

PERANCANGAN DAN IMPLEMENTASI PENGUJIAN FUZZ PARALEL

TESIS

**Karya tulis sebagai salah satu syarat
untuk memperoleh gelar Magister dari
Institut Teknologi Bandung**

**Oleh
BAYU MAHENDRA
NIM: 23218030
(Program Studi Magister Teknik Elektro)**



**INSTITUT TEKNOLOGI BANDUNG
Juni 2020**

ABSTRAK

PERANCANGAN DAN IMPLEMENTASI PENGUJIAN FUZZ PARALEL

Oleh
Bayu Mahendra
NIM: 23218030
(Program Studi Magister Teknik Elektro)

Keamanan perangkat lunak merupakan aspek penting dalam menentukan kualitas perangkat lunak sehingga aspek keamanan harus dipertimbangkan dalam proses pengembangan perangkat lunak. Pengujian perangkat lunak dengan cara manual sangat menyita waktu. Terkadang proses pengujian perangkat lunak lebih lama dibandingkan proses pengembangan perangkat lunak. Oleh karena itu, pengujian secara otomatis perlu dilakukan untuk mempersingkat waktu pengujian.

Fuzz testing (fuzzing) adalah teknik pengujian keamanan perangkat lunak secara otomatis dengan cara mengirimkan *seed* atau data masukan abnormal ke program yang diuji. *American Fuzzy Lop (AFL)* adalah salah satu *fuzzer* yang populer saat ini. AFL adalah *grey-box fuzzer* yang menggunakan teknik mutasi untuk menghasilkan data masukan abnormal. Data masukan abnormal tersebut dihasilkan oleh AFL menggunakan teknik mutasi deterministik dan non deterministik (acak). AFL sudah mendukung mode tunggal dan mode paralel. Pada mode paralel, terdapat perbedaan antara *master node* dan *slave node* terkait pembuatan data masukan. *Master node* menggunakan mutasi deterministik dan mutasi acak sedangkan *slave node* hanya menggunakan mutasi acak. Penggunaan mutasi deterministik pada kedua *node* mengakibatkan duplikasi data masukan sehingga kurang efisien dalam penggunaan sumber daya komputasi.

Pada penelitian ini, diajukan rancangan sistem untuk *fuzzing* paralel. Teknologi virtualisasi dimanfaatkan untuk melakukan *fuzzing* secara paralel. Terdapat sebuah *master node* dan 1 atau lebih *container* sebagai *slave node*. *Master node* terdiri dari aplikasi berbasis *python*, *database* dan AFL. Beberapa AFL di *container* digunakan untuk melakukan proses *fuzzing* yang dikelola oleh *master node*. Sebuah aplikasi yang bertindak sebagai agen digunakan untuk mengelola sinkronisasi informasi antar AFL. Rancangan yang diajukan mengimplementasikan kedua teknik mutasi di setiap *node*. Selain itu, sinkronisasi informasi seperti *seed* dan *path coverage* diperlukan untuk mencegah duplikasi tugas sehingga proses *fuzzing* bisa lebih efisien dan lebih cepat untuk menemukan kecacatan atau kerentanan pada program yang diuji.

Kata kunci: Keamanan perangkat lunak, *fuzzing* paralel, *container*.

ABSTRACT

DESIGN AND IMPLEMENTATION OF PARALLEL FUZZ TESTING

By

Bayu Mahendra

NIM: 23218030

(Master's Program in Electrical Engineering)

Software security is an important aspect in determining software quality. It must be considered in the software development process. Testing software manually is a time-consuming task. Sometimes the software testing process takes longer than the software development process. Therefore, to speed up the time testing needs to be done automatically.

Fuzz testing (fuzzing) is an automated software testing technique by sending abnormal data to the tested program. American Fuzzy Lop (AFL) is one of the most popular fuzzer. AFL is a grey-box fuzzer with mutation technique to generate abnormal input for feeding the tested program. Abnormal input is generated by AFL using deterministic mutation and random mutation. AFL already supports single mode and parallel mode. In parallel mode, there is a difference between the master node and the slave node in generating abnormal input. Master node uses deterministic mutation and random mutation but slave node only uses random mutation. Performing deterministic mutation in all nodes causes duplicate abnormal input. Hence, computing resource is utilized inefficient.

In this research, we propose a design system for parallel fuzzing. Virtualization technology is utilized to perform parallel fuzzing. There is a master node and one or more containers as slave node. Master node consists of python-based applications, database and AFL. AFL on master node is used to select initial seed for others. Multiple AFL instances on container are utilized to run fuzzing process. Those are controlled by master node. An agent is used to manage information synchronization between different AFL instances. Our proposed design implement both mutation techniques in every node. Besides that, information synchronization such as seed and path coverage is required to prevent task duplication. Therefore, fuzzing process is more efficient and running faster to find security flaws in the tested program.

Keywords: Software security, parallel fuzzing, container.

PERANCANGAN DAN IMPLEMENTASI PENGUJIAN FUZZ PARALEL

Oleh
Bayu Mahendra
NIM: 23218030
(Program Studi Magister Teknik Elektro)

Institut Teknologi Bandung

Menyetujui
Dosen Pembimbing

Tanggal 22 Juni 2020

(Ir. Budi Rahardjo, M.Sc., Ph.D.)

*Dipersembahkan kepada keluarga, tim AMS Disjatim, serta PT PLN (Persero)
yang senantiasa mendukung lahir dan batin.*

KATA PENGANTAR

Puji syukur kehadiran Allah SWT, berkat rahmat dan hidayah Nya penelitian ini dapat diselesaikan. Selama proses penyusunan tesis ini, penulis mendapatkan bantuan dan dukungan yang luar biasa dari berbagai pihak. Oleh karena itu, penulis ingin mengucapkan terima kasih kepada:

1. Bapak Ir. Budi Rahardjo M.Sc., Ph.D selaku dosen pembimbing yang selalu menginspirasi, memotivasi, mendukung dan membagikan banyak sekali ilmu selama penelitian serta penyusunan laporan ini.
2. Bapak Prof. Dr. Ing. Ir. Suhardi, MT selaku dosen wali opsi RMKI yang selalu membantu penulis dalam kegiatan akademik.
3. Bapak/Ibu dosen Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung yang telah memberikan ilmu pengetahuan serta seluruh staf Tata Usaha Teknik Elektro atas segala bantuannya selama masa perkuliahan.
4. Manajemen PT PLN (Persero) atas kesempatan belajar yang diberikan.
5. Tim AMS PT PLN (Persero) Distribusi Jatim, khususnya Alm. Bapak Rasid Basuki dan Alm. Bapak Anton S.B. Utomo atas semua kebaikan dan dukungannya.
6. Rekan – rekan Kerma PLN-ITB *batch* 8 atas segala dukungannya.
7. Rekan – rekan mahasiswa RMKI Angkatan 2018 yang telah membantu proses belajar selama menjadi mahasiswa pasca sarjana di ITB.

Penulis menyadari bahwa penelitian ini bukanlah tanpa kekurangan. Untuk itu, kritik dan saran sangat diharapkan dalam mengembangkan penelitian. Akhir kata, semoga penelitian ini dapat memberikan manfaat bagi para pembaca.

DAFTAR ISI

ABSTRAK	i
ABSTRACT	ii
HALAMAN PENGESAHAN.....	iii
KATA PENGANTAR	v
DAFTAR ISI	vi
DAFTAR LAMPIRAN.....	viii
DAFTAR GAMBAR DAN ILUSTRASI.....	ix
DAFTAR TABEL.....	x
DAFTAR SINGKATAN DAN LAMBANG	xi
Bab I Pendahuluan	1
I.1 Latar Belakang	1
I.2 Rumusan Masalah	3
I.3 Pertanyaan Penelitian	4
I.4 Tujuan Penelitian	4
I.5 Batasan Masalah	4
I.6 Metodologi Penelitian	4
I.6.1 <i>Concept Development</i>	5
I.6.2 <i>Engineering Development</i>	6
I.7 Sistematika Penulisan	7
Bab II Tinjauan Pustaka.....	9
II.1 Pengujian Keamanan Perangkat Lunak	9
II.2 <i>Fuzzing</i>	9
II.3 <i>American Fuzzy Lop (AFL)</i>	12
II.4 <i>Container</i>	15
II.5 Peta Literatur	17
Bab III Perancangan Sistem.....	20
III.1 Analisis Kebutuhan Sistem.....	20
III.1.1 Analisis Kebutuhan Pengguna	20
III.1.2 Analisis Kebutuhan Fungsional	21
III.1.3 Analisis Kebutuhan Non Fungsional	22
III.1.4 Analisis Kebutuhan Perangkat Lunak.....	22
III.2 Arsitektur Sistem	22
III.3 Desain Sistem	23
III.3.1 <i>Use Case Diagram</i>	23
III.3.2 <i>Activity Diagram</i>	24
III.3.3 <i>Desain Database</i>	30
Bab IV Implementasi dan Pengujian	32
IV.1 Implementasi Sistem	32
IV.1.1 Implementasi Aplikasi Master	32
IV.1.2 Implementasi Agen Sinkronisasi <i>Seed</i>	33
IV.1.3 Implementasi Agen Sinkronisasi <i>Path Coverage</i>	34
IV.2 Pengujian	35
IV.2.1 Pengujian Fungsional	35
IV.2.2 Pengujian Kualitas	41
Bab V Penutup	43

V.1Kesimpulan	43
V.2Saran.....	43
DAFTAR PUSTAKA	44
LAMPIRAN	46

DAFTAR LAMPIRAN

Lampiran A Instalasi Kebutuhan Perangkat Lunak	47
A.1 Docker	47
A.2 AFL	47
A.3 SQLite	49
Lampiran B Kode Sumber	50
B.1 Program yang Diuji	50
B.2 Aplikasi Master	51
B.2 Agen Sinkronisasi <i>Seed</i>	59
B.3 Agen Sinkronisasi <i>Path Coverage</i>	63

DAFTAR GAMBAR DAN ILUSTRASI

Gambar I.1	Laporan kerentanan per Tahun yang terdaftar di CVE	1
Gambar I.2	Model siklus hidup sistem	5
Gambar I.3	Fase <i>concept development</i>	5
Gambar I.4	Fase <i>engineering development</i>	6
Gambar I.5	Model <i>Waterfall SDLC</i>	7
Gambar II.1	Microsoft SDL	9
Gambar II.2	Iterasi proses <i>fuzzing</i>	10
Gambar II.3	<i>Fuzzer</i> berdasarkan program yang diuji	10
Gambar II.4	Klasifikasi <i>fuzzing</i>	11
Gambar II.5	<i>Compile</i> program yang diuji menggunakan <i>compiler</i> bawaan AFL	12
Gambar II.6	Proses <i>fuzzing</i> pada AFL	13
Gambar II.7	Gambaran umum <i>path coverage</i>	15
Gambar II.8	Perbandingan arsitektur antara (a) <i>container</i> dan (b) <i>VM</i>	16
Gambar II.9	Arsitektur docker	17
Gambar II.10	Peta Literatur	18
Gambar III.1	Arsitektur sistem	23
Gambar III.2	<i>Use case diagram</i>	24
Gambar III.3	<i>Activity diagram</i> menentukan <i>seed</i> untuk <i>fuzzer</i> pertama	25
Gambar III.4	<i>Activity diagram</i> menentukan <i>seed</i>	25
Gambar III.5	<i>Activity Diagram</i> menentukan program target	26
Gambar III.6	<i>Activity diagram</i> menjalankan proses <i>fuzzing</i>	28
Gambar III.7	<i>Activity diagram</i> menghentikan proses <i>fuzzing</i>	29
Gambar III.8	<i>Activity Diagram</i> untuk sinkronisasi <i>seed</i>	29
Gambar III.9	<i>Activity Diagram</i> untuk sinkronisasi <i>path coverage</i>	30
Gambar IV.1	Grafik hasil pengujian kualitas	42

DAFTAR TABEL

Tabel II.1	Perbandingan hasil mutasi	12
Tabel II.2	Keluaran AFL	14
Tabel III.1	Kebutuhan fungsional.....	21
Tabel III.2	Kebutuhan perangkat lunak	22
Tabel III.3	Tabel <i>seed</i>	30
Tabel III.4	Tabel stats	31
Tabel IV.1	Spesifikasi komputer	32
Tabel IV.2	Skenario pengujian fungsional	35
Tabel IV.3	Hasil pengujian fungsional	36
Tabel IV.4	Hasil pengujian kualitas	41

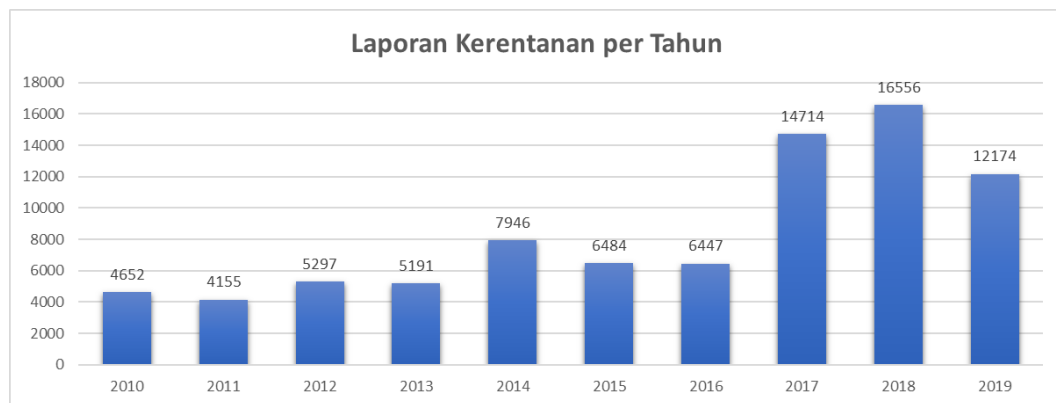
DAFTAR SINGKATAN DAN LAMBANG

SINGKATAN	Nama	Pemakaian pertama kali pada halaman
CVE	<i>Common Vulnerabilities and Exposures</i>	1
AFL	<i>American Fuzzy Lop</i>	2
SDLC	<i>Software Development Lifecycle</i>	6
UML	<i>Unified Modelling Language</i>	7
SDL	<i>Secure Development Lifecycle</i>	9
API	<i>Application Programming Interface</i>	10
VM	<i>Virtual Machine</i>	15
PaaS	<i>Platform as a Service</i>	16

Bab I Pendahuluan

I.1 Latar Belakang

Keamanan merupakan aspek penting dalam proses pengembangan perangkat lunak. Gambar I.1 menunjukkan jumlah laporan kerentanan yang terdaftar di *Common Vulnerabilities and Exposures (CVE)* dalam 10 tahun terakhir. Berdasarkan data tersebut, terdapat 83.616 laporan yang terdaftar di CVE dalam kurun waktu 2010 – 2019 yang berarti ada 23 laporan per hari. Pada tahun 2019 bahkan terdapat 12.174 laporan yang berarti 34 laporan per hari. Kualitas perangkat lunak salah satunya ditentukan oleh mutu keamanan yang dimiliki oleh perangkat lunak. Keamanan perangkat lunak adalah gagasan rekayasa perangkat lunak sehingga perangkat lunak masih bisa berfungsi dengan normal meskipun perangkat lunak dalam kondisi dibawah serangan yang berbahaya [1]. Keamanan perangkat lunak harus dipertimbangkan dalam proses pengembangan perangkat lunak meliputi proses perancangan, pembuatan dan pengujian perangkat lunak sehingga dihasilkan perangkat lunak yang aman.



Gambar I.1 Laporan kerentanan per Tahun yang terdaftar di CVE

Selama ini keamanan perangkat lunak masih belum mendapat perhatian khusus dari pengembang meskipun aspek penting pada permasalahan keamanan komputer adalah keamanan perangkat lunak. Penyerang sering meretas sistem dengan memanfaatkan celah keamanan pada perangkat lunak. Salah satu faktor penyebabnya adalah proses pengujian keamanan perangkat lunak dilakukan secara manual yang sangat menyita waktu, tenaga dan biaya. Terkadang proses pengujian perangkat lunak lebih melelahkan dibanding proses pengembangannya.

Fuzzer adalah sebuah perangkat lunak yang digunakan untuk melakukan pengujian keamanan perangkat lunak secara otomatis dengan teknik *fuzzing*. *Fuzz testing* atau biasa disingkat *fuzzing* adalah teknik deteksi kerentanan yang sangat efektif dan banyak digunakan untuk menemukan *bug* keamanan dan kerentanan dalam perangkat lunak. Ide utama di balik *fuzzing* adalah mengirimkan input tidak valid ke sebuah program target secara konstan dengan harapan dapat memicu kegagalan fungsi perangkat lunak [2].

Teknik *fuzzing* pertama kali diperkenalkan oleh Profesor Barton Miller pada tahun 1990 [3]. Penelitian itu didasarkan pada peristiwa yang terjadi pada tahun 1989 ketika beliau mengakses komputer yang ada di kampus melalui modem telepon pada saat kondisi hujan badai. Hujan badai tersebut menyebabkan gangguan koneksi dan terjadinya pengiriman karakter - karakter acak pada mesin UNIX yang sedang diakses sehingga menyebabkan mesin UNIX mengalami *crash*. Kejadian tersebut menginspirasi Profesor Barton Miller untuk melakukan penelitian terkait generator karakter acak dan menguji sebanyak mungkin program utilitas UNIX. Adapun aliran acak yang dihasilkan oleh generator dinamakan dengan *fuzz*.

Salah satu *fuzzer* yang populer saat ini adalah *American Fuzzy Lop* (AFL). AFL telah mendapatkan banyak trofi dalam menemukan kerentanan pada perangkat lunak. Kecepatan eksekusi dan kemudahan pemakaian merupakan kelebihan dari AFL yang telah meningkatkan popularitas AFL secara signifikan. AFL pertama kali dikembangkan oleh Michal Zalewski saat bekerja di Google. Saat ini, AFL telah menjadi proyek sumber terbuka dan kode sumber AFL tersedia di situs github.

AFL adalah *grey-box fuzzer* untuk menguji keamanan perangkat lunak yang dikembangkan menggunakan bahasa pemrograman C/C++. AFL merupakan sebuah *fuzzer file* karena program target dari AFL adalah sebuah file. Modifikasi data input dilakukan dengan memanfaatkan teknik mutasi. Mutasi tersebut dilakukan melalui 2 tahap yaitu tahap deterministik dan tahap non deterministik [4]. Dukungan mode tunggal dan paralel telah tersedia pada AFL. AFL mode paralel

terdiri dari 1 *master node* dan satu atau beberapa *slave node*. Perbedaan antara mode tunggal dan paralel terletak pada penerapan strategi mutasi. Pada mode paralel, AFL membedakan strategi mutasi antara *master node* dan *slave node* dimana *master node* melakukan mutasi deterministik dan non deterministik sedangkan *slave node* hanya melakukan mutasi non deterministik. Hal tersebut dilakukan untuk menghindari duplikasi *seed* antar *fuzzer*. AFL mode paralel mengkoordinasikan beberapa proses *fuzzing* dengan cara melakukan sinkronisasi informasi antar *fuzzer* [5].

Dalam meningkatkan kualitas *fuzzer*, terdapat 2 pendekatan yang berbeda. Pendekatan pertama diterapkan pada proses pembuatan dan pemilihan *seed* sedangkan pendekatan kedua dilakukan dengan menjalankan proses *fuzzing* secara paralel. Pada mode paralel, proses *fuzzing* terhadap satu program target yang sama dilakukan oleh beberapa *fuzzer*.

Saat ini teknologi virtualisasi semakin berkembang seiring dengan kapabilitas sumber daya komputasi yang semakin baik. Selain itu, keberadaan virtualisasi menjadikan komputasi paralel telah dimanfaatkan secara luas. Salah satu teknologi virtualisasi yang populer saat ini adalah teknologi *container*. Di lain sisi, waktu adalah sesuatu yang sangat berharga pada proses *fuzzing*. Dengan memanfaatkan teknologi *container*, proses *fuzzing* bisa dilakukan secara paralel untuk mempersingkat waktu *fuzzing* dalam menemukan kerentanan pada perangkat lunak.

I.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dipaparkan, maka rumusan masalah pada penelitian ini adalah sebagai berikut:

1. kegagalan fungsi pada *master node* ketika proses *fuzzing* sedang berjalan mengakibatkan proses *fuzzing* berjalan tanpa mutasi deterministik;
2. penerapan mutasi deterministik di *master node* dan *slave node* dapat mengakibatkan terjadinya duplikasi *seed*.

I.3 Pertanyaan Penelitian

Pertanyaan penelitian dari tesis ini adalah sebagai berikut:

1. Bagaimana merancang sistem *fuzzing* paralel yang tetap menjalankan kedua mutasi meskipun *master node* mengalami kegagalan fungsi?
2. Bagaimana metode sinkronisasi informasi antar *fuzzer*?
3. Apakah penerapan mutasi deterministik pada setiap *fuzzer* dapat mempersingkat proses *fuzzing* dalam menemukan kerentanan?

I.4 Tujuan Penelitian

Tujuan dari penelitian ini adalah sebagai berikut:

1. menghasilkan rancangan sistem yang menerapkan mutasi deterministik di setiap *fuzzer*;
2. menghasilkan rancangan sistem yang dapat melakukan sinkronisasi informasi antar *fuzzer*;
3. mengevaluasi hasil pencapaian proses *fuzzing* paralel dengan kedua mutasi di setiap *fuzzer*.

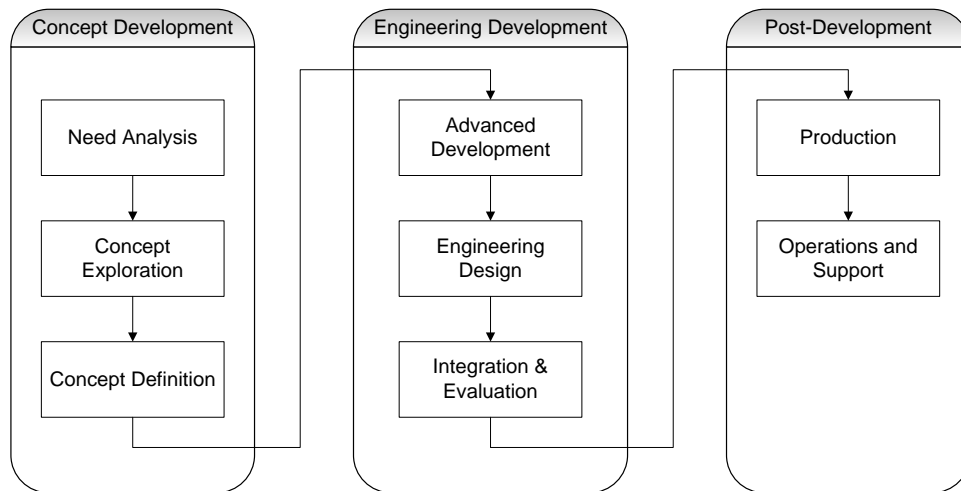
I.5 Batasan Masalah

Batasan masalah dari penelitian ini adalah sebagai berikut:

1. jumlah *fuzzer* yang dipakai sebanyak 2 *node*;
2. *seed* yang digunakan berupa data tekstual;
3. program yang diuji adalah perangkat lunak yang dikembangkan dengan bahasa pemrograman C;
4. program yang diuji memiliki 1 kerentanan yang harus ditemukan oleh *fuzzer*;
5. proses pengujian membutuhkan *source code*.

1.6 Metodologi Penelitian

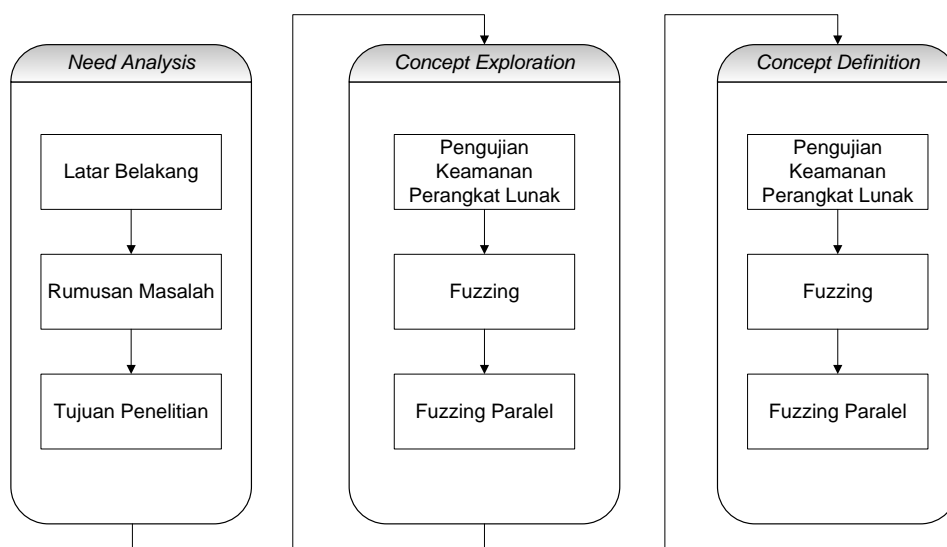
Dalam penelitian ini, metodologi yang digunakan adalah *System Engineering Principles and Practice* [6]. Metodologi ini terdiri dari tiga tahapan. Namun, untuk penelitian ini hanya dilakukan sampai tahapan kedua yaitu *engineering development*. Tahapan dalam metodologi ini dapat dilihat pada gambar I.2.



Gambar I.2 Model siklus hidup sistem

I.6.1 *Concept Development*

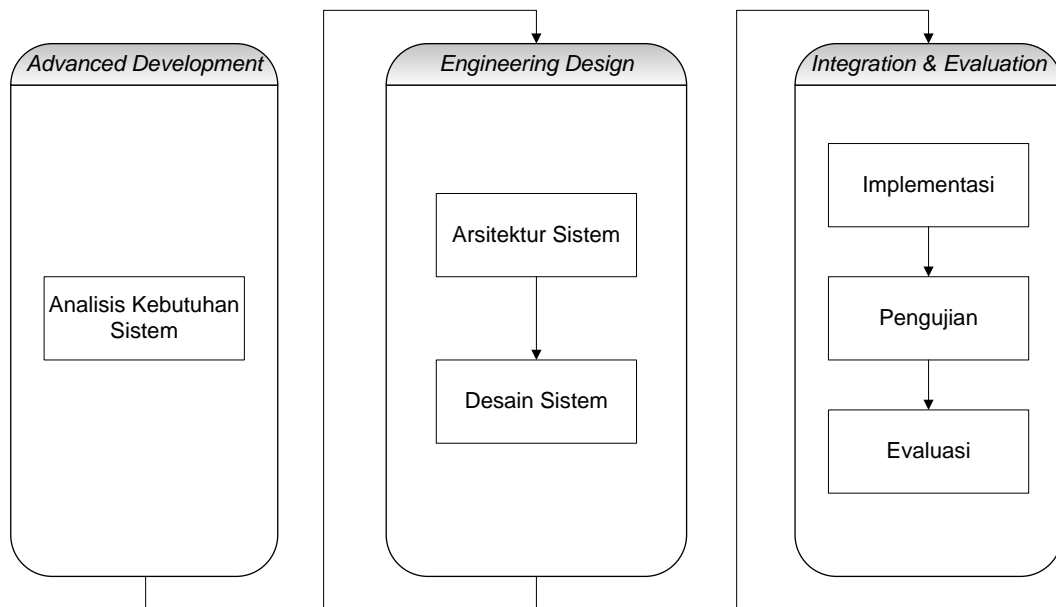
Fase *concept development* merupakan tahap awal dari penelitian yang terdiri dari tiga tahapan yaitu *need analysis*, *concept exploration* dan *concept definition*. Pada tahapan *need analysis*, permasalahan yang akan dijadikan topik penelitian akan dirumuskan dan didefinisikan sehingga dapat menjelaskan kebutuhan akan sistem baru yang menjadi tujuan penelitian. Pada tahapan *concept exploration*, dikaji konsep – konsep sistem potensial yang dapat mendukung penelitian. Pada tahapan ini, akan dipelajari literatur yang berhubungan dengan penelitian. Pada tahapan *concept definition*, ditentukan konsep sistem yang akan dikembangkan untuk mencapai tujuan penelitian. Gambar I.3 menunjukkan fase *concept development*.



Gambar I.3 Fase *concept development*

I.6.2 Engineering Development

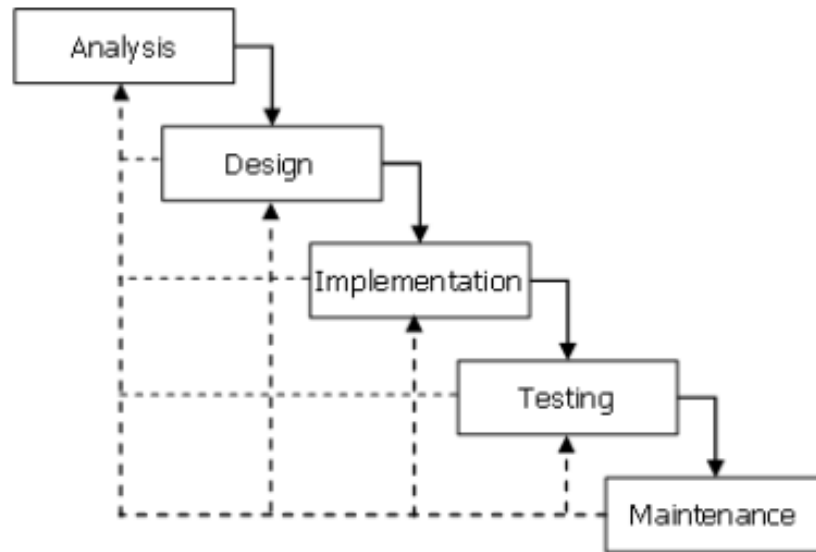
Fase *engineering development* merupakan tahap akhir dari penelitian yang terdiri dari tiga tahapan yaitu *advance development*, *engineering design* dan *integration and evaluation*. Pada fase ini, konsep yang telah ditentukan pada fase sebelumnya akan ditransformasikan menjadi sebuah desain sistem yang nantinya akan diimplementasikan, diuji dan dievaluasi. Gambar I.4 menunjukkan fase *engineering development*.



Gambar I.4 Fase *engineering development*

Pada tahapan *advance development*, dilakukan validasi terhadap konsep yang telah ditentukan pada tahapan sebelumnya. Model *Waterfall Software Development Life Cycle (SDLC)* akan digunakan sebagai metode pengembangan sistem sehingga konsep yang dibuat bisa menjadi sebuah produk. Model *Waterfall SDLC* adalah proses pengembangan perangkat lunak yang terdiri dari beberapa fase yang berurutan dimana setiap fase harus diselesaikan satu demi satu dan bergerak ke fase berikutnya hanya jika fase sebelumnya telah dilakukan sepenuhnya [7]. Model ini bersifat rekursif karena setiap fase dapat diulang tanpa henti hingga perangkat lunak yang dikembangkan disempurnakan. Model *Waterfall SDLC* terdiri dari 5 fase yaitu analisis, desain, implementasi, pengujian dan pemeliharaan. Model ini cocok dijadikan acuan perancangan sistem karena memiliki tahapan yang berurutan mulai

dari analisis kebutuhan, desain, implementasi dan pengujian sistem. Gambar I.5 menggambarkan model *Waterfall SDLC*.



Gambar I.5 Model *Waterfall SDLC*

Pada tahapan *engineering design*, ditentukan arsitektur dan desain sistem yang akan dikembangkan menjadi sebuah prototipe. Dalam hal ini, perancangan sistem menggunakan *Unified Modelling Language (UML)*. *UML* adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun sistem perangkat lunak. Pada tahapan *integration and evaluation*, dilakukan implementasi dan pengujian sistem sekaligus menganalisis hasil yang dicapai.

I.7 Sistematika Penulisan

Sistematika penulisan yang diterapkan dalam penelitian ini terdiri dari empat bagian, yaitu:

- Bab I Pendahuluan

Pada bagian pendahuluan dikemukakan latar belakang, rumusan masalah, batasan masalah, pertanyaan penelitian, tujuan penelitian dan metodologi penelitian.

- Bab II Tinjauan Pustaka dan Peta Literatur

Pada bagian tinjauan pustaka dan peta literatur dikemukakan beberapa teori yang mendukung penelitian. Teori – teori tersebut diperlukan untuk melakukan proses *fuzzing* secara paralel dengan memanfaatkan beberapa

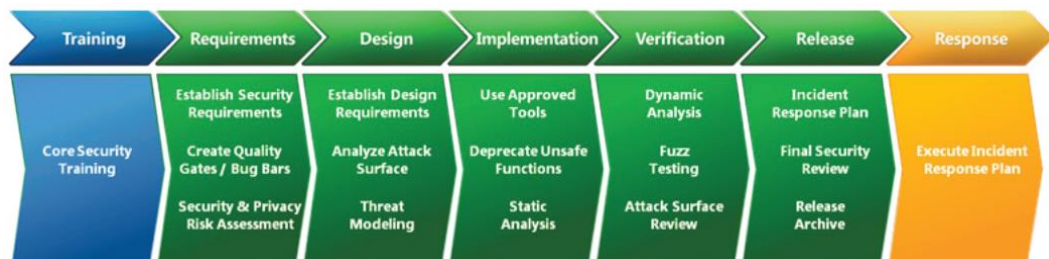
fuzzer. Selain itu, dipaparkan penelitian – penelitian yang sudah pernah dilakukan sebelumnya terkait *fuzzing* pada mode paralel. Peta literatur dibuat berupa ringkasan visual yang menggambarkan hubungan penelitian dengan literatur – literatur yang sudah ada.

- Bab III Perancangan Sistem
Pada bagian perancangan diuraikan gambaran sistem yang dibangun meliputi deskripsi dan alur sistem.
- Bab IV Implementasi dan Pengujian
Pada bagian implementasi dan pengujian disampaikan tahapan implementasi dan hasil pengujian sistem yang telah dibuat beserta analisisnya.
- Bab V Penutup
Pada bagian penutup disampaikan kesimpulan dan saran dari keseluruhan penelitian yang sudah dilakukan untuk menunjang penelitian lebih lanjut.
- Daftar Pustaka
Memuat referensi tulisan yang menjadi sumber acuan tesis ini dari mulai perancangan sampai penyusunan laporan akhir.
- Lampiran
Berisi kode sumber program yang telah dibuat dan kelengkapan penelitian.

Bab II Tinjauan Pustaka

II.1 Pengujian Keamanan Perangkat Lunak

Pengujian perangkat lunak adalah kegiatan investigasi yang dilakukan untuk memberikan informasi kepada pihak terkait tentang kualitas dari produk atau layanan yang diuji. Sedangkan pengujian keamanan perangkat lunak adalah suatu proses untuk menentukan bahwa sebuah sistem informasi melindungi data dan menjaga fungsionalitas sistem [8]. Untuk mencapai tujuan keamanan perangkat lunak, Microsoft mengembangkan *Secure Development Lifecycle (SDL)*. Microsoft SDL adalah proses jaminan keamanan untuk pengembangan perangkat lunak dengan mengimplementasikan keamanan kedalam setiap fase pengembangan perangkat lunak dan memberikan panduan dan perlindungan keamanan secara mendalam. SDL adalah serangkaian prosedur yang melibatkan penguji, pengembang, manajer program dan arsitek sistem bekerja bersama dengan tim keamanan produk. Di dalam proses Microsoft SDL terdapat beberapa tahapan dimana *fuzzing* berada di tahapan verifikasi seperti dijelaskan oleh gambar II.1.

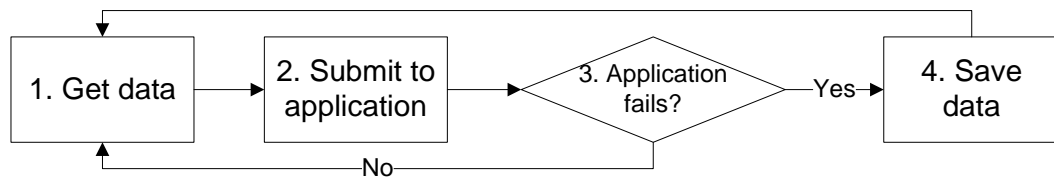


Gambar II.1 Microsoft SDL

II.2 Fuzzing

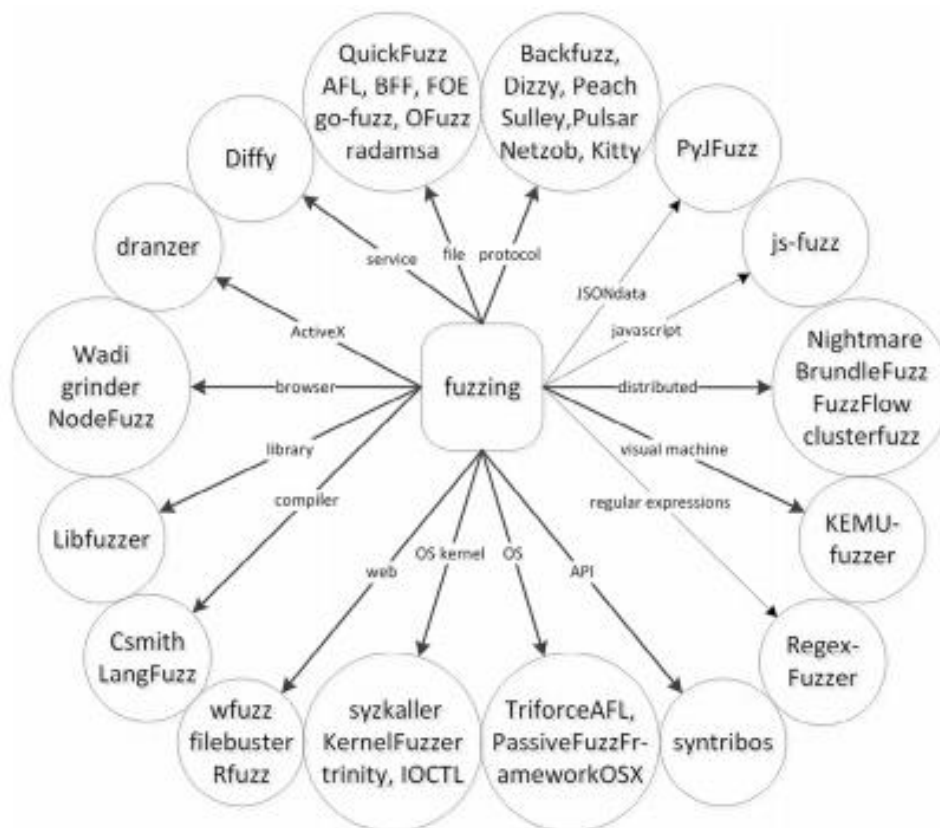
Fuzzing adalah teknik pengujian otomatis yang menggunakan data tidak valid sebagai data input untuk memastikan tidak adanya kerentanan yang dapat dieksploitasi [9]. *Fuzzing* terkenal akan kemampuannya dalam mengungkap *bug* korupsi memori, yang cenderung memiliki implikasi keamanan [8]. Gambar II.2 menunjukkan alur proses *fuzzing* yang dimulai dari pembuatan data input tidak valid yang nantinya akan diberikan ke program yang diuji untuk diproses. Apabila program berfungsi normal, maka dilakukan proses *fuzzing* berikutnya dengan data

input yang berbeda. Namun, apabila program gagal berfungsi, hasil dari proses *fuzzing* akan disimpan untuk dianalisis lebih lanjut.



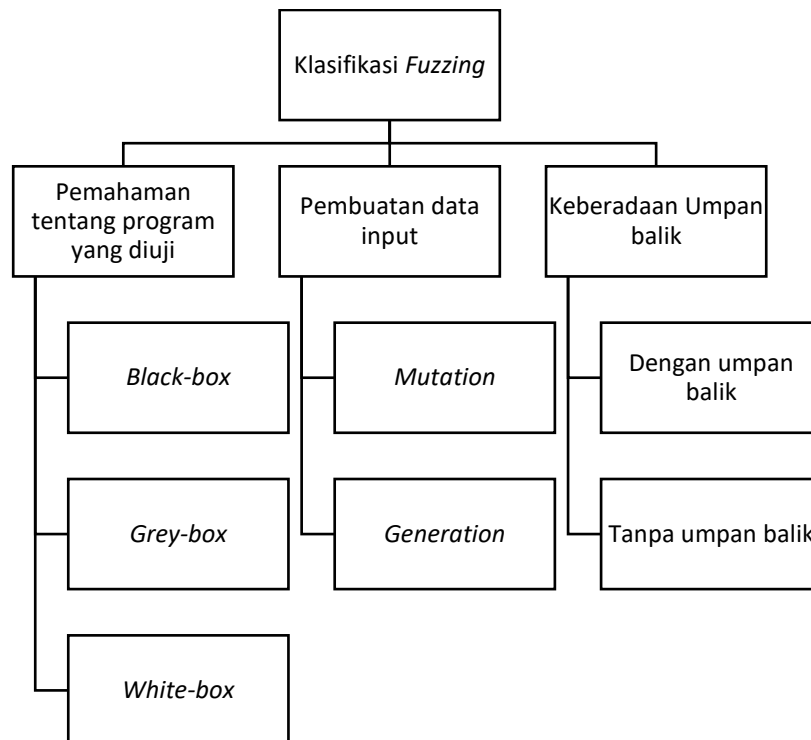
Gambar II.2 Iterasi proses *fuzzing*

Sejauh ini, sebuah *fuzzer file* seperti AFL tidak bisa digunakan untuk melakukan *fuzzing* terhadap sebuah aplikasi web. Begitu juga sebaliknya, sebuah *fuzzer web* seperti wfuzz tidak bisa digunakan sebagai *fuzzer file*. Hal tersebut mengakibatkan terdapat beragam *fuzzer* sesuai program yang diuji seperti *fuzzer file*, *fuzzer web*, *fuzzer Application Programming Interface (API)*, *fuzzer kernel* Sistem Operasi dan lain – lain. Gambar II.3 menunjukkan beragam *fuzzer* berdasarkan program yang diuji. Oleh karena itu, pemilihan perangkat *fuzzer* harus tepat dengan menjadikan program yang diuji sebagai salah satu acuan.



Gambar II.3 *Fuzzer* berdasarkan program yang diuji

Fuzzing bisa diklasifikasikan berdasarkan tiga hal yaitu metode pengujian, pembuatan data input dan keberadaan umpan balik (*feedback*) [10]. Gambar II.4 mengilustrasikan klasifikasi *fuzzing*. Metode pengujian bisa menggunakan teknik *black-box*, *grey-box* atau *white-box*. Perbedaan dari ketiganya adalah tingkat pemahaman yang diperlukan tentang program yang diuji. *Black-box fuzzing* tidak memerlukan informasi apapun dari program yang diuji. Sedangkan *white-box fuzzing* memerlukan informasi detail dari program yang diuji meliputi kode sumber, spesifikasi desain dan informasi *runtime*. Di lain sisi, *grey-box fuzzing* berada diantara *black-box* dan *white-box fuzzing* yang memerlukan beberapa informasi dari program yang diuji namun tidak terlalu detail seperti pada *white-box fuzzing*. Berdasarkan pembuatan data input, *fuzzing* dapat dibedakan menjadi *mutation-based* dan *generation-based*. *Mutation-based* melakukan transformasi acak terhadap *seed* yang disiapkan sedangkan *generation-based* menghasilkan *seed* sesuai spesifikasi format input formal. Berdasarkan umpan balik yang diberikan oleh *fuzzer*, *fuzzing* dapat dibedakan menjadi *fuzzer* dengan umpan balik atau tanpa umpan balik. Fungsi dari umpan balik ini adalah memberikan informasi yang dapat digunakan untuk menentukan data input pada proses iterasi *fuzzing* berikutnya.



Gambar II.4 Klasifikasi *fuzzing*

II.3 American Fuzzy Lop (AFL)

AFL merupakan salah satu *grey-box fuzzer* yang menggunakan teknik mutasi untuk menciptakan data input (*seed*) dan memiliki umpan balik (*feedback*) dari program yang diuji. AFL mendukung teknik mutasi deterministik dan non deterministik (acak). Perbedaan hasil mutasi dari keduanya dapat dilihat pada tabel II.1. Keberadaan umpan balik tersebut dimanfaatkan untuk menentukan *seed* yang akan digunakan pada proses iterasi berikutnya sehingga didapatkan hasil yang lebih baik [10].

Tabel II.1 Perbandingan hasil mutasi

Mutasi Deterministik	Mutasi Non Deterministik
band	band
and	5A@8
"and	66A
rand	????????????????K
{and	""

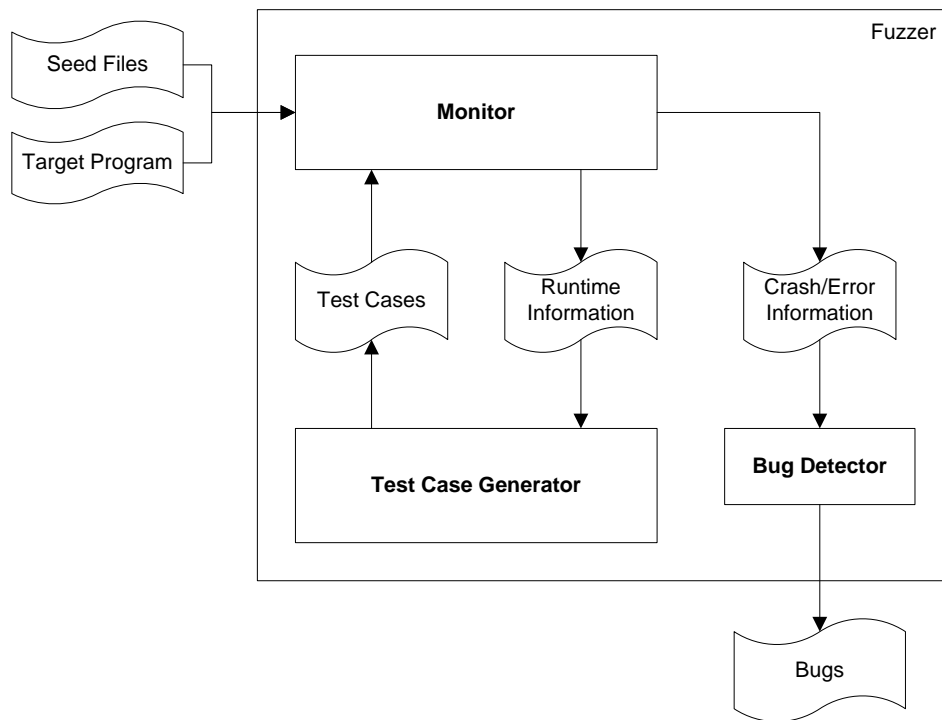
AFL dapat digunakan pada perangkat lunak yang tersedia kode sumbernya atau tersedia dalam bentuk file biner. Kode sumber akan dikompilasi menggunakan *compiler* bawaan AFL untuk memasukkan kode instrumen yang berfungsi memonitor perilaku abnormal dari program yang sedang diuji seperti yang ditunjukkan gambar II.5. Apabila program target tidak memiliki kode sumber, AFL memiliki mode *qemu*.

```
bayu@x230:~/Documents/Tesis/Fuzzgoat> afl-gcc fuzz.c -o fuzz
afl-cc 2.52b by <lcamtuf@google.com>
afl-as 2.52b by <lcamtuf@google.com>
[+] Instrumented 10 locations (64-bit, non-hardened mode, ratio 100%).
bayu@x230:~/Documents/Tesis/Fuzzgoat>
```

Gambar II.5 Compile program yang diuji menggunakan *compiler* bawaan AFL

Dalam setiap proses *fuzzing* menggunakan AFL seperti yang diperlihatkan gambar II.6 [2], diperlukan dua masukan berupa *file seed* atau spesifikasi dan program yang diuji. *Fuzzer* terdiri dari beberapa komponen seperti monitor, pembuat kasus uji, dan pendeteksi *bug*. Monitor umumnya terdapat pada *white-box* atau *grey-box fuzzer*. Monitor memanfaatkan teknik seperti kode instrumentasi, analisis *taint* atau

teknik lainnya untuk mendapatkan informasi yang berguna dari program yang diuji. Oleh karena itu, monitor tidak diperlukan di *black-box fuzzer*.



Gambar II.6 Proses *fuzzing* pada AFL

Pembuat kasus uji digunakan untuk menghasilkan kasus uji berdasarkan *file seed* yang ditentukan. Terdapat dua metode utama untuk membuat kasus uji yaitu *mutation-based* dan *grammar-based*. Metode pertama dilakukan dengan cara memodifikasi *file seed* yang valid secara acak atau menggunakan strategi mutasi yang telah ditetapkan yang nantinya dapat disesuaikan dengan informasi yang dikumpulkan dari program yang diuji selama proses *fuzzing*. Sedangkan, metode kedua tidak memerlukan *file seed*. Kasus uji dihasilkan dari spesifikasi yang telah ditetapkan. Pendeteksi *bug* membantu penguji untuk menemukan *bug* yang potensial dari program yang diuji sekaligus memastikan tidak ada duplikasi *bug*. Ketika program yang diuji mengalami kegagalan fungsi atau mengirimkan informasi kesalahan, pendeteksi *bug* akan mengumpulkan dan menganalisis informasi terkait untuk menentukan ada atau tidaknya *bug*.

Proses *fuzzing* dengan AFL dimulai dengan menyiapkan *file seed* karena AFL mengadopsi teknik berbasis mutasi. Selanjutnya AFL dijalankan dengan perintah

afl-fuzz dengan menentukan direktori masukan berisi *file seed*, program yang diuji dan direktori keluaran yang nantinya akan digunakan AFL untuk menyimpan hasil dari proses *fuzzing*. Selama proses pengujian, komponen monitor dari AFL akan mengumpulkan informasi *runtime* yang spesifik dengan memanfaatkan instrumen. Informasi berharga yang didapatkan akan diteruskan ke komponen pembuat kasus uji. Informasi tersebut digunakan untuk membantu proses pembuatan kasus uji. Kasus uji yang baru dihasilkan akan dikembalikan ke komponen monitor sebagai masukan untuk program yang diuji. Proses ini berlangsung secara terus-menerus sampai penguji menghentikan AFL atau batas waktu yang diberikan telah tercapai.

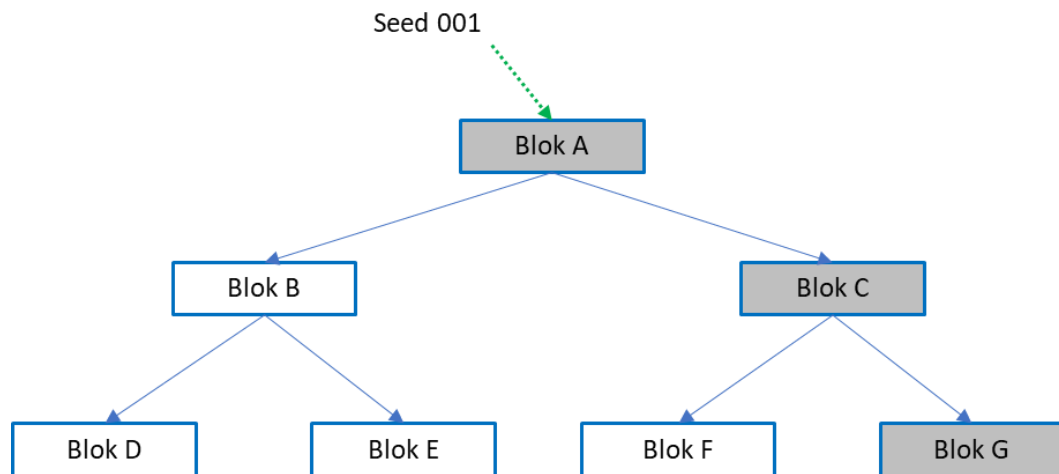
Pada saat proses *fuzzing* berjalan, AFL menampilkan sebuah *dashboard* yang berisi informasi penting seperti waktu eksekusi, jumlah *crash* yang unik, kecepatan eksekusi dan lain – lain. Jumlah *crash* yang unik merupakan nilai yang harus diperhatikan karena hal tersebut menunjukkan indikasi dari kerentanan yang mungkin dapat dieksploitasi atau menyebabkan kegagalan fungsi. Keluaran dari AFL terdiri dari beberapa hal seperti yang dijelaskan oleh tabel II.2.

Tabel II.2 Keluaran AFL

No	Nama	Keterangan
1	<i>crashes</i>	Direktori yang berisi <i>seed</i> yang menyebabkan program <i>crash</i>
2	<i>hangs</i>	Direktori yang berisi <i>seed</i> yang menyebabkan program <i>hang</i>
3	<i>queue</i>	Direktori yang berisi kumpulan <i>seed</i> yang menarik
4	<i>fuzz_bitmap</i>	File yang menyimpan data <i>path coverage</i>
5	<i>fuzzer_stats</i>	File yang menyimpan data ringkasan statistik proses <i>fuzzing</i>
6	<i>plot_data</i>	File yang menyimpan data histori statistic proses <i>fuzzing</i>

Pada mode paralel, AFL terdiri dari 1 *master node* dan 1 atau lebih *slave node*. Untuk mencegah duplikasi tugas, AFL melakukan sinkronisasi *seed* dan *path coverage*. *Path coverage* adalah jalur transisi antar blok ketika program dieksekusi. Informasi tersebut disimpan oleh AFL pada sebuah file berukuran 64KB yang disebut bitmap. Perubahan ukuran file bitmap tersebut berpengaruh terhadap kecepatan eksekusi. Penambahan ukurannya mengakibatkan penurunan kecepatan eksekusi AFL [11]. Gambar II.7 menunjukkan gambaran umum *path coverage*

yang mengilustrasikan jalur transisi dari blok A ke G ketika program dieksekusi menggunakan *seed* tertentu. Apabila *path coverage* tersebut belum pernah dijelajahi, maka *seed* yang digunakan akan dijadikan *seed* yang menarik karena menghasilkan *path coverage* baru. *Path coverage* berkaitan erat dengan *crash*. *Crash* yang terjadi pada sebuah *path coverage* baru akan dikategorikan sebagai *crash* yang unik. AFL membedakan *crash* dengan *crash* yang unik untuk mencegah duplikasi *bug*.



Gambar II.7 Gambaran umum *path coverage*

II.4 Container

Container merupakan salah satu teknik virtualisasi pada sistem komputer sama halnya dengan *virtual machine (VM)*. Namun, *container* melakukan pendekatan yang berbeda. Virtualisasi pada *VM* dilakukan di level *hardware* sedangkan pada *container* dilakukan di level Sistem Operasi yang dijalankan pada satu induk kernel Linux (*host*). Perbedaan tersebut membuat *container* tidak memerlukan perangkat keras virtual dan Sistem Operasi *guest* seperti yang ditunjukkan gambar II.8 [12]. Hal tersebut menyebabkan kebutuhan perangkat keras *container* jauh lebih ringan dibanding *VM* dan batasan kinerja perangkat keras pada *container* sepenuhnya bergantung pada perangkat keras *host*. Selain itu, waktu yang dibutuhkan untuk *deploy* aplikasi menjadi lebih cepat dengan memanfaatkan *container* karena tidak perlu menginstall Sistem Operasi *guest*.



Gambar II.8 Perbandingan arsitektur antara (a) *container* dan (b) *VM*

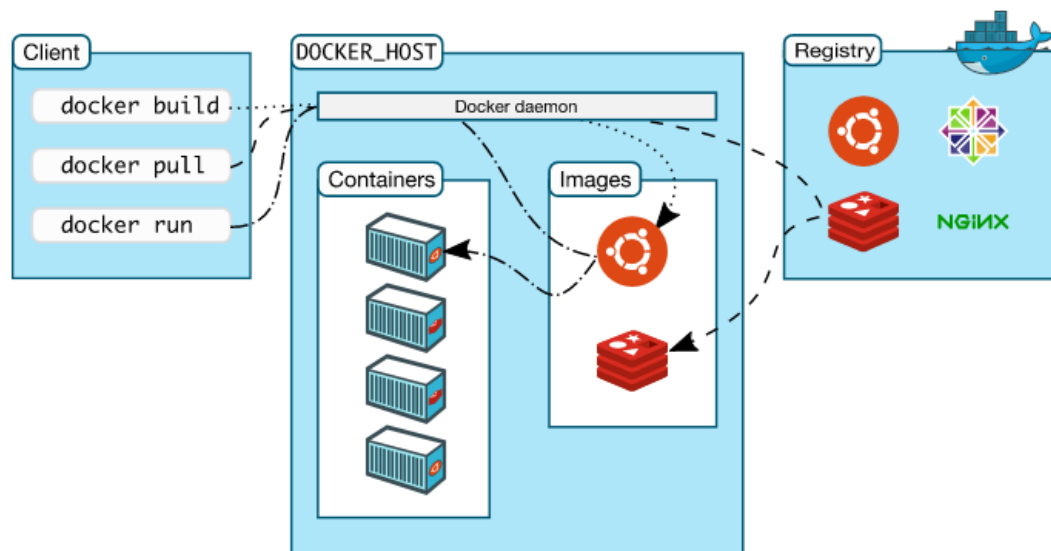
Dengan memanfaatkan *container*, aplikasi akan dikemas dalam sebuah wadah (*container*) beserta semua dependensi yang diperlukan agar aplikasi dapat berjalan dengan baik. Salah satu *platform* yang memanfaatkan teknologi *container* adalah docker. Docker adalah *platform* yang paling populer di dunia teknologi *container* saat ini [13]. Seiring dengan popularitas tersebut, orang cenderung mengartikan docker sebagai *container* itu sendiri. Docker pertama kali dikembangkan oleh Solomon Hykes dan Sebastien Pahl yang diluncurkan pada 2011. Hykes memulai proyek Docker sebagai proyek internal di dotCloud, sebuah perusahaan penyedia layanan *Platform as a Service (PaaS)*. Docker mulai diperkenalkan ke publik pada even PyCon di Santa Clara pada tahun 2013. Pada tahun yang sama, docker dirilis sebagai proyek *open source* yang memudahkan pengembang dan administrator sistem dalam membangun, mengirim dan menjalankan aplikasi terdistribusi [14].

Docker terdiri dari tiga komponen yaitu docker *image*, docker *registry* dan docker *container*. Docker *image* adalah sebuah *template read-only* dari paket perangkat lunak yang siap dijalankan menjadi docker *container*. Sebagai contoh, sebuah docker *image* bisa berisi Sistem Operasi Linux, web server Nginx dan aplikasi berbasis web yang dibuat oleh pengembang. Sebuah docker *image* yang sudah dibuat dapat dimanfaatkan oleh pengguna lain yang membutuhkan tanpa harus membuat ulang docker *image* tersebut. Untuk mendukung hal tersebut, dibutuhkan sebuah docker *registry*. Docker *registry* berfungsi sebagai tempat penyimpanan docker *image*. Peran dari docker *registry* identik dengan server lumbung berkas pada sistem Linux. Sebuah docker *image* bisa diunggah atau diunduh dengan

memanfaatkan docker *registry*. Docker *registry* bisa bersifat publik atau privat. Contoh docker *registry* adalah:

1. Docker Hub (<https://hub.docker.com>)
2. Google Container Registry (<https://cloud.google.com/container-registry>)
3. AWS Elastic Container Registry (<https://aws.amazon.com/id/ecr>)

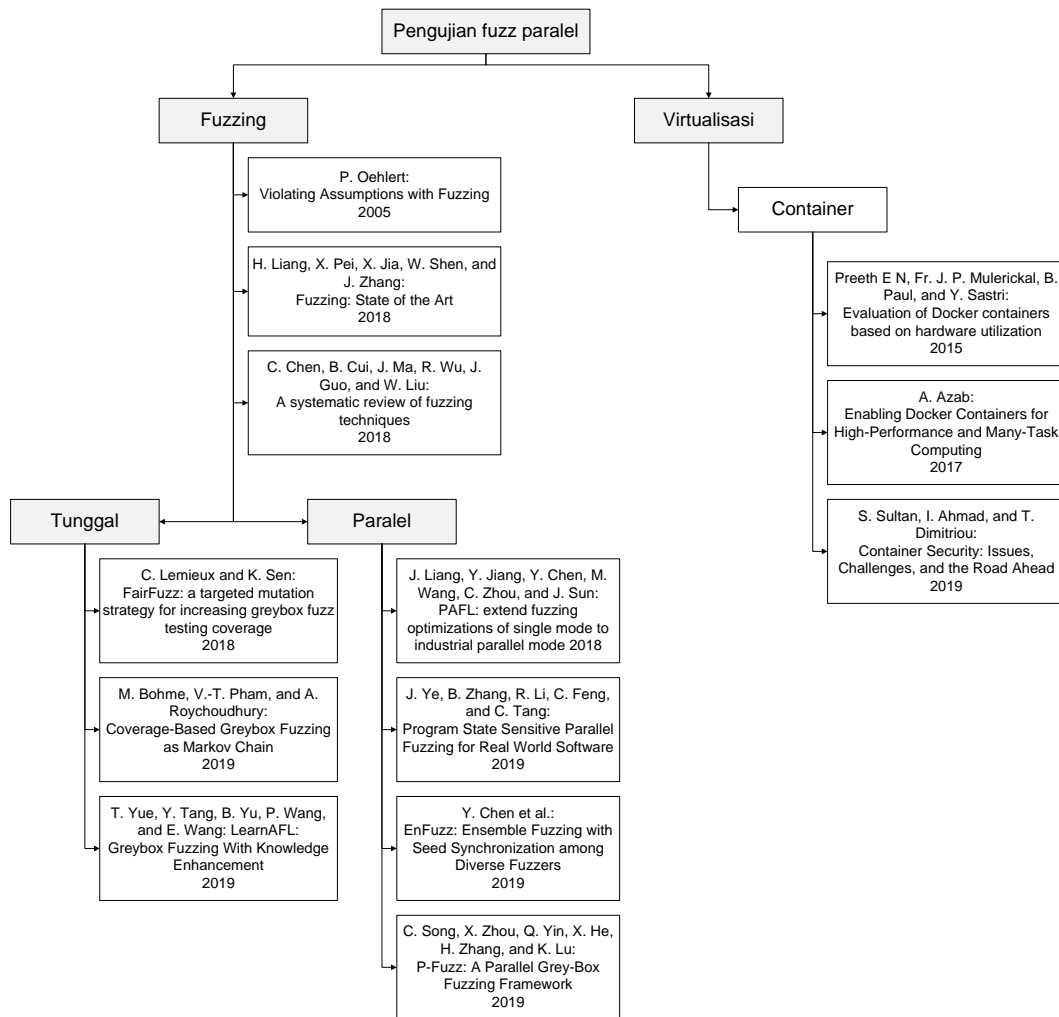
Docker *container* adalah sebuah *environment* yang terisolasi yang dihasilkan dari *docker image*. Sebuah *container* dapat dijalankan, dihentikan, dipindah atau dihapus. Gambar II.9 menunjukkan arsitektur *docker* yang menggambarkan relasi antara *docker image*, *docker registry* dan *docker container*. Dengan memanfaatkan *docker*, sebuah *image* yang berisi *fuzzer* nantinya bisa dijalankan sebagai beberapa *container* yang akan digunakan untuk melakukan proses *fuzzing* secara paralel.



Gambar II.9 Arsitektur docker

II.5 Peta Literatur

Untuk mendukung penelitian ini, diperlukan literatur yang berhubungan dengan topik penelitian. Sumber literatur didapatkan dari berbagai referensi seperti jurnal, konferensi, buku, artikel dan sumber lainnya yang masih terkait dengan penelitian. Sumber literatur yang telah dicari kemudian dipelajari. Guna menggambarkan hubungan penelitian dengan literatur – literatur yang telah dipelajari maka dibuat peta literatur seperti pada gambar II.10. Dengan adanya peta literatur, relasi antar literatur dapat dipaparkan secara visual berupa grafik yang lebih mudah dipahami.



Gambar II.10 Peta Literatur

Beberapa penelitian telah dilakukan untuk meningkatkan kapabilitas AFL. Sebagian penelitian tersebut dilakukan pada mode tunggal [4], [15], [16]. FairFuzz [4] memperbaiki algoritma mutasi AFL sehingga pengujian bisa menjangkau lebih banyak bagian percabangan pada program. AFLfast [15] mengimplementasikan model *Markov Chain* untuk mengatur mekanisme pemilihan *seed*. LearnAFL [16] menerapkan sebuah format informasi khusus yang dimanfaatkan untuk melakukan mutasi terhadap *seed*. PAFL [5], [17], enFuzz [18] dan P-Fuzz [19] adalah penelitian yang dilakukan pada mode paralel. Di samping itu, AFL sendiri sudah mendukung proses *fuzzing* secara paralel. PAFL [5] mengimplementasikan *fuzzing* paralel dengan cara membuat struktur data tambahan untuk menyimpan informasi yang akan disinkronisasikan antar *fuzzer* dan membagi tugas antar *fuzzer* berdasarkan *bitmap* keluaran dari AFL. PAFL [17] melakukan proses *fuzzing* secara

paralel dengan cara membagi program target menjadi beberapa bagian dimana masing – masing bagian akan diproses oleh beberapa *fuzzer*. enFuzz memanfaatkan sinkronisasi *seed* antar *fuzzer* yang berbeda untuk melakukan *fuzzing* paralel. P-Fuzz [19] memanfaatkan *database* untuk berbagi status proses *fuzzing*. Setiap *fuzzer* akan mendapatkan tugas berdasarkan informasi dari *database* dan memberikan informasi status proses *fuzzing* untuk disimpan ke dalam *database*. Mode paralel pada AFL melakukan sinkronisasi informasi *seed* dan *path coverage* antara *master node* dan *slave node*. Selain itu, terdapat perbedaan penerapan teknik mutasi antara kedua *node*. Tantangan utama *fuzzing* secara paralel adalah sinkronisasi informasi dan pembagian tugas antar *node* sehingga sumber daya komputasi bisa digunakan secara efisien atau tidak terjadi duplikasi tugas antar *node* [5].

Pada penelitian yang akan dilakukan, terdapat *master node* dan *slave node* dimana *master node* akan bertugas mengelola *slave node* sedangkan *slave node* akan fokus melakukan proses *fuzzing*. Pada *master node* terdapat sebuah program untuk mengontrol jalannya proses *fuzzing* dan sebuah program lagi untuk mengelola file *bitmap* yang berisi informasi *path coverage* untuk menghindari duplikasi *bug*. *Slave node* terdiri dari beberapa *container* yang didalamnya terdapat *fuzzer* dan sebuah program untuk mengelola *seed* agar tidak terjadi duplikasi. Untuk menyimpan data keluaran dari proses *fuzzing* dan mendukung proses sinkronisasi *seed*, penelitian ini memanfaatkan sebuah *database*.

Bab III Perancangan Sistem

Bagian ini menjelaskan secara lengkap fase *engineering development* yang terdiri dari 3 (tiga) tahap yaitu *advanced development*, *engineering design* dan *integration and evaluation*. Pada tahapan *advanced development*, dilakukan analisis kebutuhan sistem yang meliputi analisis kebutuhan pengguna, analisis kebutuhan fungsional dan analisis kebutuhan non fungsional. Pada tahapan *engineering design*, ditentukan arsitektur dan desain sistem yang mengacu pada hasil analisis kebutuhan sistem. Desain sistem akan dipetakan dengan menggunakan UML dalam bentuk *use case diagram*, *activity diagram* dan *class diagram*. Pada tahapan *integration and evaluation*, desain sistem ditransformasikan menjadi sebuah prototipe yang nantinya akan diuji dan dievaluasi hasil pengujiannya.

III.1 Analisis Kebutuhan Sistem

Kebutuhan sistem adalah kondisi, kriteria, syarat atau kemampuan yang harus dimiliki sistem untuk memenuhi ekspektasi pengguna. Dalam penelitian ini, sistem yang dibutuhkan adalah sistem yang mampu menjalankan proses *fuzzing* secara paralel dengan meniadakan duplikasi tugas antar *fuzzer*. Kondisi duplikasi tugas terjadi apabila *fuzzer* memproses *seed* yang sama. Selain itu, sistem harus dapat membedakan peran antara *master node* dan *slave node*.

III.1.1 Analisis Kebutuhan Pengguna

Analisis kebutuhan pengguna adalah proses mengidentifikasi aktor - aktor yang terlibat dalam penggunaan sistem. Pada sistem *fuzzing* paralel, terdapat 3 (tiga) aktor yang dibutuhkan untuk menjalankan sistem, yaitu:

1. Pengguna

Pengguna merupakan aktor utama dalam sistem yang akan diimplementasikan. Pengguna berperan dalam menentukan *seed*, menentukan program target, menjalankan dan menghentikan proses *fuzzing*.

2. *Fuzzer*

Fuzzer merupakan aktor yang berperan dalam menjalankan dan menghentikan proses *fuzzing*.

3. Agen Sinkronisasi

Terdapat 2 Agen sinkronisasi yaitu agen sinkronisasi *seed* dan agen sinkronisasi *path coverage*. Agen merupakan aktor yang berperan mengelola keduanya. Agen memastikan bahwa *seed* yang dihasilkan tidak ada yang duplikat. Selain itu, agen mengelola proses sinkronisasi *path coverage* dari masing – masing *fuzzer*.

III.1.2 Analisis Kebutuhan Fungsional

Analisis kebutuhan fungsional adalah proses mengidentifikasi semua fungsi atau proses yang nantinya harus bisa dilakukan oleh sistem. Kebutuhan fungsional harus bisa menggambarkan layanan – layanan yang yang bisa diberikan oleh sistem kepada pengguna. Tabel III.1 menunjukkan kebutuhan fungsional dari sistem yang akan dibangun.

Tabel III.1 Kebutuhan fungsional

Stakeholders	ID	Deskripsi
Pengguna	USR-01	Pengguna menentukan <i>seed</i>
	USR-02	Pengguna menentukan program yang diuji
	USR-03	Pengguna menjalankan <i>container</i>
	USR-04	Pengguna menjalankan <i>fuzzer</i>
	USR-05	Pengguna membaca <i>file plot_data</i>
	USR-06	Pengguna menghentikan <i>fuzzer</i>
	USR-07	Pengguna menyimpan data <i>seed</i> ke dalam <i>database</i>
	USR-08	Pengguna menyimpan data <i>fuzzer_stats</i> ke dalam <i>database</i>
	USR-09	Pengguna melakukan proses pemilihan <i>seed</i> dari beberapa <i>seed</i> yang dihasilkan
	USR-10	Pengguna melakukan sinkronisasi <i>path coverage</i>
	USR-11	Pengguna menghentikan <i>container</i>
Fuzzer	FZR-01	Fuzzer menghasilkan <i>seed</i>
	FZR-02	Fuzzer menghasilkan <i>file fuzzer_stats</i>
	FZR-03	Fuzzer menjalankan proses <i>fuzzing</i>
Agen sinkronisasi <i>seed</i>	ASD-01	Fuzzer melakukan sinkronisasi <i>seed</i>
Agen sinkronisasi <i>path coverage</i>	APC-01	Fuzzer melakukan sinkronisasi <i>path coverage</i>

III.1.3 Analisis Kebutuhan Non Fungsional

Analisis kebutuhan non fungsional adalah proses mengidentifikasi kebutuhan yang menitikberatkan pada perilaku sistem. Dalam penelitian ini, kebutuhan non fungsional didefinisikan sebagai kebutuhan keamanan dan kualitas berikut ini:

1. kebutuhan keamanan
 - a. Setiap *fuzzer* berjalan pada lingkungan yang terisolasi;
 - b. Setiap *container* hanya bisa mengakses docker *volume* masing – masing.
2. kebutuhan kualitas
 - a. sistem berjalan dengan memanfaatkan beberapa *fuzzer*;
 - b. kecepatan menemukan kerentanan lebih baik dibanding sistem yang sudah ada.

III.1.4 Analisis Kebutuhan Perangkat Lunak

Analisis kebutuhan perangkat lunak adalah proses mengidentifikasi perangkat lunak yang dibutuhkan untuk mendukung pengembangan sistem yang baru. Tabel III.2 menunjukkan daftar perangkat lunak yang diperlukan untuk mendukung penelitian. Selain itu, untuk mendukung proses pengujian diperlukan sebuah program yang dikembangkan menggunakan bahasa pemrograman C. Program yang diuji akan mengalami *crash* ketika menerima masukan tertentu.

Tabel III.2 Kebutuhan perangkat lunak

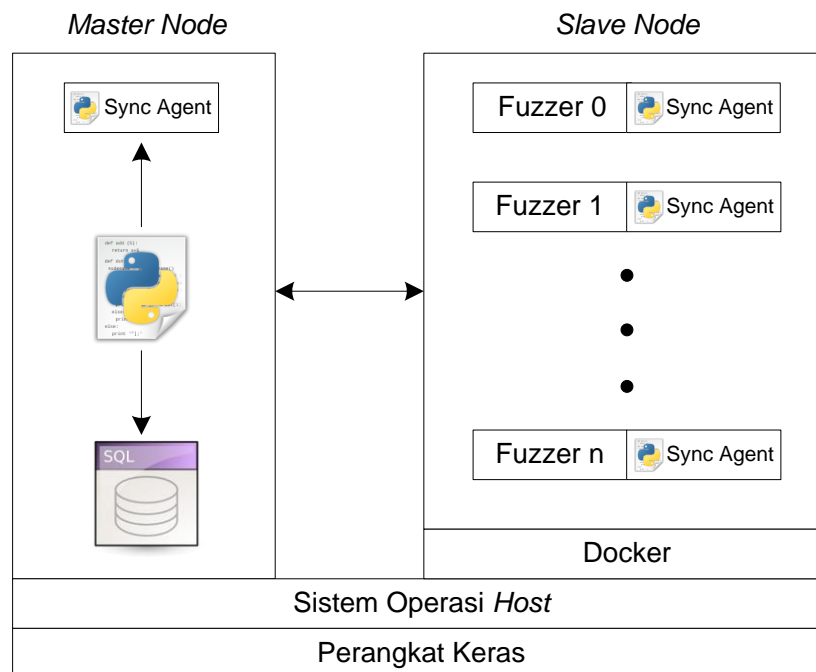
No	Nama	Keterangan	Versi
1	OpenSuse Leap	Sistem Operasi	15.1
2	Docker	<i>Container</i>	19.03
3	AFL	<i>Fuzzer</i>	2.52
4	SQLite	<i>Database</i>	3.28
5	SQLite Studio	Manajemen <i>database</i>	3.2.1
6	Python	Aplikasi master dan agen sinkronisasi	3.6.10

III.2 Arsitektur Sistem

Arsitektur sistem adalah sebuah kerangka kerja komprehensif yang mampu mendeskripsikan bentuk dan struktur sistem beserta komponen – komponen sistem sekaligus dapat menjelaskan bagaimana komponen – komponen tersebut saling bekerja sama membentuk satu kesatuan fungsi [20]. Dalam penelitian ini, terdapat beberapa komponen sistem yang dimanfaatkan untuk mendukung penelitian sebagai berikut:

1. sistem operasi berbasis GNU/Linux;
2. *fuzzer* AFL;
3. python;
4. docker;
5. *database* SQLite.

Gambar III.1 di bawah ini menunjukkan arsitektur sistem dari penelitian yang dilakukan. Aplikasi master, *database* dan agen sinkronisasi *path coverage* berada di *master node* sedangkan *fuzzer* dan agen sinkronisasi *seed* yang berada di dalam sebuah *container* terdapat di *slave node*. Berbagi data antar *master node* dan *slave node* akan memanfaatkan *docker volume*.



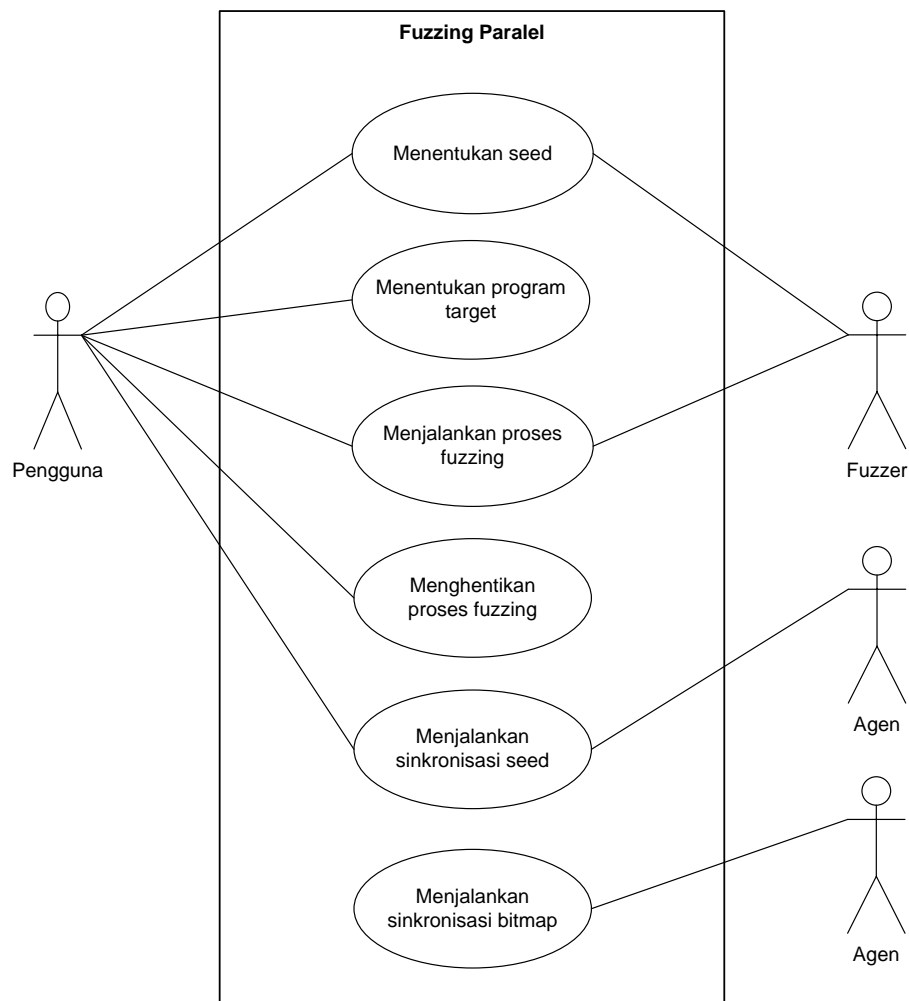
Gambar III.1 Arsitektur sistem

III.3 Desain Sistem

III.3.1 Use Case Diagram

Use case diagram adalah diagram yang digunakan untuk menggambarkan secara ringkas siapa yang menggunakan sistem dan apa saja yang bisa dilakukannya. Diagram ini menjelaskan hubungan antara aktor, *use case* dan sistem. Aktor merupakan orang, proses atau sistem lain yang akan berinteraksi dengan sistem yang akan dibuat. *Use case* adalah fungsionalitas yang disediakan oleh sistem yang

akan dibuat. Oleh karena itu, *use case diagram* fokus pada kebutuhan fungsional. Pada penelitian ini, terdapat 3 (tiga) aktor yaitu pengguna, *fuzzer* dan agen sinkronisasi. Pengguna merupakan aktor utama yang berperan dalam menentukan *seed*, menentukan target program, menjalankan/menghentikan proses *fuzzing* dan sinkronisasi *seed*. *Fuzzer* berperan untuk menjalankan proses *fuzzing* dan menentukan *seed* yang akan digunakan oleh *fuzzer* lainnya. Agen berperan dalam proses sinkronisasi *seed* antar *fuzzer*. Gambar III.2 menunjukkan *use case diagram* dari sistem yang akan dibuat.

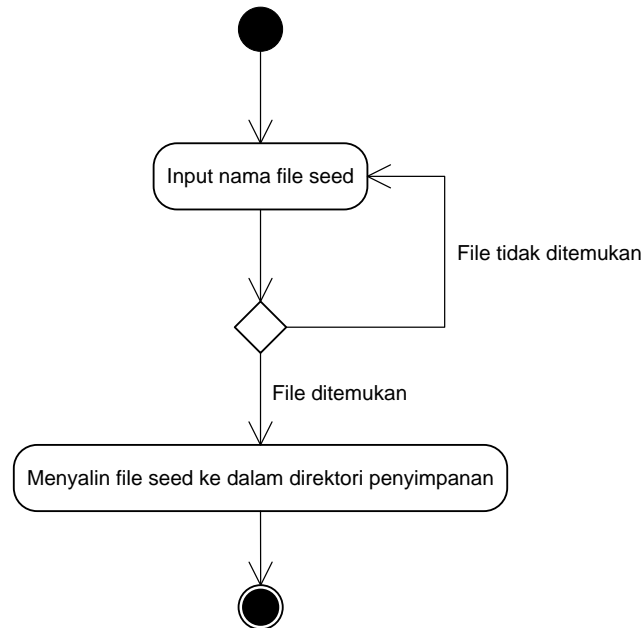


Gambar III.2 *Use case diagram*

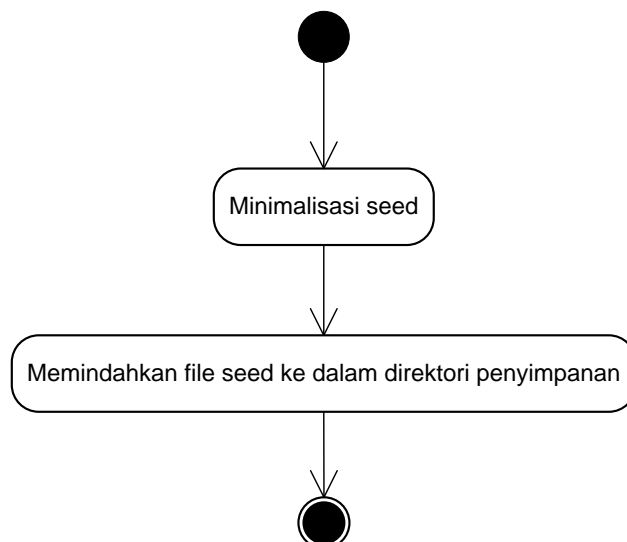
III.3.2 Activity Diagram

Activity diagram adalah diagram yang menggambarkan alur kerja atau aktivitas dari sebuah sistem. Gambar III.3 dan gambar III.4 menunjukkan *activity diagram* untuk *use case* menentukan *seed*. Pada gambar III.3, sistem menerima nama *file seed* dari

pengguna yang akan dilakukan pengecekan terlebih dahulu terkait ketersediaannya sebelum *file seed* disalin ke dalam direktori penyimpanan yang telah ditentukan. *File seed* ini merupakan salah satu masukan yang diperlukan sistem. Pada gambar III.4, sistem memilih satu *seed* dari kumpulan *seed* yang dihasilkan oleh proses *fuzzing* sebelumnya dengan memanfaatkan *fuzzer* yang terdapat di *master node*. *Seed* yang dipilih akan digunakan oleh *fuzzer* lainnya sebagai *seed* awal.

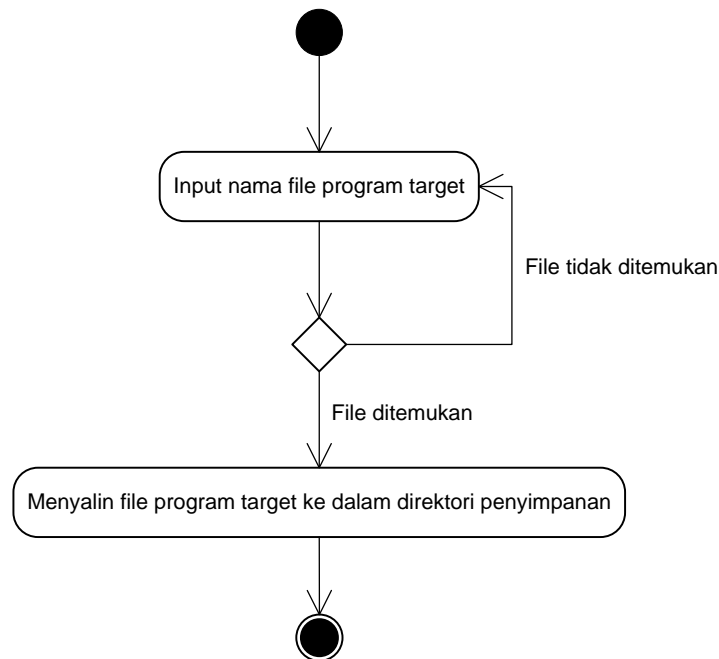


Gambar III.3 *Activity diagram* menentukan *seed* untuk *fuzzer* pertama



Gambar III.4 *Activity diagram* menentukan *seed*

Activity diagram untuk *use case* menentukan program target ditunjukkan oleh gambar III.5. Pada diagram ini, sistem menerima nama *file* program target dari pengguna yang akan dilakukan pengecekan terlebih dahulu terkait ketersediaannya sebelum *file* program target disalin ke dalam direktori penyimpanan yang telah ditentukan di setiap *fuzzer*. *File* program target ini merupakan salah satu masukan yang diperlukan sistem.

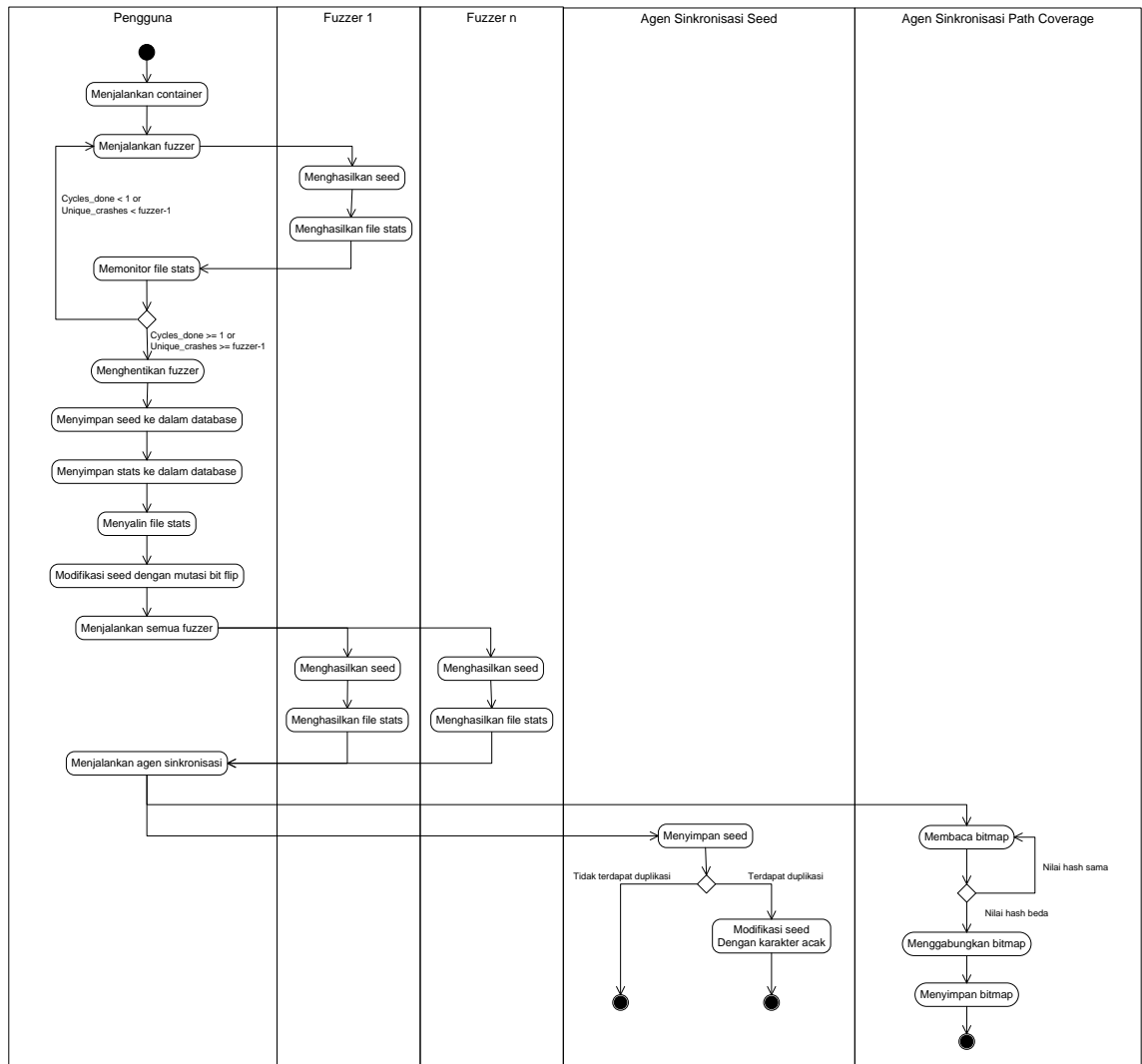


Gambar III.5 *Activity Diagram* menentukan program target

Proses *fuzzing* ditunjukkan oleh gambar III.6. Pada tahap awal, sistem menjalankan *container* dan *fuzzer*. Dalam hal ini, sistem hanya menjalankan satu *fuzzer*. Ketika proses *fuzzing* berjalan, *fuzzer* menghasilkan keluaran yang disimpan pada direktori yang telah ditentukan. *File fuzzer_stats* dan kumpulan *file seed* merupakan sebagian keluaran yang dihasilkan. *File fuzzer_stats* berisi data statistik proses *fuzzing*. Terdapat beberapa data yang tersimpan di *file* tersebut, diantaranya adalah *cycles_done* yang menunjukkan seberapa banyak iterasi proses *fuzzing* telah dilakukan dan *unique_crashes* yang menunjukkan jumlah kerentanan yang telah ditemukan. Idealnya, *fuzzer* harus menyelesaikan iterasi minimal 1 (satu) kali sebelum proses *fuzzing* dihentikan oleh pengguna. Nilai tersebut akan digunakan oleh sistem untuk menghentikan proses *fuzzing* awal yang dilakukan oleh *fuzzer* pertama ketika *cycles_done* atau *unique_crashes* bernilai lebih dari atau sama dengan satu.

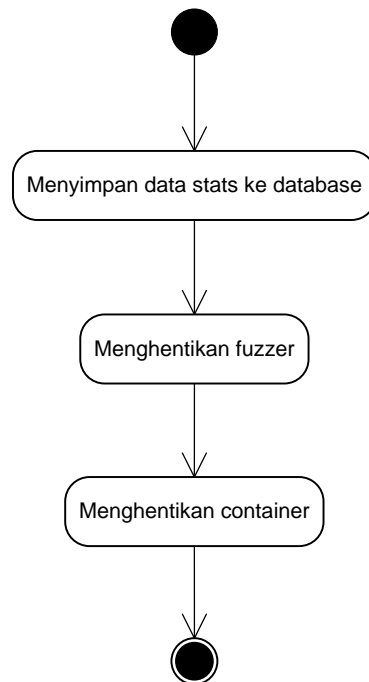
Setelah menghentikan proses *fuzzing*, sistem akan menyimpan data statistik dan *seed* yang dihasilkan. Selanjutnya sistem akan mengganti nama *file fuzzer_stats* karena data statistik yang tersimpan di *file* tersebut akan hilang ketika *fuzzer* dijalankan ulang. Langkah selanjutnya sistem akan memilih *seed* yang dihasilkan oleh proses *fuzzing* sebelumnya untuk dijadikan *seed* awal pada *fuzzer* lainnya. Dikarenakan semua *fuzzer* sudah memiliki *seed* awal dan program yang diuji, maka langkah selanjutnya adalah menjalankan ulang *fuzzer* yang sudah berjalan sebelumnya dan menjalankan *fuzzer* lainnya sehingga proses *fuzzing* bisa berjalan secara paralel dengan memanfaatkan beberapa *fuzzer*.

Proses sinkronisasi dijalankan setelah proses *fuzzing* berjalan paralel. Terdapat 2 agen yang bertugas melakukan sinkronisasi *seed* dan *path coverage*. Sinkronisasi *seed* memanfaatkan mutasi acak untuk mengubah nilai *seed* yang duplikat. Sinkronisasi *path coverage* memanfaatkan nilai hash MD5 untuk membandingkan file *bitmap* yang berisi data *path coverage*. Apabila terdapat perbedaan data *path coverage*, maka dilakukan penggabungan data *path coverage* dengan operator logika AND (^).



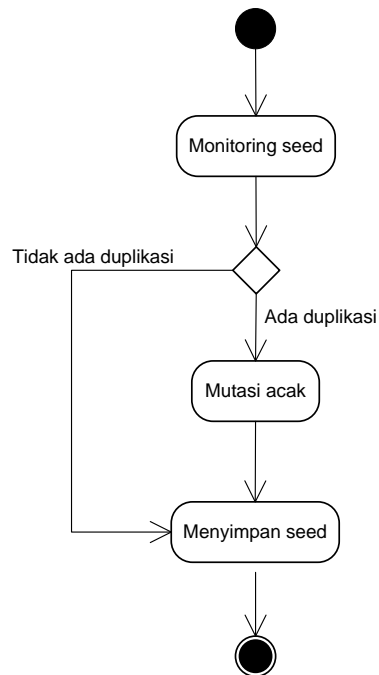
Gambar III.6 Activity diagram menjalankan proses fuzzing

Activity diagram untuk use case menghentikan proses fuzzing ditunjukkan oleh Gambar III.7. Ketika proses fuzzing dihentikan oleh pengguna, sistem akan menyimpan data statistik hasil dari proses fuzzing yang bersumber dari file *fuzzer_stats* kedalam *database* kemudian sistem akan menghentikan *fuzzer* dan *container*.



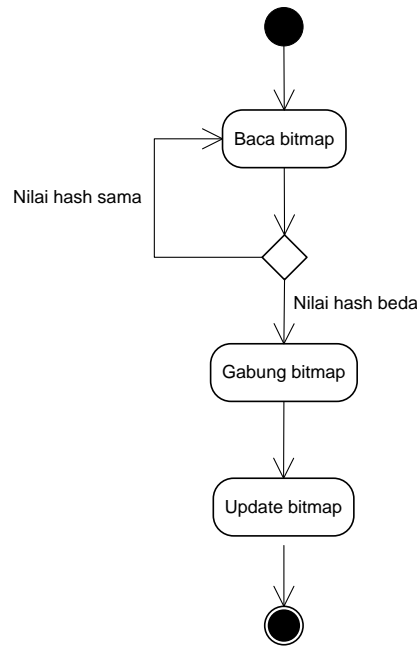
Gambar III.7 *Activity diagram* menghentikan proses *fuzzing*

Proses sinkronisasi *seed* ditunjukkan oleh gambar III.8. Sinkronisasi *seed* dilakukan untuk memastikan tidak ada duplikasi *seed*. Agen akan memonitor *seed* yang dihasilkan oleh AFL. Apabila terjadi duplikasi, dilakukan mutasi acak untuk memodifikasi *seed*.



Gambar III.8 *Activity Diagram* untuk sinkronisasi *seed*

Proses sinkronisasi *path coverage* ditunjukkan oleh gambar III.9. Sinkronisasi *path coverage* dilakukan untuk menyamakan data *path coverage* semua *fuzzer*. Agen akan memonitor file *bitmap* yang dihasilkan oleh AFL. Apabila isi file *bitmap* berbeda, maka akan dilakukan penggabungan data *path coverage* dari masing – masing *fuzzer*.



Gambar III.9 Activity Diagram untuk sinkronisasi *path coverage*

III.3.3 Desain Database

Sebuah *database* diperlukan untuk menyimpan data *seed* dan hasil statistik proses *fuzzing*. *Database* yang digunakan adalah SQLite. Data *seed* bersumber dari file *seed* yang terdapat di direktori *queue* sedangkan hasil statistik bersumber dari file *fuzzer_stats* yang dihasilkan selama proses *fuzzing*. Tabel *seed* digunakan untuk sinkronisasi *seed*. Tabel III.3 dan III.4 menunjukkan tabel *seed* dan *stats*.

Tabel III.3 Tabel *seed*

No	Kolom	Tipe Data
1	id	Integer
2	file_name	Time
3	value	Integer
4	hex_value	Decimal
5	fuzzer	Integer
6	status	Integer

Tabel III.4 Tabel stats

No	Kolom	Tipe Data
1	id	Integer
2	run_time	Time
3	cycles_done	Integer
4	execs_per_sec	Decimal
5	paths_total	Integer
6	paths_found	Integer
7	cur_path	Integer
8	uniq_crashes	Integer
9	unique_hangs	Integer
10	command_line	Varchar
11	fuzzer	Varchar

Bab IV Implementasi dan Pengujian

Tahap implementasi adalah tahap penerapan sistem dari hasil desain sistem yang sudah ditentukan sebelumnya. Setelah tahap implementasi adalah tahap pengujian yang akan menguji hasil implementasi. Dalam penelitian ini, tahap pengujian terdiri dari pengujian fungsional dan pengujian kualitas. Implementasi dan pengujian dilakukan dengan memanfaatkan komputer dengan spesifikasi sesuai tabel IV.1.

Tabel IV.1 Spesifikasi komputer

No	Komponen	Ukuran/Kapasitas
1	Prosesor	Core i7 @2,9GHz
2	Memori (RAM)	16 GB
3	SSD	500 GB

IV.1 Implementasi Sistem

Proses implementasi sistem pada penelitian ini terbagi menjadi 3 bagian yaitu pengembangan aplikasi master, pengembangan agen sinkronisasi *seed* dan agen sinkronisasi *path coverage*. Ketiganya akan dikembangkan menggunakan bahasa pemrograman Python. Proses instalasi kebutuhan perangkat lunak akan dijelaskan secara detail pada bagian lampiran.

IV.1.1 Implementasi Aplikasi Master

Aplikasi master merupakan komponen utama dari sistem yang akan dikembangkan. Aplikasi ini mengatur proses berjalannya *fuzzing* secara paralel. Pada aplikasi master terdapat 4 *class* sebagai berikut:

1. *class* Masukan

class ini memiliki 1 buah *method* yang berfungsi untuk menyalin file *seed* dan program yang diuji ke dalam direktori yang telah ditentukan;

2. *class* Container

class ini memiliki 2 buah *method*, yaitu:

- jalan
berfungsi untuk menjalankan *container*;
- berhenti
berfungsi untuk menghentikan *container*;

3. *class Fuzzer.*

class ini memiliki beberapa *method*, yaitu:

- *jalan*
berfungsi untuk menjalankan *fuzzer*;
- *berhenti*
berfungsi untuk menghentikan *fuzzer*;
- *baca_plot_data*
berfungsi untuk membaca file *plot_data*. File tersebut berisi informasi *cycles_done* dan *unique_crashes* yang diperlukan untuk menghentikan *fuzzing* diawal proses dimana proses tersebut menghasilkan kumpulan *seed* yang akan dipilih sebagai *seed* awal untuk *fuzzer* lainnya;
- *rename_file_stats*
berfungsi untuk memodifikasi nama file *file_stats* yang berisi informasi penting terkait proses *fuzzing*;
- *simpan_stats*
Berfungsi untuk menyimpan statistik proses *fuzzing* kedalam *database*;

4. *class Mutator.*

class ini memiliki 2 buah *method*, yaitu:

- *bit_flip*
Berfungsi untuk memodifikasi *seed* dengan mutasi bit *flip*;
- *update_file_seed*
Berfungsi untuk menyimpan file *seed*.

IV.1.2 Implementasi Agen Sinkronisasi Seed

Agen sinkronisasi *seed* bertugas mengelola file *seed* sehingga tidak terjadi duplikasi. Agen ini akan berjalan sebagai *background process*. Pada aplikasi ini terdapat 1 *class*, yaitu:

1. *class AgenSync*

class ini memiliki 6 buah *method*, yaitu:

- *cek_duplikasi*

berfungsi untuk mengecek duplikasi file *seed* berdasarkan nilai hex. Nilai hex tersebut akan dibandingkan dengan nilai hex file *seed* yang sudah tersimpan di *database*;

- *baca_file_seed*
berfungsi untuk membaca file *seed*;
- *simpan_seed*
berfungsi untuk menyimpan nilai *seed* kedalam *database*;
- *update_seed*
berfungsi untuk memutakhirkan nilai *seed* di *database* yang duplikat;
- *generator_acak*
berfungsi untuk menentukan nilai *seed* baru bagi *seed* yang duplikat dengan memanfaatkan mutasi acak;
- *update_file_seed*
berfungsi untuk memodifikasi file *seed* yang duplikat.

IV.1.3 Implementasi Agen Sinkronisasi *Path Coverage*

Agen sinkronisasi *path coverage* bertugas mengelola *path coverage* sehingga *path coverage* masing – masing *fuzzer* memiliki nilai yang sama. Agen ini akan berjalan sebagai *background process*. Pada aplikasi ini terdapat 1 *class*, yaitu:

1. *class AgenSync*

- *generate_hash*
berfungsi untuk menghasilkan nilai *hash* dari file *seed*;
- *baca_bitmap*
berfungsi untuk membaca file *bitmap* yang berisi data *path_coverage*;
- *gabung_bitmap*
berfungsi untuk menggabungkan nilai *path_coverage* yang tersimpan didalam file *bitmap* dari masing – masing *fuzzer*;
- *update_bitmap*
berfungsi untuk memutakhirkan file *bitmap*.

IV.2 Pengujian

Pada bagian ini akan dilakukan pengujian terhadap sistem *fuzzing* paralel berdasarkan perancangan dan implementasi yang telah dipaparkan sebelumnya. Pengujian dikelompokkan menjadi 2 bagian, yaitu pengujian fungsional dan pengujian kualitas.

IV.2.1 Pengujian Fungsional

Pengujian fungsional mengukur gap antara fase perancangan dan implementasi sistem. Pengujian ini memastikan fungsionalitas sistem sudah berjalan sesuai rancangan yang telah ditentukan. Pada pengujian fungsional dibuat deskripsi pengujian fungsional yang mengacu pada Tabel III.1 terkait kebutuhan fungsional.

Tabel IV.2 Skenario pengujian fungsional

Stakeholders	ID	Skenario Pengujian	Hasil yang Diharapkan
Pengguna	USR-01	Pengguna menjalankan aplikasi master	Berhasil
	USR-02	Pengguna menentukan <i>seed</i>	Berhasil
	USR-03	Pengguna menentukan program yang diuji	Berhasil
	USR-04	Pengguna mengeksekusi aplikasi master untuk menjalankan <i>container</i>	Berhasil
	USR-05	Pengguna mengeksekusi aplikasi master untuk menjalankan <i>fuzzer</i>	Berhasil
	USR-06	Pengguna mengeksekusi aplikasi master untuk membaca <i>file plot_data</i>	Berhasil
	USR-07	Pengguna menghentikan <i>fuzzer</i>	Berhasil
	USR-08	Pengguna mengeksekusi aplikasi master untuk melakukan mutasi bit flip	Berhasil
	USR-09	Pengguna mengeksekusi aplikasi master untuk menyimpan data <i>seed</i> ke dalam <i>database</i>	Berhasil
	USR-10	Pengguna mengeksekusi aplikasi master untuk menyimpan data <i>fuzzer_stats</i> ke dalam <i>database</i>	Berhasil
	USR-11	Pengguna mengeksekusi aplikasi master untuk melakukan proses pemilihan <i>seed</i> dari beberapa <i>seed</i> yang dihasilkan	Berhasil
	USR-12	Pengguna mengeksekusi aplikasi master untuk menjalankan sinkronisasi	Berhasil
	USR-13	Pengguna menghentikan <i>container</i>	Berhasil

Stakeholders	ID	Skenario Pengujian	Hasil yang Diharapkan
Fuzzer	FZR-01	<i>Fuzzer</i> menghasilkan <i>seed</i>	Berhasil
	FZR-02	<i>Fuzzer</i> menghasilkan <i>file fuzzer_stats</i>	Berhasil
	FZR-03	<i>Fuzzer</i> menjalankan proses <i>fuzzing</i>	Berhasil
Agen sinkronisasi seed	ASD-01	Agen melakukan mutasi acak	Berhasil
	ASD-02	Agen memodifikasi file <i>seed</i>	Berhasil
Agen sinkronisasi <i>path coverage</i>	APC-01	Agen melakukan sinkronisasi file <i>bitmap</i>	Berhasil

Berdasarkan skenario pengujian fungsional seperti tabel IV.1, maka dilakukan pengujian untuk mengukur keberhasilan rancangan sistem yang sudah diimplementasikan dengan hasil pengujian fungsional seperti yang ditunjukkan tabel IV.3.


Tabel IV.3 Hasil pengujian fungsional

Kode	Hasil Pengujian	Keterangan
USR-01	Pengguna menjalankan aplikasi master <pre> bayu@x230:~/Documents/Tesis/Fuzzgoat> python3 master.py Default direktori dari project adalah /home/bayu/Documents/Tesis/Fuzzgoat/ Silahkan pilih seed awal: █ </pre>	<i>Valid</i>
USR-02	Pengguna menentukan <i>seed</i> <pre> bayu@x230:~/Documents/Tesis/Fuzzgoat> python3 master.py Default direktori dari project adalah /home/bayu/Documents/Tesis/Fuzzgoat/ Silahkan pilih seed awal: abc File tidak ditemukan Silahkan pilih seed awal: █ bayu@x230:~/Documents/Tesis/Fuzzgoat> python3 master.py Default direktori dari project adalah /home/bayu/Documents/Tesis/Fuzzgoat/ Silahkan pilih seed awal: abc File tidak ditemukan Silahkan pilih seed awal: seed > Copy Seed Mengcopy seed ke: fuzzer0 Silahkan pilih program target: █ </pre>	<i>Valid</i>
USR-03	Pengguna menentukan program yang diuji	<i>Valid</i>

Kode	Hasil Pengujian	Keterangan
	<pre> bayu@x230:~/Documents/Tesis/Fuzzgoat> python3 master.py Default direktori dari project adalah /home/bayu/Documents/Tesis/Fuzzgoat/ Silahkan pilih seed awal: abc File tidak ditemukan Silahkan pilih seed awal: seed > Copy Seed Mengcopy seed ke: fuzzer0 Silahkan pilih program target: program File tidak ditemukan Silahkan pilih program target: █ bayu@x230:~/Documents/Tesis/Fuzzgoat> python3 master.py Default direktori dari project adalah /home/bayu/Documents/Tesis/Fuzzgoat/ Silahkan pilih seed awal: seed > Copy Seed Mengcopy seed ke: fuzzer0 Silahkan pilih program target: fuzz > Copy Target Program Mengcopy fuzz ke: fuzzer0 Mengcopy fuzz ke: fuzzer1 </pre>	
USR-04	<p>Pengguna mengeksekusi aplikasi master untuk menjalankan <i>container</i></p> <pre> bayu@x230:~/Documents/Tesis/Fuzzgoat> python3 master.py Default direktori dari project adalah /home/bayu/Documents/Tesis/Fuzzgoat/ Silahkan pilih seed awal: seed > Copy Seed Mengcopy seed ke: fuzzer0 Silahkan pilih program target: fuzz > Copy Target Program Mengcopy fuzz ke: fuzzer0 Mengcopy fuzz ke: fuzzer1 > Start Container fuzzer0 fuzzer1 </pre>	<i>Valid</i>
USR-05	<p>Pengguna mengeksekusi aplikasi master untuk menjalankan <i>fuzzer</i></p> <pre> > Start Fuzzing dengan fuzzer0 Warning: DESKTOP_STARTUP_ID not set and no fallback available. </pre>	<i>Valid</i>
USR-06	<p>Pengguna mengeksekusi aplikasi master untuk membaca <i>file plot_data</i></p> <pre> > Rename file fuzzer_stats dan plot_data fuzzer0 cp: -r not specified; omitting directory '/home/bayu/Documents/Tesis/Fuzzgoat/fuzzer0/out/queue/.state' </pre>	<i>Valid</i>
USR-07	<p>Pengguna menghentikan <i>fuzzer</i></p> <pre> +++ Testing aborted by user +++ [+] We're done here. Have a nice day! bayu@x230:~/Documents/Tesis/Fuzzgoat> █ </pre>	<i>Valid</i>

Kode	Hasil Pengujian	Keterangan																																																						
USR-08	<p>Pengguna mengeksekusi aplikasi master untuk melakukan mutasi bit flip</p> <table><thead><tr><th>id</th><th>file_name</th><th>value</th><th>hex_value</th><th>fuzzer</th><th>status</th></tr></thead><tbody><tr><td>1</td><td>id:000000,orig:seed</td><td>c</td><td>630a</td><td>fuzzer0</td><td>1</td></tr><tr><td>2</td><td>id:000001,src:000000,op:flip1,pos:0,+cov</td><td>a</td><td>610a</td><td>fuzzer0</td><td>1</td></tr><tr><td>3</td><td>id:000002,src:000001,op:havoc,rep:4,+cov</td><td>af</td><td>61660a</td><td>fuzzer0</td><td>1</td></tr><tr><td>4</td><td>id:000000,orig:id:000001,src:000000,op:flip1,pos:0,+cov</td><td>q</td><td>710a</td><td>fuzzer1</td><td>1</td></tr><tr><td>5</td><td>id:000001,src:000000,op:flip1,pos:0,+cov</td><td>a\</td><td>615c0a</td><td>fuzzer1</td><td>2</td></tr><tr><td>6</td><td>id:000002,src:000001,op:havoc,rep:32,+cov</td><td>afff</td><td>61666666</td><td>fuzzer1</td><td>1</td></tr><tr><td>7</td><td>id:000003,src:000002,op:arith8,pos:2,val:+6,+cov</td><td>a1ff</td><td>61666c66</td><td>fuzzer1</td><td>1</td></tr><tr><td>8</td><td>id:000003,src:000002,op:havoc,rep:2,+cov</td><td>a1f</td><td>61666c</td><td>fuzzer0</td><td>1</td></tr></tbody></table>	id	file_name	value	hex_value	fuzzer	status	1	id:000000,orig:seed	c	630a	fuzzer0	1	2	id:000001,src:000000,op:flip1,pos:0,+cov	a	610a	fuzzer0	1	3	id:000002,src:000001,op:havoc,rep:4,+cov	af	61660a	fuzzer0	1	4	id:000000,orig:id:000001,src:000000,op:flip1,pos:0,+cov	q	710a	fuzzer1	1	5	id:000001,src:000000,op:flip1,pos:0,+cov	a\	615c0a	fuzzer1	2	6	id:000002,src:000001,op:havoc,rep:32,+cov	afff	61666666	fuzzer1	1	7	id:000003,src:000002,op:arith8,pos:2,val:+6,+cov	a1ff	61666c66	fuzzer1	1	8	id:000003,src:000002,op:havoc,rep:2,+cov	a1f	61666c	fuzzer0	1	Valid
id	file_name	value	hex_value	fuzzer	status																																																			
1	id:000000,orig:seed	c	630a	fuzzer0	1																																																			
2	id:000001,src:000000,op:flip1,pos:0,+cov	a	610a	fuzzer0	1																																																			
3	id:000002,src:000001,op:havoc,rep:4,+cov	af	61660a	fuzzer0	1																																																			
4	id:000000,orig:id:000001,src:000000,op:flip1,pos:0,+cov	q	710a	fuzzer1	1																																																			
5	id:000001,src:000000,op:flip1,pos:0,+cov	a\	615c0a	fuzzer1	2																																																			
6	id:000002,src:000001,op:havoc,rep:32,+cov	afff	61666666	fuzzer1	1																																																			
7	id:000003,src:000002,op:arith8,pos:2,val:+6,+cov	a1ff	61666c66	fuzzer1	1																																																			
8	id:000003,src:000002,op:havoc,rep:2,+cov	a1f	61666c	fuzzer0	1																																																			
USR-09	<p>Pengguna mengeksekusi aplikasi master untuk menyimpan data <i>seed</i> ke dalam <i>database</i></p> <table><thead><tr><th>id</th><th>file_name</th><th>value</th><th>hex_value</th><th>fuzzer</th><th>status</th></tr></thead><tbody><tr><td>1</td><td>id:000000,orig:seed</td><td>c</td><td>630a</td><td>fuzzer0</td><td>1</td></tr><tr><td>2</td><td>id:000001,src:000000,op:flip1,pos:0,+cov</td><td>a</td><td>610a</td><td>fuzzer0</td><td>1</td></tr><tr><td>3</td><td>id:000002,src:000001,op:havoc,rep:4,+cov</td><td>af</td><td>61660a</td><td>fuzzer0</td><td>1</td></tr><tr><td>4</td><td>id:000000,orig:id:000001,src:000000,op:flip1,pos:0,+cov</td><td>q</td><td>710a</td><td>fuzzer1</td><td>1</td></tr><tr><td>5</td><td>id:000001,src:000000,op:flip1,pos:0,+cov</td><td>a\</td><td>615c0a</td><td>fuzzer1</td><td>2</td></tr><tr><td>6</td><td>id:000002,src:000001,op:havoc,rep:32,+cov</td><td>afff</td><td>61666666</td><td>fuzzer1</td><td>1</td></tr><tr><td>7</td><td>id:000003,src:000002,op:arith8,pos:2,val:+6,+cov</td><td>a1ff</td><td>61666c66</td><td>fuzzer1</td><td>1</td></tr><tr><td>8</td><td>id:000003,src:000002,op:havoc,rep:2,+cov</td><td>a1f</td><td>61666c</td><td>fuzzer0</td><td>1</td></tr></tbody></table>	id	file_name	value	hex_value	fuzzer	status	1	id:000000,orig:seed	c	630a	fuzzer0	1	2	id:000001,src:000000,op:flip1,pos:0,+cov	a	610a	fuzzer0	1	3	id:000002,src:000001,op:havoc,rep:4,+cov	af	61660a	fuzzer0	1	4	id:000000,orig:id:000001,src:000000,op:flip1,pos:0,+cov	q	710a	fuzzer1	1	5	id:000001,src:000000,op:flip1,pos:0,+cov	a\	615c0a	fuzzer1	2	6	id:000002,src:000001,op:havoc,rep:32,+cov	afff	61666666	fuzzer1	1	7	id:000003,src:000002,op:arith8,pos:2,val:+6,+cov	a1ff	61666c66	fuzzer1	1	8	id:000003,src:000002,op:havoc,rep:2,+cov	a1f	61666c	fuzzer0	1	Valid
id	file_name	value	hex_value	fuzzer	status																																																			
1	id:000000,orig:seed	c	630a	fuzzer0	1																																																			
2	id:000001,src:000000,op:flip1,pos:0,+cov	a	610a	fuzzer0	1																																																			
3	id:000002,src:000001,op:havoc,rep:4,+cov	af	61660a	fuzzer0	1																																																			
4	id:000000,orig:id:000001,src:000000,op:flip1,pos:0,+cov	q	710a	fuzzer1	1																																																			
5	id:000001,src:000000,op:flip1,pos:0,+cov	a\	615c0a	fuzzer1	2																																																			
6	id:000002,src:000001,op:havoc,rep:32,+cov	afff	61666666	fuzzer1	1																																																			
7	id:000003,src:000002,op:arith8,pos:2,val:+6,+cov	a1ff	61666c66	fuzzer1	1																																																			
8	id:000003,src:000002,op:havoc,rep:2,+cov	a1f	61666c	fuzzer0	1																																																			
USR-10	<p>Pengguna mengeksekusi aplikasi master untuk menyimpan data <i>fuzzer_stats</i> ke dalam <i>database</i></p> <table><thead><tr><th>id</th><th>run_time</th><th>cycles_dor</th><th>execs_per</th><th>paths_tota</th><th>paths_four</th><th>cur_path</th><th>uniq_crash</th><th>uniq_hang</th><th>command_line</th><th>fuzzer</th></tr></thead><tbody><tr><td>1</td><td>1 0:00:01</td><td>30</td><td>0</td><td>2</td><td>1</td><td>1</td><td>0</td><td>0</td><td>afl-fuzz -i in -o out -T fuzz-fuzzer0 /fuzz @@</td><td>fuzzer0</td></tr><tr><td>2</td><td>2 0:01:04</td><td>72</td><td>1535.47</td><td>4</td><td>2</td><td>3</td><td>0</td><td>0</td><td>afl-fuzz -i -o out -T fuzz-fuzzer0 /fuzz @@</td><td>fuzzer0</td></tr><tr><td>3</td><td>3 0:01:05</td><td>38</td><td>1551.96</td><td>4</td><td>3</td><td>3</td><td>1</td><td>0</td><td>afl-fuzz -i in -o out -T fuzz-fuzzer1 /fuzz @@</td><td>fuzzer1</td></tr></tbody></table>	id	run_time	cycles_dor	execs_per	paths_tota	paths_four	cur_path	uniq_crash	uniq_hang	command_line	fuzzer	1	1 0:00:01	30	0	2	1	1	0	0	afl-fuzz -i in -o out -T fuzz-fuzzer0 /fuzz @@	fuzzer0	2	2 0:01:04	72	1535.47	4	2	3	0	0	afl-fuzz -i -o out -T fuzz-fuzzer0 /fuzz @@	fuzzer0	3	3 0:01:05	38	1551.96	4	3	3	1	0	afl-fuzz -i in -o out -T fuzz-fuzzer1 /fuzz @@	fuzzer1	Valid										
id	run_time	cycles_dor	execs_per	paths_tota	paths_four	cur_path	uniq_crash	uniq_hang	command_line	fuzzer																																														
1	1 0:00:01	30	0	2	1	1	0	0	afl-fuzz -i in -o out -T fuzz-fuzzer0 /fuzz @@	fuzzer0																																														
2	2 0:01:04	72	1535.47	4	2	3	0	0	afl-fuzz -i -o out -T fuzz-fuzzer0 /fuzz @@	fuzzer0																																														
3	3 0:01:05	38	1551.96	4	3	3	1	0	afl-fuzz -i in -o out -T fuzz-fuzzer1 /fuzz @@	fuzzer1																																														
USR-11	<p>Pengguna mengeksekusi aplikasi master untuk melakukan proses pemilihan <i>seed</i> dari beberapa <i>seed</i> yang dihasilkan</p> <pre>> Buat Seed Baru untuk fuzzer1 corpus minimization tool for afl-fuzz by <lcamtuf@google.com> [*] Testing the target binary... [+] OK, 4 tuples recorded. [*] Obtaining traces for input files in '/home/bayu/Documents/Tesis/Fuzzgoat/seed-cmin/'... Processing file 3/3... [*] Sorting trace sets (this may take a while)... [+] Found 8 unique tuples across 3 files. [*] Finding best candidates for each tuple... Processing file 3/3... [*] Sorting candidate list (be patient)... [*] Processing candidates and writing output files... Processing tuple 8/8... [+] Narrowed down to 3 files, saved in '/home/bayu/Documents/Tesis/Fuzzgoat/fuzzer1/in/'.</pre> <p>Seed awal untuk fuzzer1 adalah id:000002,src:000001,op:havoc,rep:2,+cov</p>	Valid																																																						
USR-12	<p>Pengguna mengeksekusi aplikasi master untuk menjalankan sinkronisasi</p> <pre>> Jalankan sinkronisasi seed dan bitmap</pre>	Valid																																																						
USR-13	<p>Pengguna menghentikan <i>container</i></p>	Valid																																																						

Kode	Hasil Pengujian	Keterangan
	<pre> :: Proses fuzzing paralel sedang berjalan! Untuk menghentikan fuzzer0, silahkan tekan [Enter]: Terminate fuzzer0 > Stop Container fuzzer0 Untuk menghentikan fuzzer1, silahkan tekan [Enter]: Terminate fuzzer1 > Stop Container fuzzer1 :: Proses fuzzing paralel selesai. bayu@x230:~/Documents/Tesis/Fuzzgoat> █ </pre>	
FZR-01	<p><i>Fuzzer menghasilkan seed</i></p> <pre> bayu@x230:~/Documents/Tesis/Fuzzgoat/fuzzer0/out/queue> ls -l total 16 -rwxr-xr-x 2 bayu users 2 Jun 10 16:02 id:000000,orig:seed -rw----- 1 bayu users 2 Jun 10 16:03 id:000001,src:000000,op:flip1,pos:0,+cov -rw----- 1 bayu users 3 Jun 10 16:03 id:000002,src:000001,op:havoc,rep:4,+cov -rw----- 1 bayu users 3 Jun 10 16:03 id:000003,src:000002,op:havoc,rep:2,+cov bayu@x230:~/Documents/Tesis/Fuzzgoat/fuzzer0/out/queue> █ bayu@x230:~/Documents/Tesis/Fuzzgoat/fuzzer1/out/queue> ls -l total 16 -rw----- 2 bayu users 2 Jun 10 16:03 id:000000,orig:id:000001,src:000000,op:flip1,pos:0,+cov -rw----- 1 bayu users 3 Jun 10 16:03 id:000001,src:000000,op:flip1,pos:0,+cov -rw----- 1 bayu users 4 Jun 10 16:03 id:000002,src:000001,op:havoc,rep:32,+cov -rw----- 1 bayu users 4 Jun 10 16:03 id:000003,src:000002,op:arith8,pos:2,val:+6,+cov bayu@x230:~/Documents/Tesis/Fuzzgoat/fuzzer1/out/queue> █ </pre>	Valid
FZR-02	<p><i>Fuzzer menghasilkan file fuzzer_stats</i></p> <pre> start_time : 1591779795 last_update : 1591779859 fuzzer_pid : 17878 cycles_done : 72 execs_done : 113451 execs_per_sec : 1535.47 paths_total : 4 paths_favored : 4 paths_found : 2 paths_imported : 0 max_depth : 4 cur_path : 3 pending_favs : 0 pending_total : 0 variable_paths : 0 stability : 100.00% bitmap_cvg : 0.02% unique_crashes : 0 unique_hangs : 0 last_path : 1591779805 last_crash : 0 last_hang : 0 execs_since_crash : 113451 exec_timeout : 20 afl_banner : fuzz-fuzzer0 afl_version : 2.52b target_mode : default command_line : afl-fuzz -i- -o out -T fuzz-fuzzer0 ./fuzz @@ </pre>	Valid

Kode	Hasil Pengujian	Keterangan																																																						
	<pre>start_time : 1591779798 last_update : 1591779863 fuzzer_pid : 14 cycles_done : 38 execs_done : 112144 execs_per_sec : 1551.96 paths_total : 4 paths_favored : 4 paths_found : 3 paths_imported : 0 max_depth : 4 cur_path : 3 pending_favs : 0 pending_total : 0 variable_paths : 0 stability : 100.00% bitmap_cvg : 0.02% unique_crashes : 1 unique_hangs : 0 last_path : 1591779799 last_crash : 1591779840 last_hang : 0 execs_since_crash : 39233 exec_timeout : 20 afl_banner : fuzz-fuzzer1 afl_version : 2.52b target_mode : default command_line : afl-fuzz -i in -o out -T fuzz-fuzzer1 ./fuzz @@</pre>																																																							
FZR-03	<p><i>Fuzzer menjalankan proses fuzzing</i></p> <div><p>american fuzzy lop 2.52b (fuzz-fuzzer0)</p><div><div><p>process timing</p><p>run time : 0 days, 0 hrs, 0 min, 52 sec</p><p>last new path : none seen yet</p><p>last uniq crash : none seen yet</p><p>last uniq hang : none seen yet</p></div><div><p>overall results</p><p>cycles done : 210</p><p>total paths : 3</p><p>uniq crashes : 0</p><p>uniq hangs : 0</p></div></div><div><div><p>cycle progress</p><p>now processing : 1 (33.33%)</p><p>paths timed out : 0 (0.00%)</p></div><div><p>map coverage</p><p>map density : 0.01% / 0.01%</p><p>count coverage : 1.00 bits/tuple</p></div></div><div><div><p>stage progress</p><p>now trying : havoc</p><p>stage execs : 57/256 (22.27%)</p><p>total execs : 161k</p><p>exec speed : 2858/sec</p></div><div><p>findings in depth</p><p>favored paths : 3 (100.00%)</p><p>new edges on : 3 (100.00%)</p><p>total crashes : 0 (0 unique)</p><p>total tmounts : 0 (0 unique)</p></div></div><div><div><p>fuzzing strategy yields</p><p>bit flips : 0/0, 0/0, 0/0</p><p>byte flips : 0/0, 0/0, 0/0</p><p>arithmetics : 0/0, 0/0, 0/0</p><p>known ints : 0/0, 0/0, 0/0</p><p>dictionary : 0/0, 0/0, 0/0</p><p>havoc : 0/161k, 0/0</p><p>trim : n/a, n/a</p></div><div><p>path geometry</p><p>levels : 3</p><p>pending : 0</p><p>pend fav : 0</p><p>own finds : 0</p><p>imported : n/a</p><p>stability : 100.00%</p></div></div><p>[cpu:156%]</p></div>	Valid																																																						
ASD-01	<p><i>Agen melakukan mutasi acak</i></p> <table><thead><tr><th>id</th><th>file_name</th><th>value</th><th>hex_value</th><th>fuzzer</th><th>status</th></tr></thead><tbody><tr><td>1</td><td>id:000000,orig:seed</td><td>c</td><td>630a</td><td>fuzzer0</td><td>1</td></tr><tr><td>2</td><td>id:000001,src:000000,op:flip1,pos:0,+cov</td><td>a</td><td>610a</td><td>fuzzer0</td><td>1</td></tr><tr><td>3</td><td>id:000002,src:000001,op:havoc,rep:4,+cov</td><td>af</td><td>61660a</td><td>fuzzer0</td><td>1</td></tr><tr><td>4</td><td>id:000000,orig:000001,src:000000,op:flip1,pos:0,+cov</td><td>q</td><td>710a</td><td>fuzzer1</td><td>1</td></tr><tr><td>5</td><td>id:000001,src:000000,op:flip1,pos:0,+cov</td><td>a\</td><td>615c0a</td><td>fuzzer1</td><td>2</td></tr><tr><td>6</td><td>id:000002,src:000001,op:havoc,rep:32,+cov</td><td>a\ff</td><td>61666666</td><td>fuzzer1</td><td>1</td></tr><tr><td>7</td><td>id:000003,src:000002,op:arith8,pos:2,val:+6,+cov</td><td>a\ff</td><td>61666c66</td><td>fuzzer1</td><td>1</td></tr><tr><td>8</td><td>id:000003,src:000002,op:havoc,rep:2,+cov</td><td>a\ff</td><td>61666c</td><td>fuzzer0</td><td>1</td></tr></tbody></table>	id	file_name	value	hex_value	fuzzer	status	1	id:000000,orig:seed	c	630a	fuzzer0	1	2	id:000001,src:000000,op:flip1,pos:0,+cov	a	610a	fuzzer0	1	3	id:000002,src:000001,op:havoc,rep:4,+cov	af	61660a	fuzzer0	1	4	id:000000,orig:000001,src:000000,op:flip1,pos:0,+cov	q	710a	fuzzer1	1	5	id:000001,src:000000,op:flip1,pos:0,+cov	a\	615c0a	fuzzer1	2	6	id:000002,src:000001,op:havoc,rep:32,+cov	a\ff	61666666	fuzzer1	1	7	id:000003,src:000002,op:arith8,pos:2,val:+6,+cov	a\ff	61666c66	fuzzer1	1	8	id:000003,src:000002,op:havoc,rep:2,+cov	a\ff	61666c	fuzzer0	1	Valid
id	file_name	value	hex_value	fuzzer	status																																																			
1	id:000000,orig:seed	c	630a	fuzzer0	1																																																			
2	id:000001,src:000000,op:flip1,pos:0,+cov	a	610a	fuzzer0	1																																																			
3	id:000002,src:000001,op:havoc,rep:4,+cov	af	61660a	fuzzer0	1																																																			
4	id:000000,orig:000001,src:000000,op:flip1,pos:0,+cov	q	710a	fuzzer1	1																																																			
5	id:000001,src:000000,op:flip1,pos:0,+cov	a\	615c0a	fuzzer1	2																																																			
6	id:000002,src:000001,op:havoc,rep:32,+cov	a\ff	61666666	fuzzer1	1																																																			
7	id:000003,src:000002,op:arith8,pos:2,val:+6,+cov	a\ff	61666c66	fuzzer1	1																																																			
8	id:000003,src:000002,op:havoc,rep:2,+cov	a\ff	61666c	fuzzer0	1																																																			
ASD-02	<p><i>Agen memodifikasi file seed</i></p> <div> id_000001,src_000000,op_flip1,pos_0,+cov - Notepad</div> <div>File Edit Format View Help</div> <div>a\</div>	Valid																																																						

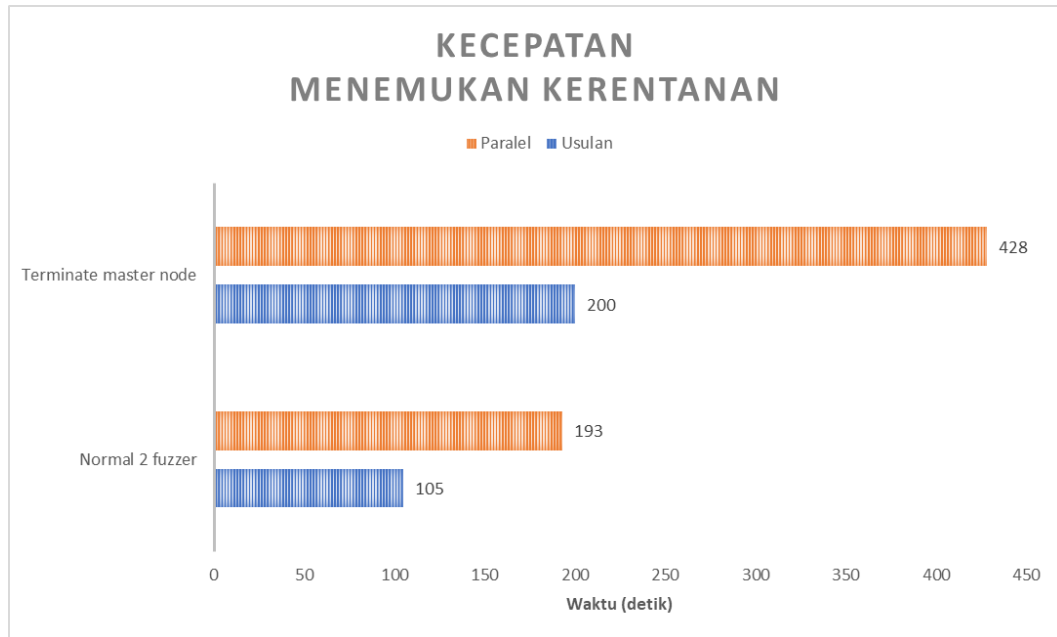
Kode	Hasil Pengujian	Keterangan
APC-01	<p>Agen melakukan sinkronisasi file <i>bitmap</i></p> <pre> bayu@x230:~> pgrep -f sync.py 6438 12378 bayu@x230:~> pgrep -f sync_bitmap.py 18686 bayu@x230:~> █ Bitmap 0: 28aa799902ad7f8247a9c7b748802e02 Bitmap 1: 493bbbebef3059e647e47c5350ffad2 Gabungan: 4300abf54e22ba6548b607f4a1f89a41 Bitmap 0: 771fe273ecfbb43fc1e24396ddacc408 Bitmap 1: e89ebc4fd41ff782050b986a03b6da8e Gabungan: 30c1ad00490e2d091a717c074c4b73ba Bitmap 0: 6c5a6a783687982032ffe0504bafce74 Bitmap 1: 83f3d54c421cbb89a1be4e118b9ec860 Gabungan: d1f991ad8100cbfcaaae4de6f58c2611 </pre>	<i>Valid</i>

IV.2.2 Pengujian Kualitas

Pengujian kualitas mengukur kecepatan sistem dalam menemukan kerentanan pada program yang diuji. Program yang diuji akan mengalami *crash* ketika menerima input kata afl!. Setelah dilakukan pengujian, didapatkan hasil pengujian seperti tabel III.4 dan grafik hasil pengujian seperti gambar IV.1 yang menunjukkan bahwa AFL paralel memerlukan waktu sekitar 3 menit untuk menemukan kerentanan pada program yang diuji. Waktu tersebut meningkat ketika *master node* dihentikan saat proses *fuzzing* berlangsung menjadi sekitar 7 menit. Sedangkan pada sistem yang dirancang memerlukan waktu sekitar 2 menit pada kondisi normal dan membutuhkan waktu sekitar 3 menit ketika salah satu *fuzzer* dihentikan pada saat proses *fuzzing* berlangsung.

Tabel IV.4 Hasil pengujian kualitas

	<i>Fuzzing dengan 2 fuzzer</i>			<i>Fuzzing dengan terminate master node</i>		
	Waktu Mulai	Waktu Selesai	Durasi	Waktu Mulai	Waktu Selesai	Durasi
AFL Paralel	16:31:54	16:35:07	00:03:13	17:03:49	17:10:57	00:07:08
Usulan	16:11:31	16:13:16	00:01:45	19:17:25	19:20:45	00:03:20



Gambar IV.1 Grafik hasil pengujian kualitas

Bab V Penutup

V.1 Kesimpulan

Berdasarkan penelitian yang telah dilakukan mengenai perancangan dan implementasi sistem *fuzzing* paralel, maka dapat disimpulkan sebagai berikut:

1. Sistem *fuzzing* paralel yang menerapkan mutasi deterministik dan non deterministik berhasil dibuat sehingga proses *fuzzing* tetap berjalan dengan kedua teknik mutasi meskipun *master node* mengalami *crash* saat proses *fuzzing* sedang berlangsung.
2. Sinkronisasi informasi *seed* dan *path coverage* dapat dicapai dengan menerapkan aplikasi monitoring yang berperan sebagai agen.
3. Penggunaan mutasi deterministik pada setiap *fuzzer* dalam mode paralel dapat mempersingkat waktu pencarian *bug*.

V.2 Saran

1. Aplikasi untuk sinkronisasi *seed* perlu disalin ke setiap *container* dan dilakukan perubahan secara manual pada kode sumber untuk mengubah nama *fuzzer* disesuaikan dengan nama *container*.
2. Hasil rancangan sistem masih bisa ditingkatkan agar dapat diimplementasikan pada beberapa komputer yang terkoneksi jaringan.

DAFTAR PUSTAKA

- [1] G. McGraw, "Software security," *IEEE Secur. Privacy Mag.*, vol. 2, no. 2, pp. 80–83, Mar. 2004, doi: 10.1109/MSECP.2004.1281254.
- [2] H. Liang, X. Pei, X. Jia, W. Shen, and J. Zhang, "Fuzzing: State of the Art," *IEEE Trans. Rel.*, vol. 67, no. 3, pp. 1199–1218, Sep. 2018, doi: 10.1109/TR.2018.2834476.
- [3] B. P. Miller, L. Fredriksen, and B. So, "An empirical study of the reliability of UNIX utilities," *Commun. ACM*, vol. 33, no. 12, pp. 32–44, Dec. 1990, doi: 10.1145/96267.96279.
- [4] C. Lemieux and K. Sen, "FairFuzz: a targeted mutation strategy for increasing greybox fuzz testing coverage," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering - ASE 2018*, Montpellier, France, 2018, pp. 475–485, doi: 10.1145/3238147.3238176.
- [5] J. Liang, Y. Jiang, Y. Chen, M. Wang, C. Zhou, and J. Sun, "PAFL: extend fuzzing optimizations of single mode to industrial parallel mode," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018*, Lake Buena Vista, FL, USA, 2018, pp. 809–814, doi: 10.1145/3236024.3275525.
- [6] A. Kossiakoff, W. N. Sweet, S. J. Seymour, and S. M. Biemer, "Systems Engineering Principles and Practice," p. 559.
- [7] Y. Bassil, "A Simulation Model for the Waterfall Software Development Life Cycle," *International Journal of Engineering*, vol. 2, no. 5, p. 7, 2012.
- [8] J. D. DeMott, R. J. Enbody, and W. F. Punch, "Systematic bug finding and fault localization enhanced with input data tracking," *Computers & Security*, vol. 32, pp. 130–157, Feb. 2013, doi: 10.1016/j.cose.2012.09.015.
- [9] P. Oehlert, "Violating Assumptions with Fuzzing," *IEEE Secur. Privacy Mag.*, vol. 3, no. 2, pp. 58–62, Mar. 2005, doi: 10.1109/MSP.2005.55.
- [10] C. Chen, B. Cui, J. Ma, R. Wu, J. Guo, and W. Liu, "A systematic review of fuzzing techniques," *Computers & Security*, vol. 75, pp. 118–137, Jun. 2018, doi: 10.1016/j.cose.2018.02.002.
- [11] S. Gan *et al.*, "CollAFL: Path Sensitive Fuzzing," in *2018 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, May 2018, pp. 679–696, doi: 10.1109/SP.2018.00040.
- [12] S. Sultan, I. Ahmad, and T. Dimitriou, "Container Security: Issues, Challenges, and the Road Ahead," *IEEE Access*, vol. 7, pp. 52976–52996, 2019, doi: 10.1109/ACCESS.2019.2911732.
- [13] A. Azab, "Enabling Docker Containers for High-Performance and Many-Task Computing," in *2017 IEEE International Conference on Cloud Engineering (IC2E)*, Vancouver, BC, Canada, Apr. 2017, pp. 279–285, doi: 10.1109/IC2E.2017.52.
- [14] Preeth E N, Fr. J. P. Mulerickal, B. Paul, and Y. Sastri, "Evaluation of Docker containers based on hardware utilization," in *2015 International Conference on Control Communication & Computing India (ICCC)*, Trivandrum, Kerala, India, Nov. 2015, pp. 697–700, doi: 10.1109/ICCC.2015.7432984.

- [15] M. Bohme, V.-T. Pham, and A. Roychoudhury, "Coverage-Based Greybox Fuzzing as Markov Chain," *IEEE Trans. Software Eng.*, vol. 45, no. 5, pp. 489–506, May 2019, doi: 10.1109/TSE.2017.2785841.
- [16] T. Yue, Y. Tang, B. Yu, P. Wang, and E. Wang, "LearnAFL: Greybox Fuzzing With Knowledge Enhancement," *IEEE Access*, vol. 7, pp. 117029–117043, 2019, doi: 10.1109/ACCESS.2019.2936235.
- [17] J. Ye, B. Zhang, R. Li, C. Feng, and C. Tang, "Program State Sensitive Parallel Fuzzing for Real World Software," *IEEE Access*, vol. 7, pp. 42557–42564, 2019, doi: 10.1109/ACCESS.2019.2905744.
- [18] Y. Chen *et al.*, "EnFuzz: Ensemble Fuzzing with Seed Synchronization among Diverse Fuzzers," *arXiv:1807.00182 [cs]*, May 2019, Accessed: Jan. 27, 2020. [Online]. Available: <http://arxiv.org/abs/1807.00182>.
- [19] C. Song, X. Zhou, Q. Yin, X. He, H. Zhang, and K. Lu, "P-Fuzz: A Parallel Grey-Box Fuzzing Framework," *Applied Sciences*, vol. 9, no. 23, p. 5100, Nov. 2019, doi: 10.3390/app9235100.
- [20] R. S. Pressman, *Software engineering: a practitioner's approach*, 7th ed. New York: McGraw-Hill Higher Education, 2010.

LAMPIRAN

Lampiran A Instalasi Kebutuhan Perangkat Lunak

A.1 Docker

1. Instalasi paket docker.

```
sudo zypper in docker docker-compose
```

2. Ijinkan docker jalan otomatis pada saat komputer boot.

```
sudo systemctl enable docker
```

3. Jalankan docker.

```
sudo systemctl start docker
```

4. Masukkan pengguna ke dalam grup docker.

```
sudo usermod -G docker -a <username>
```

5. Muat ulang konfigurasi baru grup agar dikenali sistem.

```
newgrp docker
```

6. Unduh openSUSE image dari docker hub.

```
docker pull opensuse/leap
```

```
bayu@x230:~> docker pull opensuse/leap
Using default tag: latest
latest: Pulling from opensuse/leap
fc43fae18940: Pull complete
Digest: sha256:47f55ca3101e989d0a2ca997ab4d880393816907708174efb2d55f248a38f874
Status: Downloaded newer image for opensuse/leap:latest
docker.io/opensuse/leap:latest
bayu@x230:~> █
```

A.2 AFL

- a. Instalasi di *host*

1. Jalankan perintah instalasi paket.

```
sudo zypper in afl
```

2. Cek hasil instalasi AFL di *host*.

```
afl-fuzz
```

```
bayu@x230:~> afl-fuzz
afl-fuzz 0.0.0 by <lcmtuf@google.com>

afl-fuzz [ options ] -- /path/to/fuzzed_app [ ... ]

Required parameters:

-i dir      - input directory with test cases
-o dir      - output directory for fuzzer findings

Execution control settings:

-f file     - location read by the fuzzed program (stdin)
-t msec     - timeout for each run (auto-scaled, 50-1000 ms)
-m megs     - memory limit for child process (50 MB)
-Q          - use binary-only instrumentation (QEMU mode)

Fuzzing behavior settings:

-d          - quick & dirty mode (skips deterministic steps)
-n          - fuzz without instrumentation (dumb mode)
-x dir      - optional fuzzer dictionary (see README)

Other stuff:

-T text     - text banner to show on the screen
-M / -S id  - distributed mode (see parallel_fuzzing.txt)
-C          - crash exploration mode (the peruvian rabbit thing)

For additional tips, please consult /usr/share/doc/packages/afl/README.

bayu@x230:~> █
```

b. Instalasi di *container*

1. Buat Dockerfile dengan menggunakan program editor teks. Gambar IV.1 menunjukkan isi dari file Dockerfile.

```
FROM opensuse/leap

RUN zypper ref && zypper --non-interactive up

RUN zypper --non-interactive in afl

#Setting environment
ENV AFL_I_DONT_CARE_ABOUT_MISSING_CRASHES=1 \
    AFL_SKIP_CPUFREQ=1 \
    AFL_NO_AFFINITY=1

RUN useradd -u 1000 afl

USER afl
```

2. Buat docker image yang berisi AFL.

```
docker build Dockerfile -t <nama_image>
```

```
bayu@x230:~/Documents/Tesis/container> docker build Dockerfile -t bayumahendra/afl:latest
Sending build context to Docker daemon 2.048kB
Step 1/6 : FROM opensuse/leap
----> a885319a41c1
Step 2/6 : RUN zypper ref && zypper --non-interactive up
----> Running in 20038366d998
Retrieving repository 'Non-OSS Repository' metadata [...done]
Building repository 'Non-OSS Repository' cache [...done]
Retrieving repository 'Main Repository' metadata [...done]
Building repository 'Main Repository' cache [...done]
Retrieving repository 'Main Update Repository' metadata [...done]
Building repository 'Main Update Repository' cache [...done]
Retrieving repository 'Update Repository (Non-Oss)' metadata [...done]
Building repository 'Update Repository (Non-Oss)' cache [...done]
All repositories have been refreshed.
Loading repository data...
Reading installed packages...

The following 4 packages are going to be upgraded:
  libgcrypt20 libgnutls30 libsolv-tools libzypp

4 packages to upgrade.
Overall download size: 3.9 MiB. Already cached: 0 B. After the operation, additional 12.1 KiB will be used.
Continue? [y/n/v/...? shows all options] (y): y
Retrieving package libgcrypt20-1.8.2-lp151.9.19.1.x86_64 (1/4), 416.6 KiB ( 1.1 MiB unpacked)
Retrieving: libgcrypt20-1.8.2-lp151.9.19.1.x86_64.rpm [.done (940 B/s)]
Retrieving package libgnutls30-3.6.7-lp151.2.12.1.x86_64 (2/4), 706.6 KiB ( 1.7 MiB unpacked)
Retrieving: libgnutls30-3.6.7-lp151.2.12.1.x86_64.rpm [.done (931.7 KiB/s)]
Retrieving package libsolv-tools-0.7.11-lp151.2.13.1.x86_64 (3/4), 626.3 KiB ( 5.0 MiB unpacked)
Retrieving: libsolv-tools-0.7.11-lp151.2.13.1.x86_64.rpm [.done (1.2 MiB/s)]
Retrieving package libzypp-17.19.0-lp151.2.13.1.x86_64 (4/4), 2.2 MiB ( 8.0 MiB unpacked)
Retrieving: libzypp-17.19.0-lp151.2.13.1.x86_64.rpm [.done (1.4 MiB/s)]
```

3. Buat docker container.

```
docker create --cpuset-cpus="<cpu>" -v
<direktori_host>:<direktori_container> --name
<nama_container> -it <nama_image> bash
```

```
bayu@x230:~> docker create --cpuset-cpus="0" -v /home/bayu/Documents/Tesis/Fuzzgoat/fuzzer0:/home/afl --name fuzzer0 -it bayumahendra/afl bash
280f76dbc9a46b428b087f8ad8276b5a2dd8c81d3be0406969dae6d9a86bec0
bayu@x230:~> docker create --cpuset-cpus="1" -v /home/bayu/Documents/Tesis/Fuzzgoat/fuzzer1:/home/afl --name fuzzer1 -it bayumahendra/afl bash
274e5ff9c21a9669ee1c05007cc2dd47bfad544ea7540ec6189269db45313413
bayu@x230:~> docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
274e5ff9c21a        bayumahendra/afl   "bash"             5 seconds ago      Created             -                  fuzzer1
280f76dbc9a4        bayumahendra/afl   "bash"             11 seconds ago     Created             -                  fuzzer0
bayu@x230:~>
```

4. Jalankan docker container.

```
docker start fuzzer0
```

5. Cek hasil instalasi AFL di *container*.

```
bayu@x230:~> docker exec fuzzer0 afl-fuzz
afl-fuzz by <lcantuf@google.com>

afl-fuzz [ options ] -- /path/to/fuzzed_app [ ... ]

Required parameters:

-i dir      - input directory with test cases
-o dir      - output directory for fuzzer findings

Execution control settings:

-f file     - location read by the fuzzed program (stdin)
-t msec     - timeout for each run (auto-scaled, 50-1000 ms)
-m megs     - memory limit for child process (50 MB)
-Q          - use binary-only instrumentation (QEMU mode)

Fuzzing behavior settings:

-d          - quick & dirty mode (skips deterministic steps)
-n          - fuzz without instrumentation (dumb mode)
-x dir      - optional fuzzer dictionary (see README)

Other stuff:

-T text     - text banner to show on the screen
-M / -S id  - distributed mode (see parallel_fuzzing.txt)
-C          - crash exploration mode (the peruvian rabbit thing)

For additional tips, please consult /usr/share/doc/packages/afl/README.

bayu@x230:~> █
```

A.3 SQLite

`sudo zypper in sqlite3`

```
bayu@x230:~> sudo zypper in sqlite3
Loading repository data...
Reading installed packages...
Resolving package dependencies...
```

```
The following NEW package is going to be installed:
  sqlite3
```

```
1 new package to install.
Overall download size: 143.2 KiB. Already cached: 0 B. After the operation,
additional 208.3 KiB will be used.
Continue? [y/n/v/...? shows all options] (y): y
Retrieving package sqlite3-3.28.0-lp151.2.3.1.x86_64
(1/1), 143.2 KiB (208.3 KiB unpacked)
Retrieving: sqlite3-3.28.0-lp151.2.3.1.x86_64.rpm .....[done]

Checking for file conflicts: .....[done]
(1/1) Installing: sqlite3-3.28.0-lp151.2.3.1.x86_64 .....[done]
bayu@x230:~> █
```

Lampiran B Kode Sumber

B.1 Program yang Diuji

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv) {
    char ptr[10];
    if(argc>1){
        FILE *fp = fopen(argv[1], "r");
        fgets(ptr, sizeof(ptr), fp);
    }
    else{
        fgets(ptr, sizeof(ptr), stdin);
    }
    printf("%s", ptr);
    if(ptr[0] == 'a')
    {
        if(ptr[1] == 'f')
        {
            if(ptr[2] == 'l')
            {
                if(ptr[3] == '!')
                {
                    abort();
                }
                else printf("%c",ptr[3]);
            }
            else printf("%c",ptr[2]);
        }
        else printf("%c",ptr[1]);
    }
    else printf("%c",ptr[0]);
}
```

```
    return 0;
}
```

B.2 Aplikasi Master

```
import os
import sqlite3
import datetime
import random
import string
import time

# [host]
direktori_proyek = '/home/bayu/Documents/Tesis/Fuzzgoat/'
direktori_cmin = '/home/bayu/Documents/Tesis/Fuzzgoat/seed-cmin/'
nama_database = 'fuzzgoat.db'

# [container]
nama_container = ['fuzzer0', 'fuzzer1']
direktori_afl = '/home/afl/'

class Masukan:
    def __init__(self, nama):
        self.nama = nama

    def copy(self, path_sumber, path_tujuan):
        perintah = "cp " + path_sumber + self.nama + " " + path_tujuan
        os.system(perintah)

class Container:
    def __init__(self, nama):
        self.nama = nama

    def jalan(self):
```

```

perintah = "docker start " + self.nama
os.system(perintah)

def berhenti(self):
    print("\n> Stop Container')
    perintah = "docker stop " + self.nama
    #print(perintah)
    os.system(perintah)

class Fuzzer:
    def __init__(self, nama):
        self.nama = nama
        self.direktori_out = direktori_proyek + self.nama + "/out/"

    def jalan(self, nama_target):
        perintah = "gnome-terminal --geometry=80x26 -- bash -c 'docker exec -t -w " + direktori_afl + " " + self.nama + " afl-fuzz -i in -o out -T " + nama_target +
        "-" + self.nama + " ./" + nama_target + " @@; exec bash'"
        #print(perintah)
        os.system(perintah)
        time.sleep(3)

    def berhenti(self):
        perintah = "docker exec " + self.nama + " kill afl-fuzz"
        #print(perintah)
        os.system(perintah)
        print("Terminate", self.nama)
        time.sleep(1.5)

    def resume(self, nama_target):
        print("\n> Resume Fuzzing dengan', self.nama)

```



```

        perintah = "gnome-terminal --geometry=80x26 -- bash -c 'docker exec -t -w
" + direktori_afl + " " + self.nama + " afl-fuzz -i- -o out -T " + nama_target + "-"
+ self.nama + " ./" + nama_target + " @@; exec bash"

        #print(perintah)
        os.system(perintah)
        time.sleep(3)

    def baca_plot_data(self):
        plot_data = self.direktori_out + 'plot_data'
        file = open(plot_data, 'r')
        isi_file = file.readlines()
        for i in range(1, len(isi_file)):
            cycles_done = isi_file[i].split(",")
            cd = int(cycles_done[1])
            unique_crashes = isi_file[i].split(",")
            uc = int(unique_crashes[7])
            if cd >= 1 or uc >= len(nama_container)-1:
                terminate = 1
                return terminate
            else:
                terminate = 0
                return terminate

    def rename_file_stats(self):
        src = self.direktori_out + 'fuzzer_stats'
        dst = self.direktori_out + 'fuzzer_stats_init'
        os.rename(src, dst)
        src = self.direktori_out + 'plot_data'
        dst = self.direktori_out + 'plot_data_init'
        os.rename(src, dst)
        print("\n> Rename file fuzzer_stats dan plot_data", self.nama)

```

```

def simpan_stats(self):
    file_stats = direktori_proyek + self.nama + '/out/' + 'fuzzer_stats'
    f = open(file_stats, 'r')
    baris = f.readlines()
    start_time = baris[0].split(":")
    last_update = baris[1].split(":")
    cycles_done = baris[3].split(":")
    execs_per_sec = baris[5].split(":")
    paths_total = baris[6].split(":")
    paths_found = baris[8].split(":")
    cur_path = baris[11].split(":")
    unique_crashes = baris[17].split(":")
    unique_hangs = baris[18].split(":")
    command_line = baris[27].split(":")
    st = datetime.datetime.fromtimestamp(int(start_time[1].strip()))
    lu = datetime.datetime.fromtimestamp(int(last_update[1].strip()))
    rt = lu - st
    cd = cycles_done[1].strip()
    es = execs_per_sec[1].strip()
    pt = paths_total[1].strip()
    pf = paths_found[1].strip()
    cp = cur_path[1].strip()
    uc = unique_crashes[1].strip()
    uh = unique_hangs[1].strip()
    cl = command_line[1].strip()
    parameter = (str(rt), cd, es, pt, pf, cp, uc, uh, cl, self.nama)
    con = sqlite3.connect(nama_database)
    cur = con.cursor()
    cur.execute("INSERT INTO stats(run_time, cycles_done, execs_per_sec,
paths_total, paths_found, cur_path, uniq_crashes, uniq_hangs, command_line,
fuzzer) VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?);", parameter)
    con.commit()

```

```
con.close()
```

```
class Mutator:
```

```
    def __init__(self, direktori, file):
```

```
        self.direktori = direktori
```

```
        self.file = file
```

```
    def bit_flip(self):
```

```
        file_seed = open(self.direktori + self.file, 'rb')
```

```
        nilai_seed = file_seed.read()
```

```
        list_nilai_seed = list(nilai_seed)
```

```
        index = random.randint(0, len(nilai_seed)-1)
```

```
        biner_seed = bin(ord(chr(nilai_seed[index])))
```

```
        posisi = random.randint(2, len(biner_seed)-1)
```

```
        list_biner_seed = list(biner_seed)
```

```
        if list_biner_seed[posisi]=='0':
```

```
            list_biner_seed[posisi] = '1'
```

```
        else:
```

```
            list_biner_seed[posisi] = '0'
```

```
        biner_seed_baru = ''.join(list_biner_seed)
```

```
        karakter_baru = int(biner_seed_baru, 2)
```

```
        list_nilai_seed[index] = karakter_baru
```

```
        i = 0
```

```
        for i in range(i, len(list_nilai_seed)):
```

```
            list_nilai_seed[i] = chr(list_nilai_seed[i])
```

```
            i+=1
```

```
        nilai_seed_baru = ''.join(list_nilai_seed)
```

```
        return nilai_seed_baru
```

```
    def update_file_seed(self, nilai_seed_baru):
```

```
        file_seed_baru = open(self.direktori + self.file, 'w')
```

```
        file_seed_baru.write(nilai_seed_baru)
```

```

file_seed_baru.close()

print("Default direktori dari project adalah", direktori_proyek)

# Pilih initial seed
while True:
    try:
        nama_seed = input("\nSilahkan pilih seed awal: ")
        if os.path.isfile(nama_seed)==True:
            seed_awal = Masukan(nama_seed)
            print('> Copy Seed')
            direktori_tujuan = direktori_proyek + nama_container[0] + "/in/"
            seed_awal.copy(direktori_proyek, direktori_tujuan)
            print('Mengcopy seed ke: ' + nama_container[0])
            break
        else:
            raise FileNotFoundError
    except FileNotFoundError:
        print("File tidak ditemukan")

# Pilih program target
while True:
    try:
        nama_target = input("\nSilahkan pilih program target: ")
        if os.path.isfile(nama_target)==True:
            target = Masukan(nama_target)
            print('> Copy Target Program')
            for i in nama_container:
                direktori_tujuan = direktori_proyek + i
                target.copy(direktori_proyek, direktori_tujuan)
                print('Mengcopy ' + nama_target + ' ke: ' + i)
            break
    
```

```

        else:
            raise FileNotFoundError
    except FileNotFoundError:
        print("File tidak ditemukan")

# Start container
print("\n> Start Container')
for i in nama_container:
    container = Container(i)
    container.jalan()

# Start fuzzer
print("\n> Start Fuzzing dengan', nama_container[0])
fuzzer = Fuzzer(nama_container[0])
fuzzer.jalan(nama_target)

# Baca plot_data
print("\n> Monitoring Seed')
while True:
    time.sleep(1)
    terminate = fuzzer.baca_plot_data()
    if terminate==1:
        print("\n> Terminate Fuzzer')
        fuzzer.berhenti()
        fuzzer.simpan_stats()
        fuzzer.rename_file_stats()
        break

# Buat seed baru
direktori_queue = direktori_proyek + nama_container[0] + "/out/queue/"
direktori_cmin = direktori_proyek + "seed-cmin/"
files = sorted(os.listdir(direktori_queue))

```

```

for file in files:
    seed = Masukkan(file)
    seed.copy(direktori_queue, direktori_cmin)
for i in range(1, len(nama_container)):
    jumlah_seed = 0
    print("\n> Buat Seed Baru untuk', nama_container[i])
    direktori_in = direktori_proyek + nama_container[i] + "/in/"
    perintah = "afl-cmin -i " + direktori_cmin + " -o " + direktori_in + " ./" +
nama_target + " @@ -C"
    os.system(perintah)
    for seed_awal in os.listdir(direktori_in):
        jumlah_seed+=1
    j = 0
    for seed_awal in sorted(os.listdir(direktori_in)):
        if j == jumlah_seed-1:
            nama_file = direktori_cmin + seed_awal
            os.remove(nama_file)
            #print("Hapus dari cmin", seed_awal)
            seed_awal_baru = seed_awal
        else:
            nama_file = direktori_in + seed_awal
            os.remove(nama_file)
            #print("Hapus dari in", seed_awal)
        j+=1
    print(f"Seed awal untuk {nama_container[i]} adalah {seed_awal_baru}")
    mutator = Mutator(direktori_in, seed_awal_baru)
    nilai_seed_baru = mutator.bit_flip()
    mutator.update_file_seed(nilai_seed_baru)

# Resume initial fuzzer
fuzzer.resume(nama_target)

```

```

# Parallel fuzzing
for i in range(1, len(nama_container)):
    fuzzer = Fuzzer(nama_container[i])
    fuzzer.jalan(nama_target)

# Sinkronisasi seed dan bitmap
time.sleep(1)
print("\n> Jalankan sinkronisasi seed dan bitmap")
for i in nama_container:
    perintah = "python3 " + direktori_proyek + i + "/sync.py &"
    os.system(perintah)
    time.sleep(1.5)
perintah = "python3 " + direktori_proyek + "sync_bitmap.py &"
os.system(perintah)

print("\n:: Proses fuzzing paralel sedang berjalan!")
while True:
    for i in nama_container:
        input(f"\nUntuk menghentikan {i}, silahkan tekan [Enter]: ")
        fuzzer = Fuzzer(i)
        container = Container(i)
        fuzzer.berhenti()
        fuzzer.simpan_stats()
        container.berhenti()
    break
print("\n:: Proses fuzzing paralel selesai.")

```

B.2 Agen Sinkronisasi Seed

```

import os
import sqlite3
import time
import datetime

```

```

import random
import string

direktori_proyek = '/home/bayu/Documents/Tesis/Fuzzgoat/'
nama_database = direktori_proyek + 'fuzzgoat.db'
# Sesuaikan nama_fuzzer dengan direktori container
nama_fuzzer = 'fuzzer0'

class AgenSync:
    def __init__(self, fuzzer):
        self.fuzzer = fuzzer
        self.direktori = direktori_proyek + fuzzer + "/out/queue/"

    def cek_duplikasi(self, nilai_hex):
        con = sqlite3.connect(nama_database)
        cur = con.cursor()
        cur.execute("SELECT COUNT(*) as total FROM seed WHERE
hex_value=?;", (nilai_hex, ))
        baris = cur.fetchone()
        total = baris[0]
        con.commit()
        con.close()
        return total

    def baca_file_seed(self):
        files = sorted(os.listdir(self.direktori))
        files.remove('.state')
        return files

    def simpan_seed(self, file, nilai_seed, hex_seed):
        con = sqlite3.connect(nama_database)
        cur = con.cursor()

```



```

        cur.execute("INSERT INTO seed(file_name, value, hex_value, fuzzer)
VALUES(?, ?, ?, ?);", (file, nilai_seed, hex_seed, self.fuzzer))

        con.commit()

        con.close()

def update_seed(self, nilai_seed_baru, hex_seed, hex_seed_baru):
    status='2'

    con = sqlite3.connect(nama_database)

    cur = con.cursor()

    cur.execute("UPDATE seed SET value=?, hex_value=?, status=? WHERE
hex_value=? AND fuzzer=?;", (nilai_seed_baru, hex_seed_baru, status,
hex_seed, self.fuzzer))

    con.commit()

    con.close()

def generator_acak(self, nilai_seed, hex_seed):
    list_nilai_seed = list(nilai_seed)

    posisi = random.randint(0,len(list_nilai_seed))

    #jumlah_karakter_acak = random.randint(1,3)

    #karakter_acak = ".join(random.choice(string.ascii_letters + string.digits +
string.punctuation) for i in range(jumlah_karakter_acak))

    #for karakter in karakter_acak:

    # list_nilai_seed.insert(posisi, ord(karakter))

    # posisi+=1

    karakter_acak = random.choice(string.ascii_letters + string.digits +
string.punctuation)

    list_nilai_seed.insert(posisi, ord(karakter_acak))

    list_nilai_seed_baru = list()

    nilai_seed_baru = ""

    for karakter_desimal in list_nilai_seed:

        karakter_ascii = chr(karakter_desimal)

        list_nilai_seed_baru.append(karakter_ascii)

```

```

        str_nilai_seed_baru = nilai_seed_baru.join(map(str, list_nilai_seed_baru))
        nilai_seed_baru = str.encode(str_nilai_seed_baru)
        hex_seed_baru = nilai_seed_baru.hex()
        seed_baru = [str_nilai_seed_baru, hex_seed_baru]
        return seed_baru

    def update_file_seed(self, file, nilai_seed_baru):
        file_seed_baru = open(self.direktori + file, 'w')
        file_seed_baru.write(nilai_seed_baru)
        file_seed_baru.close()

index = 0
agen = AgenSync(nama_fuzzer)
while(True):
    file = agen.baca_file_seed()
    if len(file)!=index:
        for i in range(index, len(file)):
            seed = open(direktori_proyek + nama_fuzzer + "/out/queue/" + file[i],
'rb')
            nilai_seed = seed.read()
            hex_seed = nilai_seed.hex()
            total = agen.cek_duplikasi(hex_seed)
            agen.simpan_seed(file[i], nilai_seed, hex_seed)
            if total>0:
                #if file[i].find('orig')== -1:
                seed_baru = agen.generator_acak(nilai_seed, hex_seed)
                nilai_seed_baru = seed_baru[0]
                hex_seed_baru = seed_baru[1]
                agen.update_seed(nilai_seed_baru, hex_seed, hex_seed_baru)
                agen.update_file_seed(file[i], nilai_seed_baru)
            i+=1
        index=len(file)

```

```
time.sleep(1)
```

B.3 Agen Sinkronisasi *Path Coverage*

```
import os
import hashlib
import time

nama_fuzzer = ['fuzzer0', 'fuzzer1']

class AgenSync:
    def __init__(self, fuzzer):
        self.fuzzer = fuzzer
        self.direktori = '/home/bayu/Documents/Tesis/Fuzzgoat/'
        self.file_bitmap = self.direktori + self.fuzzer + '/out/fuzz_bitmap'

    def generate_hash(self):
        file = open(self.file_bitmap, 'rb')
        isi_file = file.read()
        md5_hash = hashlib.md5()
        md5_hash.update(isi_file)
        digest = md5_hash.hexdigest()
        return digest

    def baca_bitmap(self):
        dec_bitmap = list()
        file = open(self.file_bitmap, 'rb')
        isi_file = file.read()
        for karakter in isi_file:
            dec_bitmap.append(karakter)
        return dec_bitmap

    def gabung_bitmap(self, dec_karakter, dec_karakter1):
```

```

        dec_karakter_baru = dec_karakter & dec_karakter1
        karakter_baru = chr(int(dec_karakter_baru))
        list_bitmap_baru.append(karakter_baru)
        return list_bitmap_baru

    def update_bitmap(self, bitmap_baru):
        bitmap = open(self.file_bitmap, 'wb')
        bitmap.write(bitmap_baru)
        bitmap.close()

#counter = 1

while True:
    # Baca bitmap
    list_dec_bitmap = list()
    list_digest = list()
    for fuzzer in nama_fuzzer:
        agen = AgenSync(fuzzer)
        dec_bitmap = agen.baca_bitmap()
        list_dec_bitmap.append(dec_bitmap)
        digest = agen.generate_hash()
        list_digest.append(digest)
    if list_digest[0] != list_digest[1]:
        # Merge bitmap
        list_bitmap_baru = list()
        for i in range(1, len(nama_fuzzer)):
            for j in range(0, 65536):
                dec_karakter = list_dec_bitmap[i-1][j]
                dec_karakter1 = list_dec_bitmap[i][j]
                if i == 1:
                    tmp_bitmap = agen.gabung_bitmap(dec_karakter, dec_karakter1)
                else:

```

```
        mrg_bitmap = agen.gabung_bitmap(tmp_bitmap[j], dec_karakter1)
        tmp_bitmap = mrg_bitmap
    list_bitmap_baru = tmp_bitmap
    bitmap_baru = bytes("").join(list_bitmap_baru), 'iso8859-1')
    # Update bitmap
    for fuzzer in nama_fuzzer:
        agen = AgenSync(fuzzer)
        agen.update_bitmap(bitmap_baru)
    merge_bitmap = open(agen.direktori + 'bitmap', 'wb')
    merge_bitmap.write(bitmap_baru)
    merge_bitmap.close()
time.sleep(1)
```