BAB I
PENDAHULUAN

1.1 Latar Belakang

Pada kehidupan sehari-hari dalam era modern hari ini tidak lepas dengan penggunaan perangkat lunak dalam berbagai aspek kehidupan, seperti untuk berkomunikasi, bermain *game* hingga aktivitas-aktivitas lainnya. Semakin bervariasinya sebuah perangkat lunak membuat semakin banyak potensi adanya celah keamanan. Hal ini dapat mengancam aspek keamanan informasi yang dikenal dengan prinsip CIA (*Confidentiality, Integrity,* dan *Availability*), di mana kerahasiaan, integritas, serta ketersediaan data dan sistem harus dijaga. Menemukan *bug* atau kerentanan sejak awal sangat penting dalam manajemen proses [1]. Dalam konteks keamanan perangkat lunak, hal ini menjadi bagian dari fase *security assessment*, yang mencakup kegiatan identifikasi, analisis, dan evaluasi terhadap potensi kerentanan yang dapat dieksploitasi oleh pihak yang tidak bertanggung jawab. Fase ini umumnya dilakukan melalui beberapa pendekatan, seperti *static analysis* dan *dynamic testing* yang bertujuan untuk memastikan bahwa perangkat lunak memiliki tingkat keamanan yang memadai sebelum digunakan secara luas. Identifikasi *bug* pada perangkat lunak, baik yang bersifat modern maupun lawas, dapat dilakukan melalui analisis kode sumber maupun analisis manual. Namun, kedua metode tersebut memiliki kelemahan, yaitu cenderung membosankan, membutuhkan waktu yang cukup lama, serta kurang efisien apabila diterapkan pada perangkat lunak dengan ukuran dan kompleksitas yang besar. Hal ini menimbulkan permasalahan dalam hal efisiensi waktu dan sumber daya, sehingga menjadikan metode tersebut kurang ideal untuk diterapkan dalam pengujian perangkat lunak modern. Salah satu pendekatan alternatif yang banyak digunakan dalam proses *security assessment* adalah *fuzzing*.

Fuzzing merupakan sebuah teknik yang dapat menemukan *bug* secara otomatis [2]. *Fuzzing* merupakan teknik paling dominan dan umum

dalam melakukan deteksi kerentanan [3]. Ketika melakukan pengujian *Fuzzing* pada perangkat lunak, digunakan pendekatan random input, dimasukkan input yang besar, selanjutnya dilakukan input kesalahan format yang dapat menimbulkan kesalahan atau *bug* pada perangkat lunak [4]. Kesalahan maupun *bug* yang ditemukan oleh Fuzzing ini dapat dimunculkan ketika *software* dikompilasi menggunakan *Address Sanitizer* (ASan). *Address Sanitizer* dapat didefinisikan sebagai sebuah alat pendeteksi kesalahan memori pada sebuah aplikasi. Pada dasarnya alat ini dirancang oleh *Google* untuk melakukan deteksi kesalahan pada memori dan dapat digunakan untuk berbagai macam arsitektur komputer seperti *x86, MIPS, ARM* dan *PowerPC*. Alat tersebut akan mengeluarkan output kesalahan ketika terjadi masalah pada memori masalah tersebut meliputi *Stack Buffer Overflow, Heap Buffer Overflow, Use-after-Free, Out-of-Bound, Use After Return, Memory Leaks* maupun kesalahan memori lainnya [5]. *Address Sanitizer* dapat digunakan ketika menerapkan *White-box Fuzzing*. *White-box fuzzing* merupakan teknik *fuzzing* yang mengharuskan memiliki akses menuju kode sumber sehingga kode sumber dapat dikompilasi dengan *Address Sanitizer* [6].

Dalam penggunaan *Address Sanitizer* dibutuhkan *compiler* yang mendukung fitur *Address Sanitizer*. Salah satu *compiler* yang mendukung penggunaan *Address Sanitizer* adalah *GCC (GNU Compiler Collection)*.Compiler ini merupakan compiler yang paling populer untuk kompilasi C/C++ dan dalam waktu 5 tahun terakhir telah ditemukan total 29 laporan *bug* dari penggunaan GCC-ASan [7]

Dalam melakukan implementasi *fuzzing* penulis menggunakan tool yang bernama *American Fuzzy Looper* (AFL++). *American Fuzzy Looper* adalah *fuzzer* mutasional dan termasuk *coverage-guided fuzzer,* AFL akan melakukan berbagai inputan acak yang dilakukan mutasi dengan cara menguji kasus yang sudah ada dan menciptakan kasus baru. Kasus baru tersebut didapatkan dari cakupan kode yang ada pada perangkat lunak yang kemudian AFL menggunakan informasi tersebut

untuk memutasi input dan membuat kasus baru dari program yang belum tereksekusi. *American Fuzzy Looper* juga merupakan alat *coverage-guided fuzzer* paling sukses sepanjang masa [2]. *American Fuzzy Looper* telah berhasil menemukan 330 kerentanan dalam kurun waktu 4 tahun dari tahun 2013 - 2017 pada 70 program yang berbeda [6]. Maka dari itu penulis memutuskan menggunakan alat AFL++ dalam melakukan penelitiannya.

1.2 Rumusan Masalah

Berdasarkan latar belakang diatas dapat dirumuskan suatu permasalahan yaitu sebagai berikut :

1. Berapa banyak *crash* yang dihasilkan oleh AFL++ ketika melakukan *fuzzing*?
2. Apa saja kerentanan yang ditemukan oleh AFL++ ?
3. Bagaimana cara melakukan remediasi terhadap kerentanan yang ditemukan oleh AFL++?

1.3 Batasan Masalah

Batasan masalah dibuat untuk membantu peneliti tetap fokus pada topik penelitian. Oleh karena itu, peneliti memfokuskan kepada pembahasan masalah- masalah pokok yang dibatasi dalam konteks permasalahan adalah sebagai berikut:

1. Penelitian berfokus pada Implementasi *Fuzzing* menggunakan AFL++ (*American Fuzzy Looping*).
2. Perangkat Lunak yang digunakan ialah perangkat lunak yang menggunakan bahasa pemrograman C/C++ yang dapat dikompilasi menggunakan GCC.
3. *Fuzzing* hanya dilakukan pada perangkat lunak yang berjalan pada sistem operasi *Linux*.
4. Metode *fuzzing* yang digunakan adalah *Coverage-Guided Fuzzing* .
5. Perangkat lunak yang digunakan sebagai objek penelitian adalah *pdfcrack* yang dilisensikan dengan *GNU General Public License v2.0* ,

yang memperbolehkan untuk dipelajari, dimodifikasi, dijalankan, serta dianalisis kode sumbernya .

1.4 Tujuan Penelitian

Tujuan yang akan dicapai oleh peneliti dalam penelitiannya adalah untuk mengetahui jumlah input *crash* yang dihasilkan oleh AFL++ selama proses *fuzzing*, mengidentifikasi kerentanan keamanan yang ditemukan oleh AFL++ dari input *crash* yang dihasilkan, dan mengevaluasi apakah semua input crash yang dihasilkan oleh AFL++ berhubungan langsung dengan kerentanan keamanan atau terdapat input *crash* yang tidak mengindikasikan adanya kerentanan.

1.5 Manfaat Penelitian

Manfaat pada penelitian ini adalah sebagai berikut:

a. Memberikan informasi yang berguna bagi *developer* dalam mengidentifikasi secara dini kerentanan keamanan yang berpotensi dieksploitasi, berdasarkan analisis input *crash* yang ditemukan selama proses *fuzzing* menggunakan AFL++.
b. Menambah pengetahuan bagi pembaca serta pegiat keamanan siber.
c. Mengurangi kerugian yang disebabkan oleh serangan siber pada saat perangkat lunak sudah rilis.
d. Skripsi ini dapat dimanfaatkan sebagai bahan referensi bagi mahasiswa yang akan menyelesaikan skripsi pada periode yang akan datang dan sebagai acuan sejauh mana pemahaman dan penguasaan mahasiswa terhadap perkuliahan yang diberikan sehingga dapat dijadikan sebagai bahan evaluasi kampus dalam membangun pengetahuan.

1.6 Sistematika Penulisan

Peneliti menyusun penelitian ini secara sistematis agar mudah dibaca dan dipahami. Berikut adalah sistematika penulisan pada penelitian ini:

BAB I PENDAHULUAN

Berisi Latar belakang masalah, rumusan masalah, Batasan masalah, tujuan penelitian, manfaat penelitian, dan sistematika penulisan.

BAB II TINJAUAN PUSTAKA

Berisi studi literatur dan penjelasan mengenai teori-teori yang mendukung proses pembuatan laporan skripsi ini.

BAB III METODE PENELITIAN.

Berisi alur penelitian, analisis kebutuhan, dan proses fuzzing yang dilakukan.

BAB IV HASIL DAN PEMBAHASAN

Bab ini memuat mengenai serangkaian tahapan yang peneliti lakukan mencakup hasil dan pembahasan mengenai implementasi *fuzzing*.

BAB V PENUTUP

Berisi kesimpulan dan saran yang dapat peneliti rangkum selama proses penelitian