Mendeteksi Serangan *Buffer Overflow* pada *Source Code* Bahasa C menggunakan Metode *Taint Analysis*

Deby Nofalia Fardiansari[1], Vera Suryani[2], Aulia Arif Wardana[3]

[1,2,3]Fakultas Informatika, Universitas Telkom, Bandung
[1]debynofalia@students.telkomuniversity.ac.id, [2] verasuryani@telkomuniversity.ac.id,
[3] auliawardan@telkomuniversity.ac.id

Abstrak

Program merupakan bagian yang yang sangat penting dari kemajuan teknologi pada zaman seperti sekarang. Program banyak di ciptakan oleh *programmer- programmer* handal yang tentunya harus memenuhi kebutuhan dari kemajuan teknologi itu sendiri, dalam membuat suatu program pastinya ada beberapa faktor penting yang harus di penuhi salah satunya adalah faktor keamanan dari program itu sendiri. Biasanya peretas lebih banyak menyerang aplikasi melalui program, salah satu program yang rentan terhadap serangan yaitu program dengan bahasa C. Bahasa pemograman jenis ini di kenal rentan terhadap serangan *cyber* khususnya serangan *buffer overflow*. Karena bahasa C tidak memiliki *automatic bounds checking* untuk mengecek batasan suatu *buffer.* Serangan ini dapat dipicu oleh suatu masukan yang ditambahkan secara berlebihan melebihi jumlah atau ukuran data yang akan memenuhi *memory* pada program yang di jalankan dan serangan ini akan memberikan *access control* pada program tersebut. Terdapat beberapa metode yang bisa di lakukan untuk mendeteksi serangan *buffer overflow* , salah satu metode yang dapat di gunakan adalah dengan metode *taint analysis* . Metode ini befungsi untuk mengecek fungi – fungsi yang berpotensi sebagai ancaman *buffer overflow* dengan mengedepankan presentase akurasi sebesar 90% dan waktu eksekusi 0.026 detik pada hasil uji cobanya.

Kata Kunci: Program Bahasa C, Serangan *Buffer Overflow, Static taint analysis*

Abstract

Programs are a very important part of technological advances in this era. Many programs are created by reliable programmers who of course have to meet the needs of technological advancement itself. In making a program, of course there are several important factors that must be fulfilled, one of which is the security factor of the program itself. Usually, hackers attack more applications through programs, one of the programs that is vulnerable to attack is a program in C language. This type of programming language is known to be vulnerable to cyber attacks, especially buffer overflow attacks. Because the C language doesn't have automatic bounds checking to check the boundaries of a buffer. This attack can be triggered by an input that is added in excess of the amount or size of data that will fill the memory of the program being run and this attack will provide access control to the program. There are several methods that can be done to detect buffer overflow attacks, one method that can be used is the taint analysis method. This method is used to check for potential functions as a buffer overflow threat by prioritizing an accuracy percentage of 90% and an execution time of 0.026 seconds on the testresults.

Keywords: ***C Language Program, Buffer Overflow Attacks, Taint analysis***

1. Pendahuluan

1.1 Latar belakang

Bahasa pemrograman C merupakan salah satu bahasa pemrograman komputer yang dikembangkan pertama kali oleh Dennis Ritchie pada tahun 1972 [1]. Didesain untuk diimplementasikan di sistem perangkat lunak, namun juga dapat diimplementasikan diperangkat kecil, contohnya HP dan mikrokontroler [2]. Bahasa C sering kali dipakai untuk membuat file-file pustaka yang menyimpan fungsi-fungsi tertentu, karena dapat dikompile menjadi bahasa mesin yang sangat cepat dan kecil ukurannya [3]. Bahasa C ini lebih mengedepankan performansi daripada keamanan yang membuat library standar C sangat rawan jika tidak digunakan dengan berhati-hati. Kekurangan pada bahasa C tidak menyediakan *built-in* perlindungan terhadap cara mengakses atau menimpa data dalam setiap bagian data dari memori, dan tidak secara otomatis memeriksa bahwa data tersebut ditulis ke *array* (*built-in tipe buffer*) yang termasuk dalam batas-batas *array* [4] sehingga rentan terhadap serangan. Salah satu serangan yang terjadi yaitu *buffer overflow*.

Pada serangan *buffer overflow* dapat dipicu oleh suatu masukan yang ditambahkan secara berlebihan melebihi jumlah atau ukuran data yang diterima program. Masukan ini berupa kode yang disisipkan oleh penyerang pada slot alamat memori dari program, kode yang disisipkan seolah-olah merupakan kode yang sah dari input program [5]. Namun sebenarnya kode tersebut dapat mengakibatkan kejadian yang tidak diinginkan seperti merusak data, bisa merubah path dan perintah eksekusi [6]. Tujuan dari serangan ini yaitu melemahkan fungsi dari suatu program sehingga penyusup dapat mengambil alih kendali pada program tersebut .

Berdasarkan permasalahan yang ada di atas, pada tugas akhir ini penulis mengembangkan penelitian-penelitian sebelumnya dengan melakukan pengecekan pada program yang terindikasi serangan *buffer overflow*. Salah satu cara untuk mengatasi serangan tersebut, diperlukannya pendeteksian terhadap program pada bahasa C yaitu menggunakan metode *taint analysis. Taint analysis* merupakan tahapan analisis program berdasarkan fungsi-fungsi yang dimiliki oleh progam tersebut [7]. Terdapat dua macam cara kerja dalam metode *taint analysis* yaitu *static taint analysis* dan *dynamic static taint analysis.* Pada penelitian ini menggunakan konsep *static taint analysis* yang mendeteksi serangan tanpa harus menjalankan program terlebih dahulu. Program yang dicurigai terindikasi serangan *buffer overflow* dicek menggunakan sistem yang akan dibangun. Jika terdapat fungsi yang berpotensi sebagai serangan *buffer overflow* maka file tersebut akan ditandai sebagai file *tainted*. Sistem pendeteksi serangan *buffer overflow* ini sudah pernah dilakukan, oleh karena itu pada tugas akhir ini akan dilakukan penelitian menganalisa serangan *buffer overflow* pada program bahasa C dan hal yang membedakan dari penelitian sebelumnya yaitu hasil akurasi dan waktu eksekusi pada saat mendeteksi.

1.2 Topik dan batasan

Rumusan masalah yang akan diselesaikan pada tugas akhir ini adalah sebagai berikut :

1. Bagaimana cara mendeteksi serangan *buffer overflow* pada bahasa pemrograman C dengan menggunakan metode *taint analysis.*
2. Bagaimana performansi sistem yang dibangun dengan metode *static taint analysis* untuk mendeteksi serangan *buffer overflow.*

Batasan masalah yang akan digunakan pada tugas akhir ini adalah sebagai berikut :

1. Analisis deteksi serangan *buffer overflow* pada pemrograman bahasa C menggunakan konsep *static taint analysis.*
2. Penelitian dilakukan dengan mendeteksi fungsi yang berpotensi mengalami serangan *buffer overflow*.
3. Penelitian hanya berfokus pada file yang terindikasi serangan *buffer overflow.*

1.3 Tujuan

Pada penelitian ini dilakukan dengan tujuan mengimplementasikan metode *static taint analysis* untuk mendeteksi serangan *buffer overflow* pada pemrograman bahasa C. Untuk mengetahui waktu yang dibutuhkan pada saat mendeteksi serangan *buffer overflow* dan akurasi yang didapatkan dari sistem tersebut.

1.4 Organisasi Tulisan

Pada penulisan ini untuk bab 2 akan dijelaskan tentang Studi Terkait, bab 3 dijelaskan tentang sistem yang dibangun diantaranya adalah pembuatan program dengan metrode *static taint analysis*. Pada bab 4 akan dijelaskan tentang evaluasi, dan bab 5 dijelaskan kesimpulan dan saran. Pada sub-bagian ini dituliskan bagian-bagian selanjutnya (setelah Pendahuluan) pada jurnal TA ini, disertai penjelasan sangat singkat.

2. Studi Terkait

2.1 Peneliti Terkait

Table 1 Penelitian Terkait

No	Judul	Penulis	Tahun terbit	Hasil
1	SecureC: Control-flow Protection Against General Buffer Overflow Attack [8]	Hiroyasu Nishiyama	2005	Hasil dalam penelitian ini secara keseluruhan adalah menggambarkan bagaimana *buffer flow* menyerang suatu program dengan mengunakan sistem yang disebut SecureC yang menggunakan metode terjemahan yang disebut shadow stack. Hubungannya dengan tugas akhir ini adalah sama-sama mengkaji cara kerja Buffer Overflow menyerang suatu program.
2	Buffer Overflow, Ancaman Keamanan Perangkat Lunak dan Solusinya [6]	Damar Widjaja	2008	Hasil dalam penelitian ini secara keseluruhan adalah memahami proses serangan buffer overflow. Penelitian ini menjelaskan proses terjadinya buffer overflow secara kompleks. Karena sifat serangan yang beragam, buffer overflow merupakan ancaman serius untuk kedepannya. Teknik yang digunakan pada penelitian ini yaitu peluncuran Security Patches. Hubungan dengan tugas akhir ini adalah sama-sama mengkaji metode apa saja yang digunakan untuk mendeteksi serangan Buffer Overflow pada suatu program.
3	Classification of Static Analysis-based Buffer Overflow Detectors [9]	Hossain Shahriar, Mohammad Zulkernine	2010	Hasil dalam penelitian ini secara keseluruhan adalah menyajikan klasifikasi terperinci berdasarkan survei ekstensif dengan metode static static taint analysis yang mendeteksi kerentanan BOF. Hubungan dengan tugas akhir ini adalah sama-sama menjelaskan metode static static taint analysis untuk mendeteksi serangan buffer over flow.
4	Basic Static Code Analysis untuk Mendeteksi Backdoor Shell pada Web Server [7]	Nelly Indriani Widiastuti, Muhammad Iqbal	2017	Hasil dari penelitian ini secara keseluruhan adalah menganalisa bagaimana tahapan metode static taint analysis. Hubungan dengan tugas akhir ini adalah sama-sama mengkaji metode static taint analysis yang merupakan tahapan analisis source code berdasarkan fungsi-fungsi yang dimiliki oleh source code tersebut.

5	Staticly Detect Stack Overflow Vulnerabilities with Taint Analysis [10]	Zhang XING, Zhang BIN, Feng CHAO, Zhang QUAN	2016	Hasil dari penelitian ini secara keseluruhan adalah menyajikan proses taint analysis dalam mendeteksi serangan stack overflow. Hubungan dengan tugas akhir ini adalah sama-sama mengkaji metode taint analysis berdasarkan parameter fungsi yang ada pada source code tersebut.

2.2 Bahasa Pemrograman C

Program bahasa C ini sangat rentan terhadap serangan. Salah satu penyebabnya yaitu minimnya keamanan dan tidak mempunyai pengecekan otomatis (*automatic bounds checking*) untuk mengecek batasan suatu *buffer* [11]. Namun pada dasarnya bahasa pemrograman C untuk mikrokontroler sama dengan bahasa pemrograman C untuk dekstop, akan tetapi ada sedikit perbedaan pada proses pengaksesan register dan memori yang digunakan di dalam pemrograman mikrokontroler [12].

2.3 Serangan *Buffer Overflow*

Serangan *buffer overflow* merupakan serangan yang dilakukan dengan memanfaatkan kelemahan pada program yang dibuat agar memiliki celah untuk dimodifikasi. Serangan *buffer overflow* sendiri terjadi karena penyerang berusaha untuk memberikan data melebihi alokasi memory untuk buffer [6] sehingga program mengalami kelebihan muatan dan memory tidak dapat mengalokasikannya. Terkadang pembuat program juga hanya menulis baris kode tanpa mengecek ukuran dari *buffer* yang dituju untuk mengetahui apakah cukup untuk menerima data baru [6]. Ketika memory tidak mampu menampung input yang berlebihan maka akan memberikan kesempatan kepada penyerang untuk menindih data pada program dan mengambil alih kontrol program yang berhasil diserang [13].

Serangan terhadap sistem keamanan menggunakan teknik *buffer overflow* bertujuan untuk mendapatkan hak akses ke sistem dengan cara merubah aliran kontrol dari sebuah program sehingga program akan mengeksekusi kode yang telah dibuat dengan sangat hati-hati oleh penyerang [5]. Karena kerentanan *buffer overflow* adalah salah satu target serangan yang paling sering, pertahanannya penting untuk meningkatkan keamanan sistem [8].

Penyebab terjadinya serangan *buffer overflow* karena adanya celah kelemahan pada pemrograman terutama pada bahasa pemrograman C dan C++ yang tidak mendukung pengecekan terperinci kode yang tidak sah seperti inputan di luar batas. Dengan demikian, program yang ditulis dalam bahasa C cenderung rentan. Proses ntuk mencegah terjadinya serangan *buffer overflow* yaitu dengan menghindari beberapa fungsi standar C, karena mereka tidak melakukan pengecekan terhadap panjang string yang dimasukkan sehingga fungsi yang tidak aman harus diganti dengan fungsi yang aman [14]. Berikut ini adalah beberapa buah fungsi standar C umum yang tidak aman beserta alternatif solusinya :

Table 2 Fungsi Standar C

Fungsi Yang Vulnerability	Fungsi Yang Unvunerability	Kegunaan
strcpy()	strlcpy()	Menyalin string sumber ke sebuah buffer
gets()	fgets()	Membaca sebuah baris dari stdin ke buffer
strcat()	strncat()	Menambah sebuah string ke string lain
scanf()	-	Membaca STDIN
memcpy()	-	Menyalin dari area memori ke tujuan memori

sprint()	snprintf()	Mencetak output berdasarkan suatu format string
strlen()	len()	Menghitung panjang karakter pada suatu string
strncpy()	strcpy_s()	Menggabungkan satu string dengan bagian lain

Fungsi-fungsi yang tidak memiliki tingkat keamanan yang aman makan harus ditulis ulang dengan pengecekan implementasi. Seperti fungsi strcpy () untuk menyalin string sumber ke buffer. Karena tidak ada cara untuk membatasi jumlah data yang dibaca oleh fungsi ini [16]. Fungsi-fungsi ini dikategorikan rentan terhadap serangan *buffer overflow* dikarenakan dalam fungsi tersebut memiliki hubungan yaitu area memori yang digunakan untuk penyimpanan sementara ketika data disalin kedalam buffer yang ukurannya lebih kecil dari string sumber maka buffer kelebihanmuatan.

2.4 Taint analysis

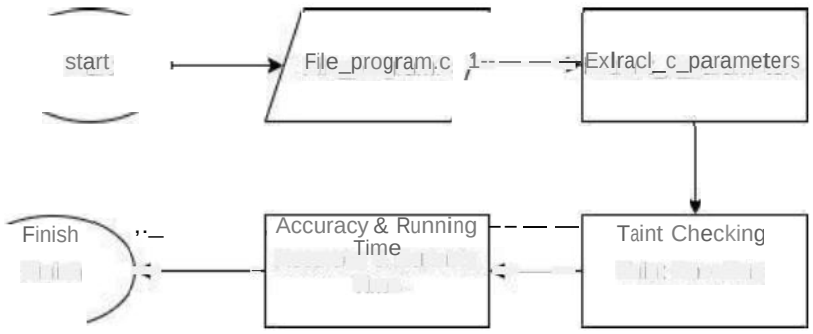
Berdasarkan penelitian *static taint analysis* digunakan untuk menganalisis kode sumber yang mengandung fungsi-fungsi yang diidentifikasi sebagai serangan *buffer overflow*. Perlu dicatat bahwa dalam sistem ini kode sumber akan dicari ciri-ciri yang menunjukkan *buffer overflow*. Metode ini dapat membantu mengidentifikasi dan menemukan bagian yang rentan dari kode sumber. *Static taint analysis* mengidentifikasi input berbahaya yang berasal dari sumber yang tidak tepercaya dan melacak semua data variabel yang terpengaruh oleh sumber input. Metode ini telah banyak digunakan di seluruh bidang keamanan *Cyber*.

Taint analysis dibagi menjadi 2 konsep yaitu *dynamic static taint analysis* dan *static static taint analysis*. Perbedaan dari keduanya yaitu, jika pada *dynamic taint analaysis* mendeteksi serangan pada saat program dijalankan. Jika *static taint analysis*, mendeteksi serangan pada saat program sebelum dijalankan. Pada penelitian kali ini, konsep yang digunakan untuk mendeteksi serangan *buffer overflow* adalah konsep *static taint analysis*. Karena hanya akan mengecek pada saat *coding* tidak pada saat program berjalan. Detail prosedur pengujian disediakan termasuk pengaturan baris perintah untuk alat dan skrip. Tidak ada notasi yang ditambahkan ke kode sumber untuk salah satu alat. Satu-satunya modifikasi yang dibuat adalah untuk *PolySpace* karena *buffer overflows* terdeteksi dalam rutinitas perpustakaan seperti *strcpy* dan tidak dipetakan ke dalam program utama ke titik di mana rutin perpustakaan dipanggil [15].

3. Perancangan Sistem

3.1 Gambaran Umum Sistem

Pada penelitian ini sistem yang akan dibuat dalam tugas akhir adalah dengan membaca program yang mengalami kondisi serangan *buffer overflow* dengan menggunakan metode *static taint analysis.* Untuk menghitung keberhasilan sistem yang dibangun berasal dari seberapa akuratnya sistem mendeteksi serangan serta waktu yang dihasilkan dalam mendeteksi serangan tersebut. Berikut adalah penjelasan sistem yang dibangun pada penelitian ini:



Gambar 1 Flowchart Deteksi Program

Pada Gambar 1 menampilan alur dari sistem yang dibangun yaitu dengan dilakukannya pendeteksian terhadap ±10 program yang berpotensi serangan *buffer overflow*. Proses awal, mengambil program berbahasa C yang berpotensi serangan *buffer overflow* menggunakan 10 program. Proses selanjutnya extract_c_parameters, yaitu proses pemisahan parameter terhadap fungsi yang digunakan pada program. Untuk menyimpan parameter yang telah dipisah disimpan kedalam *array*. Kemudian proses *taint checking* yaitu pencocokan parameter dan fungsi yang disimpan di *array* dengan fungsi yang didefinisikan di ruleset. Kemudian jika terdapat parameter dan fungsi yang berpotensi *buffer overflow* maka akan menampilkan *warning*. Proses terakhir adalah menghitung akurasi dari sistem dan juga *execution time* pada saat mendeteksi *sourcecode*.

3.2 Penerapan Static taint analysis

Langkah pertama penerapan metode *static taint analysis* ini yaitu memisahkan parameter pada setiap fungsi yang ada di program. Berikut merupakan kategori yang bukan parameter sebagai berikut :

Table 3 Kategori String dan Komentar

Nama variable	Karakter
Instring	Tanda petik ganda (“ ”) , tanda petik satu (‘)
Incomment	"/*" dan "*/", “//”

Jika pada text ditemukan tanda “(” dan “)” bukan tanda yang telah dikategorikan diatas maka parenlevel ditambahkan dan *array* parameter di append. Kemudian dilakukan proses pencocokan fungsi yang diikuti dengan parameter dilihat dan dipindai berdasarkan [7]. Fungsi-fungsi pada program dicocokan dengan fungsi yang telah didefiniskan pada ruleset. Peneliti dapat melihat suatu program terkena serangan *buffer* overflow dari fungsi-fungsi string C yang rentan.

3.3 Performansi Sistem

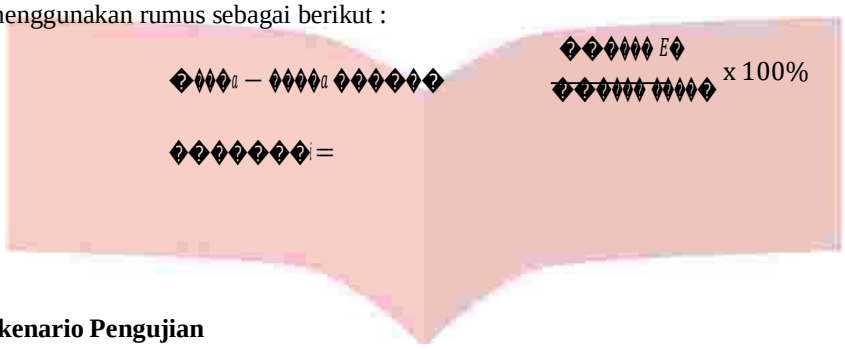
Pengukuran terhadap kinerja suatu sistem klasifikasi merupakan hal yang penting. Kinerja sistem klasifikasi menggambarkan seberapa baik sistem dalam mengklasifikasikan data. Penelitian ini menerapkan metode *taint analysis* dan menggunakan confusion matrix untuk menghitung akurasinya. Untuk melakukan perhitungan akurasi yaitu dengan melakukan proses pengujian program yang terkena serangan *buffer overflow* menggunakan data test aplikasi bahasa C diantaranya *VisualCodeGrepper* [18]. Membandingkan total jumlah serangan yang terdeteksi pada penelitian dengan total jumlah serangan yang diuji cobakan pada data test. Selanjutnya perhitungan menggunakan rumus yang telah dibuat sebelumnya untuk mendapatkan akurasi yang tinggi. Akurasi pada penelitian memiliki rumus sebagai berikut:

$$\text{Akurasi} = \frac{\text{Jumlah Serangan Terdeteksi}}{\text{Jumlah Serangan Diuji}} \times 100\%$$

Dan untuk menghitung rata-rata akurasi memiliki rumus sebagai berikut:

$$\text{Rata-rata Akurasi} = \frac{\text{Jumlah Akurasi}}{\text{Jumlah Data Uji}} \times 100\%$$

Selain perhitungan akurasi, penelitian ini menghitung *execution time* dari proses berjalannya sistem menggunakan metode *static taint analysis.* Perhitungan rata-rata waktu eksekusi menggunakan rumus sebagai berikut :



4. Evaluasi

4.1 Skenario Pengujian

Pada pengujian sistem yang ingin diperoleh adalah seberapa akurat penggunaan metode *static taint analysis* serta *execution time* dalam mendeteksi serangan *buffer overflow*. Tahap pengujian dilakukan menggunakan ±10 data uji bahasa C dengan kondisi *buffer overflow.* Berikut merupakan 10 *source code* yang akan diuji :

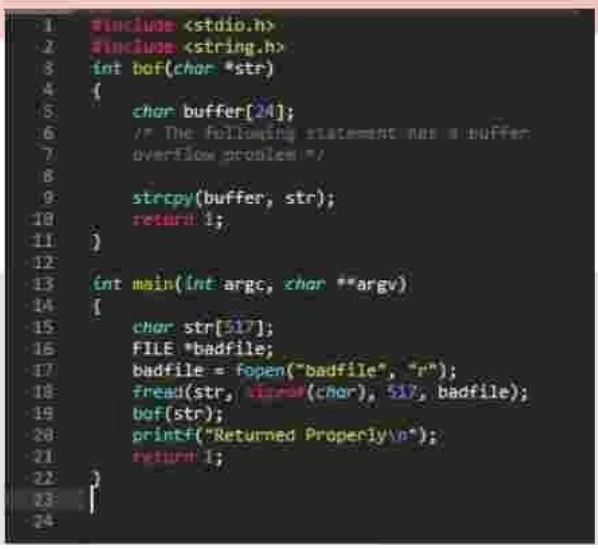
Table 4 Sumber Program Penelitian

Filename	Line	Sumber
Stack1.c	22 line	https://github.com/npapernot/buffer-overflow-attack
Stack2.c	29 line	https://www.thegeekstuff.com/2013/06/buffer-overflow/
Stack3.c	14 line	https://www.programiz.com/c-programming/library-function/string.h/strcat
Stack4.c	31 line	https://github.com/DhavalKapil?tab=overview&from=2020-04-01&to=2020-04-30
Stack5.c	14 line	https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.3.0/com.ibm.zos.v2r3.bpxbd00/memcpy.htm
Stack6.c	9 line	https://ccode123.blogspot.com/2015/11/strlen.html
Stack7.c	16 line	https://c-for-dummies.com/blog/?p=3921
Stack8.c	10 line	https://www.codesdope.com/c-string/
Stack9.c	19 line	https://owasp.org/www.community/attacks/Buffer_overflow_attack
Stack10.c	15 line	https://github.com/npapernot/buffer-overflow-attack

Sistem akan mendeteksi seluruh program pada tabel 4 yang berpotensi serangan *buffer overflow* dengan melihat fungsi-fungsi yang rentan. File program diproses satu persatu baris untuk dilakukan pemindaian. Tahap awal yang dilakukan yaitu *extract c parameter* , memisahkan antara parameter dan fungsi pada *source code* bahasa C. Proses ini memindai text setiap baris lalu dilakukan pencocokan setiap text, jika text pada baris ditemukan tanda "(" dan ")" tanpa diikuti dengan tanda-tanda yang dikategorikan pada Table 3 , maka telah ditemukan parameter (parenlevel) akan ditambahkan dan array parameter akan diappend sehingga ia akan menyimpan bagian kode yang tidak memenuhi semua kondisi di atas sebagai parameter. Hasil dari *extract c parameter* berupa list parameter dari *source code* kemudian akan disimpan ke dalam hitlist. Setelah itu dilakukannya penelitian [7] proses *taint checking*, proses pencocokan fungsi otomatis diikuti dengan pencocokan parameter. Dipindai dan dicocokan berdasarkan fungsi yang rentan terhadap serangan *buffer overflow*. Fungsi-fungsi yang rentan terhadap serangan *buffer overflow* ini

didefinisikan pada ruleset. Jika pada *source code* terdapat fungsi yang terindikasi *buffer overflow* maka sistem akan mengeluarkan output berupa nama fungsi, letak baris, panjang string, warning, dan suggestion bahwa program tersebut yang mengalami serangan *buffer overflow*. Terakhir dilakukan proses perhitungan *execution time* dengan menghitung waktu ketika *proses static taint analysis* dikerjakan.

Program bahasa C/C++ pada Tabel 4 ini diambil secara acak. Dalam Gambar 2 diberikan contoh *source code* stack1.c yang akan diuji dengan sistem yang dibangun.



```c
#include <stdio.h>
#include <string.h>
int bof(char *str)
{
    char buffer[24];
    /* The following statement has a buffer
    overflow problem */

    strcpy(buffer, str);
    return 1;
}

int main(int argc, char **argv)
{
    char str[517];
    FILE *badfile;
    badfile = fopen("badfile", "r");
    fread(str, sizeof(char), 517, badfile);
    bof(str);
    printf("Returned Properly\n");
    return 1;
}

```

Gambar 2 Contoh Source Code Stack1.c

Proses *extract_c_parameters*

Tahap awal proses ini, *source code* dipindai kemudian dipecah perbaris untuk memisahkan antara parameter dan fungsi. Jika ditemukan tanda "(" dan ")" tanpa diikuti dengan tanda-tanda lainnya maka parenlevel akan ditambahkan dan array parameter akan diappend.

```
Extract C Parameters starts here
FUNCTION extract_c_parameters(text, pos-0):
    whILe i < len(text):
        IF text(i] = '(':
            break
        ELSEIF text[i]
            i += 1
        else:
            RETURN ( ]
        ENDIF
    else:
        RETURN [ ]
    ENDWHILE
```

```
    while i < len(text):
        c <- text[i)
        IF instring:
            IF c • ....
                instring <- 0
            ELSEIF c • "'"
                instring <• 0
            ELSEIF c • '\\':
                i +• 1
            ENDIF
        ELSEIF incomment:
            IF c • '*' AND '*I':
                incomment <- 0
                i +• 1
            ENDIF
        else:
            IF c •
                instring c•
            ELSEIF c • """: 1
                instring c-
            ELSEIF c • '/' AND 't•':
                incomment <- 1
                i += 1
            ELSEIF c = '/' AND 'II':
                while i < len(text) AND text(i) != "\n":
                    i += 1
                ENOWHILE
            ELSEIF c   '\\' AND '\\'":
                i += 1
            ELSEIF c = '(':
                parenlevel +=
            ELSEIF c   ',' AND1parenlevel   1:
                parameters.append(
                currentstart c- i + 1
        ELSEIF c • ')':
            parenlevel -= 1
            IF parenlevel <= 0:
                parameters.append
    ENDIF
        ENDIF
            ENDIF
ENllhttIL
```

Gambar 3 Pseudocode Extract C Parameters

```
C:\Windows\system32\cmd.exe

C:\Users\LENOVO\Downloads\scratch\scratch>call python ui.py

C:\Users\LENOVO\Downloads\scratch\scratch>call python out.py -D src
Examining src\stack1.c
['', 'buffer', 'str']
['', '"Returned Properly\\n"']
```

Gambar 4 Proses Exctract C Parameters

Proses Pengecekan Fungsi *Buffer Overflow*

Pada proses ini, pengujian dilakukan pada program bahasa C, sistem membaca nama file dari data, setelah itu membaca setiap hasil dari pencocokan pada *ruleset*, jika pada proses *taint checking* terdapat fungsi dan parameter yang memiliki potensi untuk terjadi serangan *buffer overflow* maka sistem akan mendata fungsi dan akan dikeluarkan sebagai output bahwa fungsi tersebut terindikasi serangan *buffer overflow*.

Gambar 5 Hasil Pengecekan Fungsi Buffer Overflow

4.2 Hasil Pengujian

Pengujian ini dilakukan perhitungan *execution time* (waktu eksekusi) yang digunakan untuk melihat seberapa waktu yang dibutuhkan pada saat mendeteksi serangan *buffer overflow*. Perhitungan *execution time* dimulai dari fungsi pada saat pemanggilan file c, membaca program, kemudian proses *taint analysis* dijalankan yaitu fungsi *extract c parameters* sampai dengan hasil pencocokan fungsi yang rentan terhadap serangan *buffer overflow*. Dengan melihat total fungsi pada masing-masing file uji dan total fungsi yang mengalami kondisi *buffer overflow* terdapat pada file program merupakan hasil pengujian 10 program yang telah dipindai dapat dilihat pada Tabel 5.

Table 5 Hasil Pengujian

File Name	*Total Function In Program*	*Total Function Buffer Overflow detected*	*Execution Time*
Stack1.c	7	1 (strcpy)	0.03 detik
Stack2.c	7	1 (gets)	0.02 detik
Stack3.c	4	1 (strcat)	0.02 detik
Stack4.c	11	2 (1 strcpy, 1 scanf)	0.04 detik
Stack5.c	4	1 (memcpy)	0.03 detik
Stack6.c	3	1 strlen	0.02 detik
Stack7.c	4	1 (sprintf)	0.02 detik
Stack8.c	4	2 (1 strncpy, 1 strlen)	0.03 detik
Stack9.c	7	1 (gets)	0.02 detik
Stack10.c	5	1 (strcpy)	0.03 detik
Rata-rata waktu eksekusi			**0.026 detik**

Pada Tabel 5 dapat dilihat total *function buffer* pada setiap file yang di uji. Total *function buffer* yang ditemukan pada masing-masing file memiliki 1 atau lebih fungsi yang rentan terhadap *buffer overflow*. Pada penelitian ini, waktu yang dibutuhkan untuk melakukan pendeteksian *buffer overflow* pada 10 file program tersebut yaitu 0.026 detik.

4.3 Analisis Hasil Pengujian

Pada penelitian ini akan dilakukan perbandingan dengan 2 skenario. Untuk mendapatkan nilai akurasi dari penelitian ini dilakukannya pengujian pada sistem yang telah dibangun menggunakan metode *static taint analysis* dan dengan data test *VisualCodeGrepper* [18]. Data test tersebut sama-sama menggunakan *static analysis* sebagai metode yang digunakan untuk mendeteksi serangan [17]. Kemudian membandingkan hasil pengujian tersebut. Skenario perbandingan dilakukan dengan melihat *total function buffer overflow* yang terdeteksi menggunakan masing- masing skenario. Hasil perbandingan dari kedua skenario dapat dilihat pada Tabel 6.

Table 6 Hasil Akurasi Pengujian

File Name	*Total Function Buffer Overflow with Static Static taint analysis (STA)*	*Total Function Buffer Overflow in VisualCodeGrepper (VCG)*	*Accuration*
Stack1.c	1 (strcpy)	2 (1 strcpy, 1 fopen)	50%
Stack2.c	1 (gets)	1 (gets)	100%
Stack3.c	1 (strcat)	1 (strcat)	100%
Stack4.c	2 (1 strcpy, 1 scanf)	2 (1 strcpy, 1 scanf)	100%
Stack5.c	1 (memcpy)	1 (memcpy)	100%
Stack6.c	1 strlen	2 (strlen, scanf)	50%
Stack7.c	1 (sprintf)	1 (sprintf)	100%
Stack8.c	2 (1 strncpy, 1 strlen)	2 (1 strncpy, 1 strlen)	100%
Stack9.c	1 (gets)	1 (gets)	100%
Stack10.c	1 (strcpy)	1 (strcpy)	100%
Average Accuration			**90%**

Berdasarkan Tabel 6 dapat dilihat perbedaan *total function buffer overflow* yang terdeteksi pada kedua metode. Dari sepuluh data uji delapan diantaranya menghasilkan *True Positive* (TP), *False Positive* (FP) sebanyak 2. Terdapat dua data uji yang terdeteksi memiliki *total function buffer overflow* yang berbeda yaitu pada data uji nomor 1 dan 6. Pada data nomer satu memiliki dua *total function buffer overflow* yang terdeteksi. Kemudian data uji nomer 6 tidak ada fungsi *buffer overflow* yang terdeteksi menggunakan data

test. Berikut merupakan hasil rata-rata dari pengujian file c yang mengandung *buffer overflow*, akurasi yang didapatkan adalah 90%. Pengecekan ini dilakukan berulang-ulang pada *VisualCodeGrepper* [18].

5. Kesimpulan

Bedasarkan hasil pengujian dan analisis yang dilakukan pada percobaan ini terdapat kesimpulan yaitu :

1. Dalam penelitian yang menggunakan metode *static taint analysis* terdapat langkah-langkah yang mendukung perannya masing-masing dalam proses pendeteksian serangan *buffer overflow*. Terbukti dari hasil pengujian pada penelitian ini dapat mendeteksi *buffer overflow* dengan metode *static taint analysis* berdasarkan ±10 program yang telah diuji. Hasilnya mampu mendapatkan letak serangan *buffer overflow* yang berupa letak baris, warning, dan panjang string masing-masing file.
2. Metode *taint analysis* ini mampu menghasilkan akurasi 90% dari hasil rata-rata pengujian file c yang mengandung *buffer overflow* yaitu dengan membandingkan hasil pengujian pada data test *VisualCodeGrepper*.
3. Metode *static taint analysis* membutuhkan waktu untuk mendeteksi *buffer overflow* denganrata – rata sebesar 0.026 detik sesuai dengan jumlah baris dan proses pendeteksian pada program.

Adapun saran untuk pengembangan sistem yaitu pendeteksian dapat dilakukan dengan menggunakan binary analysis dan percobaan selanjutnya adalah metode *static taint analysis* tidak hanya digunakan dalam mendeteksi satu serangan, melainkan dapat mendeteksi berbagai kerentanan lainnya. Serta dapat dilakukan pada pemrograman berbasis tinggi seperti java maupun pemrograman lainnya.

Daftar Pustaka

[1] richy rotuahta Saragih, "Pemrograman dan bahasa Pemrograman," *STMIK-STIE Mikroskil*, no. December, pp. 1–91, 2016.

[2] P. P. J. Kalatiku Yuri Yudhaswana, "Pemrograman Motor Stepper Dengan Menggunakan Bahasa Pemrograman C," *Mektek*, no. Vol 13, No 1 (2011), 2011.

[3] J. Sihombing, "BAB I Sekilas Tentang Bahasa C," pp. 1–2, 2012.

[4] J. C. Foster, V. Osipov, N. Bhalla, N. Heinen, and D. Aitel, *Buffer Overflow Attacks*. 2005.

[5] B. Overflow, "1. Buffer Overflow," pp. 1–7, 1998.

[6] Damar Widjaja, "Buffer Overflow, Ancaman Keamanan Perangkat Lunak dan Solusinya," *Jur. Tek. Elektro, Univ. Sanata Dharma Yogyakarta Kampus III, Paingan, Maguwoharjo, Sleman damar@staff.usd.ac.id*, pp. 1–7, 2008.

[7] N. I. Widiastuti and M. Iqbal, "Basic Static Code Analysis Untuk Mendeteksi Backdoor Shell Pada Web Server," *J. Infotel*, vol. 9, no. 2, p. 177, 2017, doi: 10.20895/infotel.v9i2.209.

[8] H. Nishiyama, "SecureC: Control-flow protection against general buffer overflow attack," *Proc. - Int. Comput. Softw. Appl. Conf.*, vol. 1, pp. 149–155, 2005, doi: 10.1109/COMPSAC.2005.136.

[9] H. Shahriar and M. Zulkernine, "Classification of Static Analysis-Based Buffer Overflow Detectors Classification of Static Analysis-based Buffer Overflow Detectors," no. June 2010, 2015, doi: 10.1109/SSIRI-C.2010.28.

[10] Z. Xing, Z. Bin, F. Chao, andZ. Quan, "Staticly Detect Stack Overflow Vulnerabilities with Taint Analysis," *ITM Web Conf.*, vol. 7, p. 03003, 2016, doi: 10.1051/itmconf/20160703003.

[11] D. Untuk, M. Salah, S. Syarat, M. Gelar, S. Komputer, and J. Teknik, "Secure Programming Untuk Mencegah Buffer Overflow Fakultas Sains Dan Teknologi Universitas Islam Negeri (Uin) Alauddin Makassar," 2011.

[12] P. Fungsi, "Dasar Pemrograman Mikrokontroler dengan Bahasa C."

[13] T. Menghindari and B. Overflow, "Serangan Buffer Overflow," pp. 1–3.

[14] D. Riana, "SOFWARE SECURITY ANALYSIS Evaluasi Flawfinder dan ITS4 pada Source Code C / C ++," 2019.

[15] M. Zitser, D. E. S. Group, and T. Leek, "Testing Static Analysis Tools using Exploitable Buffer Overflows from Open Source Code," pp. 97–106, 1996.

[16] The Open Web Application Security Project "Buffer Overflow Attack", 2013. [Online]. Available: https://owasp.org/www-community/attacks/Buffer_overflow_attack. [Accessed: 05-Juli-2020]

[17] The Open Web Application Security Project "Source Code Analysis Tools", 2012. [Online]. Available: https://owasp.org/www-community/Source_Code_Analysis_Tools. [Accessed : 19-Agustus-2020]

[18] NCCGroup "Visual Code Grepper", 2019. [Online]. Available: https://github.com/nccgroup/VCG. [Accessed : 20 November 2020]