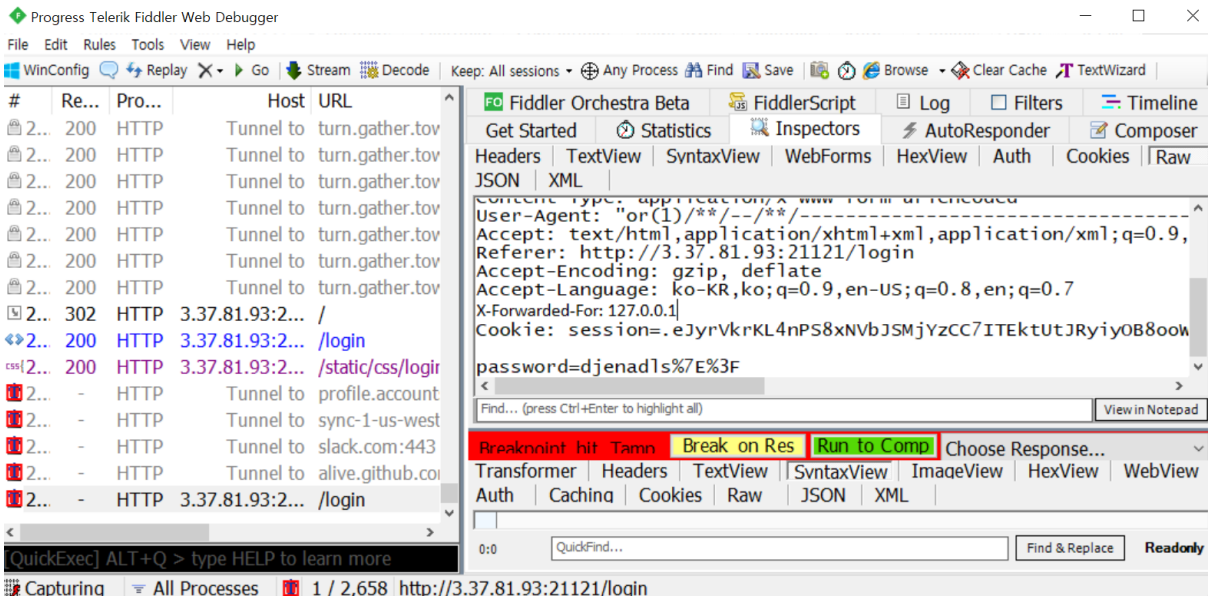
white_list에 존재하는 IP를 얻었으니, 로그인 요청 시 `X-Forwarded-For` 헤더를 통해 클라이언트 IP를 위장하면 정상적으로 로그인이 됩니다

방법이 여러가지 있는데 Fiddler를 기준으로 설명드리겠습니다.

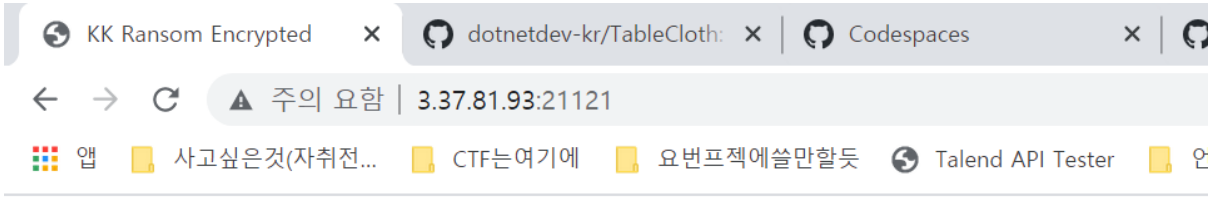
우선 패킷캡처를 킵니다.



로그인 페이지에서 비밀번호를 치고 엔터를 누른 후 캡쳐된 요청 패킷에
X-Forwarded-For 헤더를 사용해줍니다.



Run to Competition을 누르면 !



If You earn decrypt key? I'll decrypt it !

Key :

Decrypt

Do you wanna decrypt? send me a money :)

아 Step이 하나 더있네요,,,

step 3. 랜섬웨어 해결

막막할땐 일단 아무값이나 입력해봅니다.

If You earn decrypt key? I'll decrypt it !

Key :

Decrypt

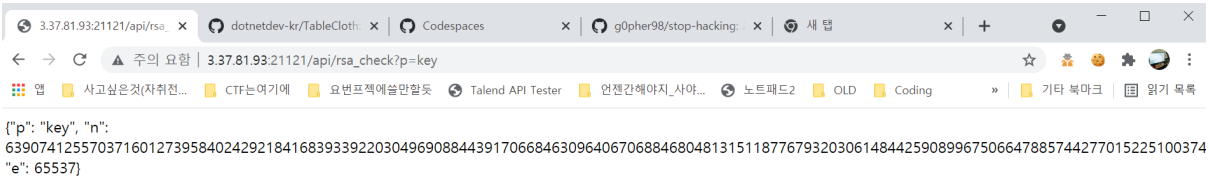
Do Not BruteForce~~ Just send me a money :)

Bruteforce 공격은 하지 말라는 군요

혹시나 하고 소스코드를 확인해보면 해커의 인간미를 볼 수 있습니다.

```
</head>
<body>
    <!-- TODO: 테스트 용도이므로 RSA Api 배포할 때 꼭 지울 것 !!! -->
    <!-- RSA Key exchange check : /api/rsa_check?p=key -->
<!-- RSA Encrypted key : https://gist.githubusercontent.com/l0vey0u/07fbf731bdfb4daf73fd2207b044eb8d/raw/cac971cbf0efc9eb5d6db71fbe78528c10f10977/encrypted_key -->
    <h1>If You earn decrypt key? I'll decrypt it !</h1>
```

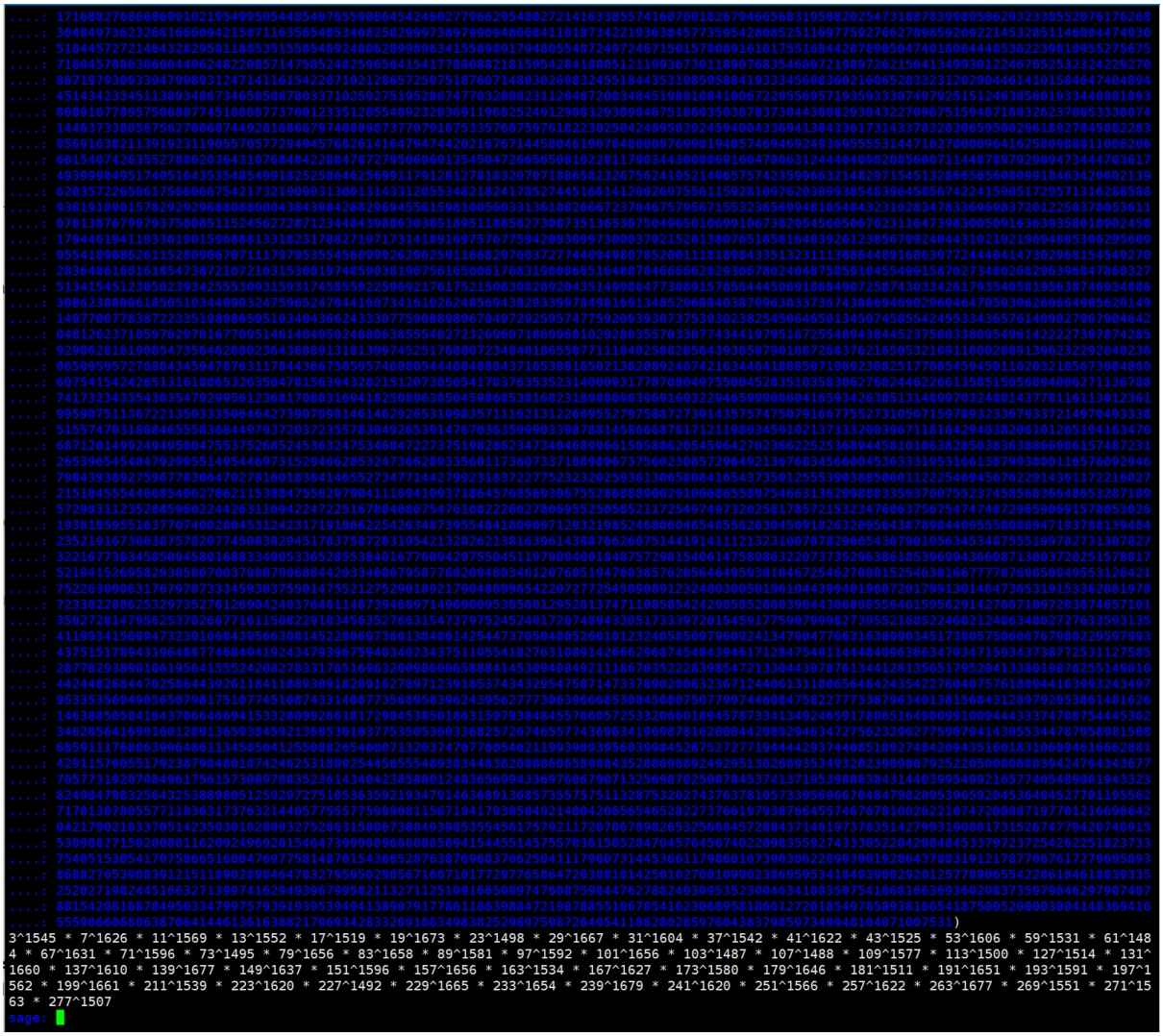
RSA로 암호화된 키와 테스트 용으로 사용한 api 주소 위치까지 알려주네요



친절하게 n, e 정보를 알 수 있습니다.

c도 저 주소로 들어가면 알 수 있습니다.

n이 인수분해가 되는지 sage 를 활용하여 factor(n) 을 돌려보면



인수분해가 된다는 기쁨도 잠시 뭔가 많이 이상합니다.

보통 RSA의 형태인 $N = pq$ 가 아닌 $N = pqr \ldots$ 꼴로 되어있음을 확인할 수 있습니다.

multiprime rsa는 여러개의 소수의 곱을 활용하여 계산됩니다.

d를 구하는 방법도 동일합니다. ed mod n = 1 인 d 를 찾으면 되죠

다만 그 d가 무지하게 크기 때문에 $p = c^d \mod n$ 계산이 문제가 됩니다.

여기서 사용되는 것이 바로 중국인의 나머지 정리 (Chinese Remainder Theorem)

전체적인 계산 방식은 다음과 같습니다

$$M \equiv M_{p_1} \pmod{p_1}$$
$$M \equiv M_{p_2} \pmod{p_2}$$
$$\cdots$$
$$M \equiv M_{p_n} \pmod{p_n}$$

$p_1 ... p_n$은 모두 서로소일 때 위 식들은 CRT를 이용하여 하나의 식으로 정리할 수 있고

$$M = \sum_{i=0}^{n} M_{p_i} \cdot N/p_i \cdot ((N/p_i)^{-}1 \pmod{p_i}) \pmod{N}$$

$$M_p \equiv M \pmod{p}$$
$$M_p \equiv (C^d \pmod{N}) \pmod{p}$$
$$M_p \equiv C^d \pmod{p} \because p|N$$
$$M_p \equiv C^{k\phi(p)+d \bmod \phi(p)} \pmod{p} \equiv C^{d \bmod \phi(p)} \pmod{p}$$
$$\because C^{\phi(p)} \bmod p = 1$$

맨 마지막 줄이 이해가 안 간다면 오일러의 정리와 페르마의 소정리를 공부하자
C, p가 서로소가 아닌 경우 오일러의 정리로는 불가능하다.

그럴 땐 피함수를 전개해보면
$$\phi(p_i) = p_i - 1$$
$$\phi(p_i) = p_i^k \cdot (p_i - 1)$$

두 경우 모드 페르마의 소정리를 통해 정리가 가능합니다.

최종적으로
$$M \equiv \left(\sum_{i=0}^{n} C^{d \bmod \phi(p_i)} \bmod p_i \cdot \ N/p_i \cdot ((N/p_i)^{-}1 \pmod{p_i}) \right) \pmod{N}$$

아래 코드는 sage를 활용하여 구현한 예시입니다.

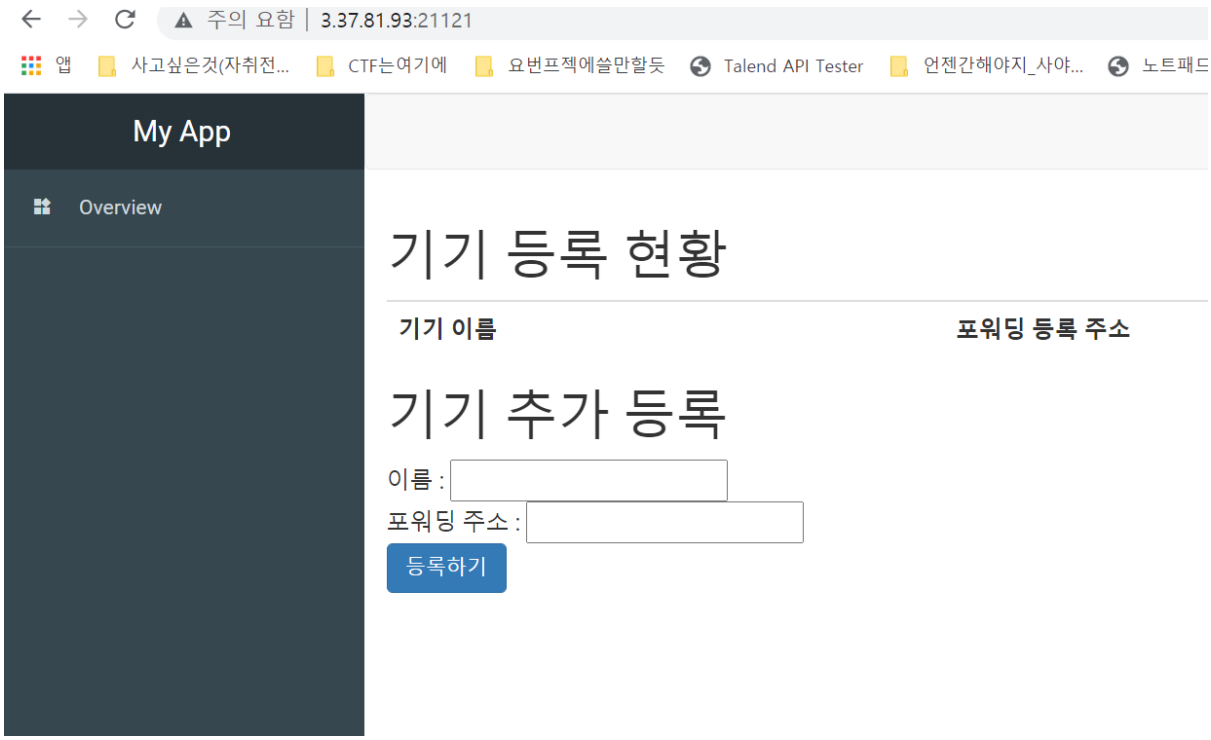
```
#!/usr/bin/env sage
# -*- coding: utf-8 -*-
# Refer : FireShell CTF 2019 Biggars

from operator import mul
from Crypto.Util.number import long_to_bytes, bytes_to_long

# make prob.txt
N =
e = 65537
```

```
C = int(open('encrypted_key', 'r').read())
# Factor N using sage factor , ex ) 45 = 3^2 * 5 = [(3,2),(5,1)]
UPF = list(factor(N))
# Calc Phi
phi_fs = [pow(p, k-1) * (p-1) for p, k in UPF]
# calc raw factor
factors = [pow(p, k) for p, k in UPF]
# d = 1/e mod phi(N)
d = inverse_mod(e, reduce(mul, phi_fs, 1))      # Private exponent
# Use CRT to compute d rather than the fast-power algorithm
#
M = CRT_list(
    [int(pow(C, d % p_i, f_i)) for p_i, f_i in zip(phi_fs, factors)],
    [N / f_i for f_i in factors]
)

# Flag
print(long_to_bytes(M))
```

해당 코드를 돌리면 키를 얻을 수 있습니다.

```
root@c8f93675629e:/var/prob# sage decrypt.sage
Setting permissions of DOT_SAGE directory so only you can read and write it.

b'Hacker_d0!_Maliya_saram_e_yah_uh~'
root@c8f93675629e:/var/prob#
```

Hacker_d0!_Maliya_saram_e_yah_uh~ 를 키로 입력하면 대시보드가 반깁니다.



Flag는 어디있는가

바로 title에 있습니다.