Penerapan Kombinatorial dan Analisis Kompleksitas dalam Pengembangan Algoritma untuk Menembus Proteksi Stack Canary pada Eksploitasi Biner

Panji Sri Kuncara Wisma - 13522028[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522028@std.stei.itb.ac.id

***Abstract*—Keamanan dari perangkat lunak adalah hal yang penting untuk diperhatikan. Ada banyak jenis proteksi yang digunakan untuk meningkatkan keamanan perangkat lunak. Salah satunya adalah proteksi stack canary untuk mencegah serangan *buffer overflow*. Akan tetapi tidak ada program yang sepenuhnya aman terhadap, pada kasus-kasus tertentu algoritma untuk menembus proteksi stack canary pada perangkat lunak adalah hal yang mungkin untuk dibuat. Algoritma tersebut dirancang dengan memanfaatkan prinsip-prinsip kombinatorial dan analisis kompleksitas. Pendekatan ini dilakukan agar algoritma tidak hanya berjalan dengan benar tapi juga memiliki kompleksitas waktu dan ruang yang baik.**

***Keywords*—Stack Canary, Byte, Bit, Kombinatorial, Kompleksitas, Buffer Overflow.**

I. PENDAHULUAN

Perkembangan teknologi informasi membawa dampak yang signifikan pada peradaban umat manusia, namun bersamaan dengan itu, tantangan keamanan juga menjadi semakin kompleks, khususnya dalam konteks perangkat lunak. Salah satu aspek krusial dalam keamanan perangkat lunak adalah perlindungan terhadap serangan eksploitasi biner.

Eksploitasi biner adalah kegiatan memanfaatkan kelemahan atau bug dari suatu program pada tingkatan biner untuk mendapatkan akses tidak sah, mengambil alih kendali, atau bahkan menyusupkan kode program untuk tujuan tertentu. Ada banyak teknik yang bisa digunakan untuk mengeksploitasi suatu perangkat lunak. Akan tetapi, di antara teknik-teknik tersebut, serangan buffer overflow adalah metode yang paling umum digunakan, khsusunya untuk code program yang menggunakan bahasa C atau C++.

Serangan buffer overflow biasanya digunakan ketika ada fungsi-fungsi rentan seperti gets(), strcpy(), dan fungsi buatan sendiri yang tidak memvalidasi jumlah data masukan dengan hati-hati. Buffer overflow bekerja dengan cara menyusupkan data yang lebih besar dari yang dapat ditampung oleh buffer (tempat penyimpanan sementara dalam program) dan menulis ulang isi dari suatu memori sesuai dengan keinginan penyerang. Hal itu sangat berbahaya karena penyerang bisa melakukan banyak hal dari menulis ulang isi memori seperti mengendalikan alur program dan mendapatkan informasi penting. Oleh karena itu, proteksi Stack Canary muncul sebagai salah satu metode pertahanan untuk mencegah terjadinya serangan buffer overflow.

Meskipun proteksi Stack Canary telah membawa kemajuan dalam meningkatkan keamanan perangkat lunak, masih ada kemungkinan suatu teknik dapat menembus proteksi ini.. Ada dua metode yang mungkin bisa digunakan untuk menembus proteksi Stack Canary, yang pertama adalah dengan melakukan *leak* memori dan yang kedua adalah dengan metode *brute force.*

Makalah ini akan berfokus pada eksplorasi dan pengembangan algoritma *brute force* yang efektif dan efisien untuk menembus proteksi Stack Canary. Penerapan kombinatorial dan analisis kompleksitas akan menjadi unsur yang sangat penting untuk mengembangkan algoritma tersebut. Kombinatorial digunakan untuk mengeksplorasi berbagai kemungkinan kombinasi Stack Canary yang mungkin dan analisis kompleksitas digunakan untuk membuat algoritma yang sefektif dan sefisien mungkin.

Untuk keperluan uji coba, program yang akan dieksploitasi adalah program sederhana yang memiliki kerentanan terhadap buffer overflow dan memiliki sistem proteksi Stack Canary. Dengan merinci aspek-aspek di atas, makalah ini diharapkan dapat memberikan wawasan yang mendalam tentang peran kritis kombinatorial dan analisis kompleksitas dalam menghadapi tantangan keamanan pada eksploitasi biner. Melalui penelitian ini, diharapkan pula dapat ditemukan solusi-solusi inovatif yang dapat meningkatkan keamanan perangkat lunak dan melindungi data dari ancaman yang terus berkembang.

II. LANDASAN TEORI

A. Kombinatorial

Kombinatorial adalah cabang matematika untuk menghitung

jumlah penyusunan objek-objek tanpa harus mengenumerasi semua kemungkinan susunannya. Di dalam kombinatorial, penting untuk menghitung segala kemungkinan pengaturan objek. Dua kaidah dasar yang dapat diterapkan sebagai metode perhitungan dalam kombinatorial adalah kaidah perkalian dan kaidah penjumlahan.[3]

1. *Kaidah perkalian (rule of product)*
   Bila dilakukan dua buah percobaan dan hasilnya adalah percobaan 1 menghasilkan kemungkinan jawaban sejumlah p, sedangkan percobaan 2 menghasilkan kemungkinan jawaban sejumlah q, maka bila percobaan 1 **dan** percobaan 2 dilakukan , akan dihasilkan kemungkinan jawaban sejumlah (p × q).[3]

2. *Kaidah penjumlahan (rule of sum)*
   Bila dilakukan dua buah percobaan dan hasilnya adalah percobaan 1 menghasilkan kemungkinan jawaban sejumlah p, sedangkan percobaan 2 menghasilkan kemungkinan jawaban sejumlah q, maka bila percobaan 1 **atau** percobaan 2 dilakukan, akan dihasilkan kemungkinan jawaban sejumlah (p + q).[3]

Ada dua konsep penting dan fundamental di dalam kombinatorial, yaitu permutasi dan kombinasi. Permutasi digunakan ketika urutan atau posisi dari sekumpulan objek adalah hal yang penting. Sementara itu, kombinasi digunakan ketika urutan atau posisi dari sekumpulan objek tidaklah penting. [3]

1. *Permutasi*
   Permutasi r dari n elemen atau $P(n,r)$ adalah jumlah kemungkinan urutan r buah elemen yang dipilih dari n buah elemen, dengan r ≤ n, yang dalam hal ini, pada setiap kemungkinan urutan tidak ada elemen yang sama.[3]

$$P(n,r) = \frac{n!}{(n-r)!}$$

2. *Kombinasi*
   Kombinasi r dari n elemen atau $C(n,r)$ adalah jumlah pemilihan yang tidak terurut r elemen yang diambil dar n buah elemen.[3]

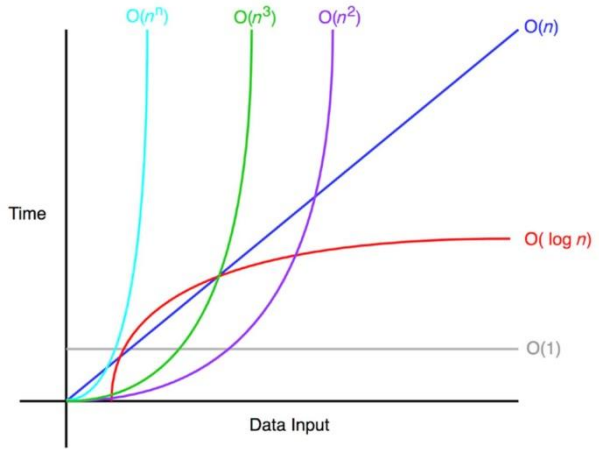
$$C(n,r) = \frac{n!}{r!\,(n-r)!}$$

B. Kompleksitas Algoritma

Dalam konteks pemrograman, penting untuk memiliki algoritma yang tidak hanya menemukan jawaban yang tepat tetapi juga melakukanya secara efisien. Efisiensi suatu algoritma dinilai dari waktu dan ruang yang dibutuhkan. Algoritma yang efisien adalah algoritma yang membutuhkan waktu dan ruang seminimum mungkin. Besaran yang dipakai untuk menjelaskan model pengukuran ruang/waktu disebut kompleksitas algoritma.[3]

Ada dua jenis kompleksitas algoritma, yaitu kompleksitas waktu dan kompleksitas ruang. Kompleksitas waktu lebih berfokus pada banyak tahapan komputasi yang dilakukan oleh algoritma dari ukuran masukan n. Kompleksitas ruang diukur dari seberapa banyak kebutuhan memori yang digunakan oleh suatu algoritma.[3]

Pada beberapa kasus, informasi mengenai kompleksitas waktu yang presisi tidak terlalu dibutuhkan. Terkadang lebih penting untuk mengetahui bagaimana kebutuhan waktu suatu algoritma tumbuh ketika ukuran masukannya meningkat. Salah satu notasi yang digunakan untuk merepresentasikan hal tersebut adalah notasi Big-O. Big-O melambangkan batas atas dari jumlah proses komputasi yang perlu dilakukan oleh suatu algoritma. Big-O akan memiliki makna yang berarti apabila nilainya sedekat mungkin atau tidak terlalu jauh dengan batas yang ditentukan. Algoritma dengan kompleksitas waktu O(f(n)) memiliki f(n) sebagai orde paling besar dengan n sebagai jumlah masukan.



Gambar 1 Grafik perbandingan efisiensi beberapa jenis Big-O

Sumber:https://droidtechknow.com/programming/algorithms/big-o-notation/

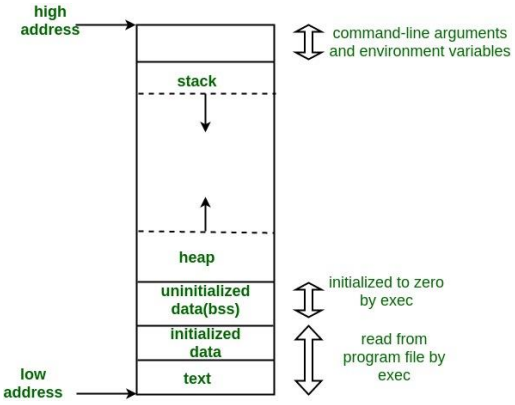
C. Arsitektur Komputer

Secara umum saat ini ada dua jenis arsitektur komputer yang sering digunakan, yaitu IA-32 dan x86-64. Pada komputer, *Central processing unit* (CPU) dan memori adalah bagian yang sangat krusial dalam konteks eksploitasi biner dan keamanan perangkat lunak. CPU bertugas untuk mengambil, menerjemahkan, dan menjalankan instruksi-instruksi yang bersumber dari perangkat lunak dan perangkat keras. Sementara itu, memori adalah media menyimpan data atau informasi pada komputer. Secara sederhana ada dua jenis memori yang penting di dalam komputer, yaitu *register* dan *main memory.*

1. *Register*
   Register terletak di dalam CPU. Memori ini digunakan agar komputer bisa mengakses informasi atau data tertentu dengan lebih cepat. Komputer dengan arsitektur IA-32 memiliki 8 jenis register, yaitu %eax, %ecx, %edx, %ebx, %esi, %edi, %esp, %ebp. Ukuran dari masing-masing *register* tersebut

adalah 32 bit.[2]

2. *Main memory*
   Main memory adalah lokasi penyimpanan utama di dalam komputer dan memiliki ukuran yang lebih besar dibandingkan *register*. Komputer akan menyiapkan tempat di *main memory* dengan ukuran tertentu ketika suatu program sedang berjalan atau dieksekusi. Secara intuitif, cara komputer mengelola memori mirip seperti pengelolaan tipe data stack.[2]



Gambar 2 Layout main memory

Sumber: https://www.geeksforgeeks.org/memory-layout-of-c-program/

Pada dasarnya komputer tidak bisa memahami kode program secara langsung. Hal itu karena komputer memyimpan dan mengelola informasi dalam bentuk bilangan biner 0 dan 1. Satuan data terkecil dalam penyimpanan komputer adalah bit. Setiap bit hanya bisa berisi nilai 0 atau 1. 8 bit bersama-sama akan membentuk 1 byte.[2]

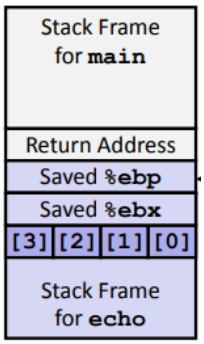
Biasanya memori di komputer memiliki alamat tertentu yang bisa digunakan sebagai referensi. Pada komputer dengan arsitektur IA-32 alamat memori memiliki panjang 32 bit atau 4 byte. Komputer memiliki dua cara untuk menyimpan informasi di dalam alamat tertenu, yaitu *little endian* dan *big endian*. Pada *little endian*, byte paling tidak signifikan dari suatu data di simpan di alamat memori yang paling rendah. Sedangkan Pada *big endian*, byte paling signifikanlah yang di simpan di dalam alamat memori paling rendah. Agar uji coba lebih sederhana, makalah ini akan lebih berfokus pada program-program yang diperuntukkan untuk arsitektur 32 bit dan dikelola secara *little endian*.[2]

D. Buffer Overflow

Bahasa C dan C++ adalah bahasa pemrograman yang memberi sedikit kebebasan kepada programmer untuk memanipulasi memori. Hal tersebut dapat dibuktikan dari adanya pointer. Untuk beberapa kasus, hal tersebut membuat program menjadi lebih efisien. Akan tetapi, untuk beberapa kasus lain, apabila programmer tidak berhati-hati dalam memanajemen memori, program bisa menjadi rentan terhadap serangan.

Buffer adalah tempat penyimpanan sementara. Ketika data yang dimasukan lebih besar dari ukuran yang sudah dialokasikan, ekstra data akan mengalami *overflow*. Hal tersebut bisa menimpa atau menulis ulang informasi lain yang ada di dalam memori. Normalnya, fungsi untuk membaca masukan harus melakukan penanganan untuk kasus tersebut. Akan tetapi, ada beberapa fungsi yang tidak melakukannya dan rentan terhadap serangan *buffer overflow*, contohnya adalah fungsi bawaan seperti gets() dan strcpy(). Selain dari fungsi bawaan tersebut, fungsi yang direalisasikan sendiri juga bisa memiliki kerentanan terhadap serangan *buffer overflow*.[1]

Seperti yang sudah diketahui, komputer akan menyiapkan tempat di memori ketika mengeksekusi suatu program. Tempat tersebut dinamakan *Stack Frame*.



Gambar 3 Stack frame untuk program yang sederhana

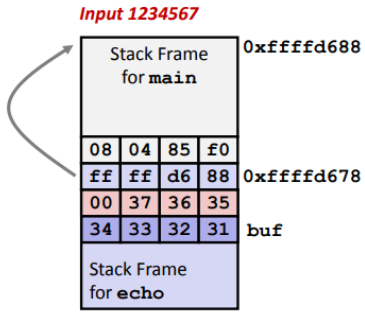
Sumber : https://www.cs.cmu.edu/afs/cs/academic/class/15213-m13/www/lectures/old/09-machine-advanced.pdf

```
/* Echo Line */
void echo()
{
    char buf[4];  /* Way too small! */
    gets(buf);
    puts(buf);
}
```

Gambar 4 Kode program dari stack frame gambar 3

Sumber: https://www.cs.cmu.edu/afs/cs/academic/class/15213-m13/www/lectures/old/09-machine-advanced.pdf

. Gambar 4 menunjukan potongan kode program sederhanad dalam bahasa C yang akan meminta masukan dari pengguna dan menampilkanya. Kode tersebut memiliki fungsi gets() yang rentan terhadap serangan *buffer overflow*. Apabila pengguna memasukan data lebih besar dari ukuran buffer yang dalam kasus ini adalah 4 byte, maka blok memori lain di dalam *stack frame*. Gambar 5 menampilkan kondisi *saved %ebx* yang tertulis ulang setelah pengguna memasukan data dengan ukuran 7 byte.[4]

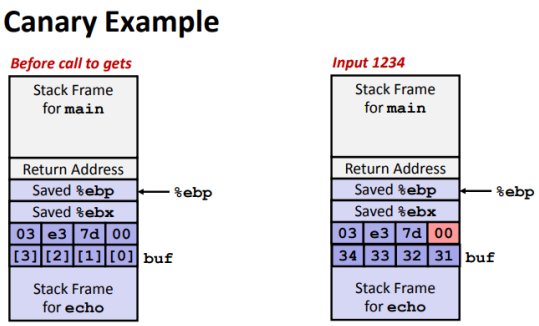


Gambar 5 Stack frame setelah mengalami buffer overflow

Sumber: https://www.cs.cmu.edu/afs/cs/academic/class/15213-m13/www/lectures/old/09-machine-advanced.pdf

E. Stack Canary

Stack canary adalah salah satu mekanisme keamanan yang digunakan untuk melindungi program dari serangan *buffer overflow*. Sebelum fungsi utama dieksekusi, suatu nilai acak berukuran 4 byte akan dihasilkan dan disimpan di lokasi tertentu di dalam stack. Nilai acak tersebut dinamakan nilai canary. Sebelum fungsi selesai dieksekusi, nilai canary yang ada di dalam stack akan di cek kembali. Jika nilai canary berubah, ini menandakan adanya upaya *buffer overflow*. Program akan langsung dihentikan dan akan muncul pesan *Stack Smashing Detected*. Gambar 6 menunjukan nilai canary yang ditempatkan dibawah blok *saved %ebx*.[4]



Gambar 6 Nilai canary pada stack frame

Sumber: https://www.cs.cmu.edu/afs/cs/academic/class/15213-m13/www/lectures/old/09-machine-advanced.pdf

III. Analisis Algoritma

A. Strategi penyerangan stack canary

Stack canary adalah nilai acak yang diletakan pada *Stack Frame* ketika program dieksekusi. Salah satu strategi yang bisa digunakan adalah menebak nilai canary dengan serangan *brute force*. Program akan mengalami *crash* atau otomatis dihentikan ketika nilai canary berubah. Oleh karena itu, nilai canary akan berhasil didapatkan ketika program tidak mengalami *crash* ketika dilakukan serangan *brute force.*

Untuk arsitektur IA-32 ukuran dari stack canary biasanya adalah 4 byte atau 32 bit. 1 bit direpresentasikan oleh bilangan 0 dan 1. Dari informasi tersebut, dapat diketahui bahwa nilai canary berada di antara bilangan 32 bit yang seluruhnya 0 sampai bilangan 32 bit seluruhnya 1.

$$\underbrace{000000\ldots0000000}_{32\text{ bit}} \quad - \quad \underbrace{1111111\ldots111111}_{32\text{ bit}}$$

Apabila direpresentasikan dalam bentuk desimal, angka yang coba ditebak berada pada rentang 0 – 4294967295. Dengan melakukan *brute force*, peluang mendapatkan nilai canary yang benar dari rentang tersebut adalah $\frac{1}{4924967296}$ atau 2.03047×10^{-10}. Jawaban yang benar untuk nilai canary akan didapatkan setelah melakukan percobaan paling buruk sebanyak 4294967296 kali. Ini adalah angka yang besar dan mungkin membutuhkan banyak waktu untuk melakukannya.

Pada dasarnya 4294967296 adalah jumlah percobaan maksimal yang bisa dilakukan untuk menembus *stack canary* pada mesin 32 bit. Akan tetapi hal tersebut memiliki syarat, yaitu mekanisme *stack canary* yang digunakan adalah *stack canary default* yang dihasilkan oleh kompiler. Hal itu karena nilai canary yang dihasilkan oleh compiler secara *default* memiliki ukuran 4 byte.

Masalahnya adalah jika programmer melakukan realisasi stack canary sendiri. Ukuran dari canary bisa dibuat sesuka hati programmer. Misalkan k adalah jumlah percobaan untuk menebak nilai dari n byte canary, maka $2^8 \times k$ adalah banyaknya percobaan yang dibutuhkan untuk menembus $(n + 1)$ byte canary. Dari sudut pandang lain, untuk n byte canary, paling buruk dibutuhkan 2^n percobaan hingga mendapatkan jawaban yang benar. Notasi O-besar dari algoritma ini adalah $O(2^n)$. Melihat informasi tersebut dapat diketahuii jika efisiensi dari algoritma yang menggunakan strategi ini tergolong buruk. Oleh karena itu, Strategi lain diperlukan untuk mendapatkan algoritma yang lebih efisien.

Untuk mendapatkan strategi penyerangan yang lebih baik, pendekatan lain perlu dilakukan. Oleh karena itu, daripada menebak nilai canary secara keseluruhan atau dalam satuan n byte, lebih baik memeriksa canary per byte secara terpisah. Poin pentingnya adalah tidak merubah nilai canary yang lain ketika menebak canary dari byte tertentu. Misalkan terdapat suatu canary yang berukuran 4 byte.

$$\underbrace{\_\ \_\ \_\ \_}_{4\text{ byte}}$$

Misalkan nilai canary yang ingin ditebak adalah 03e37d00. Oleh karena fokus bahasan dari makalah ini adalah pada mesin *little endian*, maka urutannya akan menjadi 007de303. Perlu diketahui, untuk setiap dua karakter pada nilai tersebut yakni 00, 7d, e3, dan 03 masing-masing berukuran 1 byte. 1 byte berukuran 8 bit. Oleh karena itu, paling banyak ada 2^8

kombinasi bit yang mungkin pada 1 byte yakni pada range 0 sampai 255. Misalkan x adalah 1 byte bagian dari nilai canary yang ingin ditebak, maka $0 \leq x \leq 255$ dan diperlukan paling buruk 256 percobaan untuk mendapatkan x yang benar.

$$\underbrace{256\ 256\ 256\ 256}_{\text{4 byte}}$$

Untuk menebak nilai canary berukuran 4 byte, byte pertama membutuhkan paling banyak 256 percobaan, byte kedua membutuhkan paling banyak 256 percobaan, begitu seterusnya hingga byte keempat. Mengubah 1 byte dari canary tidak akan menganggu nilai canary lainnya, jadi tidak masalah. Berdasarkan analisis tersebut, jumlah maksimal percobaan yang dibutuhkan untuk menebak 4 byte canary adalah

$$256+256+256+256 = 1024 \text{ percobaan.}$$

Angka yang jauh lebih kecil dibandingkan strategi menebak canary secara utuh. Strategi pertama membutuhkan maksimal 2^n percobaan untuk menembus n byte canary. Sementara itu, strategi menebak canary secara parsial atau per 1 byte membutuhkan $256 \times n$ percobaan untuk menembus n bytes canary. Notasi O-besar dari algoritma ini adalah O(n). Melihat informasi tersebut dapat diketahuii jika efisiensi dari algoritma yang kedua sudah cukup bagus dan lebih masuk akal untuk direalisasikan.

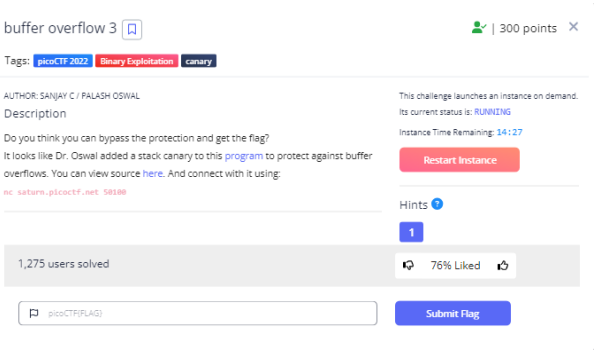
Merangkum analisis di atas, secara umum ada dua jenis algoritma yang dapat digunakan untuk menebak nilai canary dan menembusnya, yaitu

1. Algoritma menebak canary secara utuh
   Algoritma ini bekerja dengan cara menebak nilai canary secara keseluruhan. Untuk menebak *n* byte canary, dibutuhkan maksimal 2^n percobaan. Kompleksitas waktu dari algoritma ini adalah $O(2^n)$.

2. Algoritma menebak canary secara parsial
   Algoritma ini beroperasi dengan cara menebak nilai canary per satu byte. Untuk menebak *n* byte canary, dibutuhkan maksimal 256*n* percobaan. Kompleksitas waktu dari algoritma ini adalah O(*n*).

B. Realisasi penyerangan stack canary

Contoh soal yang akan digunakan untuk merealisasikan algoritma dari makalah ini berasal dari picoCTF.com, yaitu platform yang menyediakan tantangan keamanan siber. Oleh karena itu, file biner dan kode rentan yang digunakan juga berasal dari platform tersebut. Tujuan utama dari realisasi ini adalah menembus stack canary dari nilai yang ada dan melompat ke fungsi yang berisi jawaban.



Gambar 7 Soal terkait stack canary

Sumber: https://play.picoctf.org/practice/challenge/260

```
#define BUFSIZE 64
#define FLAGSIZE 64
#define CANARY_SIZE 4

void vuln(){
   char canary[CANARY_SIZE];
   char buf[BUFSIZE];
   char length[BUFSIZE];
   int count;
   int x = 0;
   memcpy(canary,global_canary,CANARY_SIZE);
   printf("How Many Bytes will You Write Into the Buffer?\n> ");
   while (x<BUFSIZE) {
      read(0,length+x,1);
      if (length[x]=='\n') break;
      x++;
   }
   sscanf(length,"%d",&count);

   printf("Input> ");
   read(0,buf,count);

   if (memcmp(canary,global_canary,CANARY_SIZE)) {
      printf("***** Stack Smashing Detected ***** : Canary Value Corrupt!\n"); // crash immediately
      fflush(stdout);
      exit(0);
   }
   printf("Ok... Now Where's the Flag?\n");
   fflush(stdout);
```

Gambar 8 Potongan kode program yang rentan

Sumber : dokumentasi penulis

Gamber 8 menunjukan potongan kode dari soal. Potongan tersebut menunjukan kode yang rentan terhadap serangan *buffer overflow*. Pengguna bisa mengatur masukan agar bisa lebih besar dari ukuran *buffer* dan memasukan data sejumlah yang diinginkan. Dapat dilihat pula bahwa kode ini tidak menggunakan proteksi canary dari compiler, melainkan buatan sendiri. Ukuran canary yang dibuat juga tidak terlalu besar yaitu sekitar 4 byte. Pada kasus ini sepertinya nilai canary tidak diacak setiap kali server dijalankan ulang. Oleh karena itu penyerangan dapat dilakukan dengan mengisi *buffer* dengan nilai random, kemudian menebak nilai canary, dan terakhir adalah melompat ke fungsi win() seperti yang ditunjukan oleh gambar 9. Dengan menggunakan *debugger* seperti GDB alamat dari fungsi win dapat diketahui yaitu berada pada 0x08049336.

```
void win() {
  char buf[FLAGSIZE];
  FILE *f = fopen("flag.txt","r");
  if (f == NULL) {
    printf("%s %s", "Please create 'flag.txt' in this directory with your",
                    "own debugging flag.\n");
    fflush(stdout);
    exit(0);
  }

  fgets(buf,FLAGSIZE,f); // size bound read
  puts(buf);
  fflush(stdout);
}
```

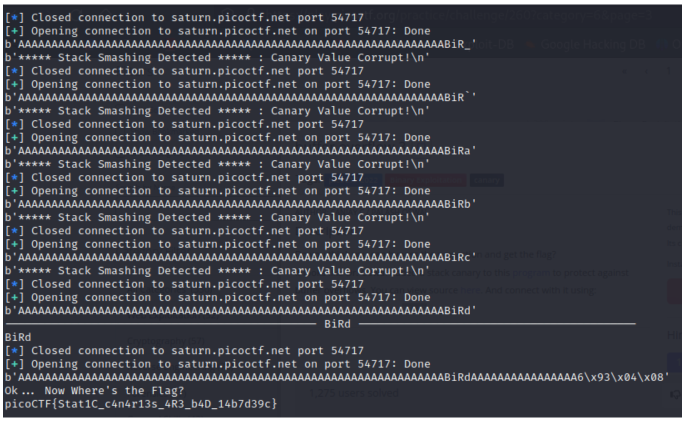
Gambar 9 Fungsi yang ingin dituju

Sumber: dokumentasi penulis

Gambar 10 Algoritma utama penembus stack canary

Sumber: dokumentasi penulis.

Gambar 10 menunjukan kode program utama untuk menembus stack canary. Program tersebut akan mencoba menghubungkan diri dengan server yang disediakan oleh soal. Algoritma yang digunakan adalah algoritma menebak nilai canary secara parsial. Program akan menulis ulang nilai canary yang ada pada server per byte. Jika salah maka program akan mengalami *crash*. Program akan terus menulis ulang canary hingga tidak terjadi *crash* sebagai penanda nilai canary sudah benar. Setelah berhasil menebak 1 byte, proses diulang untuk mendapatkan nilai canary secara keseluruhan. Tahap terakhir setelah menembus canary adalah melompat ke fungsi win() dan mendapatkan jawaban.



Gambar 11 Hasil penyerangan stack canary

Sumber: dokumentasi pribadi penulis

Gambar 11 adalah dokumentasi ketika program berjalan. Ternyata nilai canary yang didapat adalah 4 karakter yang apabila digabungkan menjadi *"BiRd"* . Sementara itu, jawaban dari soal yang dibahas dapat dilihat pada baris paling bawah.

.

IV. KESIMPULAN

Algoritma untuk menembus proteksi stack canary pada perangkat lunak adalah hal yang mungkin untuk dibuat. Algoritma tersebut dirancang dengan memanfaatkan prinsip-prinsip kombinatorial dan analisis kompleksitas. Pendekatan ini dilakukan agar algoritma tidak hanya berjalan tapi juga memiliki kompleksitas waktu dan ruang yang baik.

Meskipun proteksi stack canary telah memberikan lapisan keamanan tambahan, penelitian ini menunjukan bahwa tidak ada sistem yang sepenuhnya kebal terhadap serangan. Teknik-teknik seperti *brute force* tetap merupakan ancaman yang memerlukan perhatian serius dalam pengembangan perangkat lunak. Inovasi-inovasi untuk meningkatkan keamanan adalah hal yang penting, tapi kesadaran diri terhadap serangan yang munkin terjadi juga tidak kalah penting. Jadi, keamanan pada pengembangan perangkat lunak adalah hal yang juga penting untuk dipertimbangkan.

V. UCAPAN TERIMA KASIH

Terima kasih kepada Tuhan Yang Maha Esa karena berkat rahmat dan kasih karunia-Nya penulis dapat menyelesaikan makalah yang berjudul "Penerapan Kombinatorial dan Analisis Kompleksitas dalam Pengembangan Algoritma untuk Menembus Proteksi Stack Canary pada Eksploitasi Biner" dengan baik. Tak lupa juga penulis mengucapkan terima kasih kepada dosen-dosen pengampuh mata kuliah IF2120 terutama kepada Dr. Nur Ulfa Maulidevi, S.T., M. Sc. selaku dosen pengampuh mata kuliah IF2120 untuk kelas 01 karena telah memberikan pengetahuan yang dapat digunakan penulis untuk menulis makalah ini. Penulis berharap makalah ini juga bisa memberi manfaat baik bagi pelajar maupun masyarakat secara umum.

REFERENSI

[1] A. Sharan, "Buffer Overflow Attack with Example." Accessed: Dec. 11, 2023. [Online]. Available: https://www.geeksforgeeks.org/buffer-overflow-attack-with-example/

[2] R. Bryant and D. O'Hallaron, *Computer Systems A Programmer's Perspective*, vol. 2. Pearson, 2022.

[3] Munir, R. (2023). IF2120 Matematika Diskrit - Semester I Tahun 2023/2024. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/matdis23-24.htm

[4] Kesden, G. (2012, June 12). *Machine-Level Programming V: Advanced Topics*. https://www.cs.cmu.edu/afs/cs/academic/class/15213-m13/www/lectures/old/09-machine-advanced.pdf

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 11 Desember 2023

Panji Sri Kuncara Wisma
13522028