TUGAS AKHIR - IF184802

RANCANG BANGUN APLIKASI VULNERABILITY ASSESSMENT AND MANAGEMENT DENGAN METODE PASSIVE SCANNING DAN CVSS

GUSTI NGURAH SATRIA ARYAWAN
NRP 05111450000066

Dosen Pembimbing
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.
Bagus Jati Santoso, S.Kom.,Ph.D.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2019

TUGAS AKHIR – IF184802

RANCANG BANGUN APLIKASI VULNERABILITY ASSESSMENT AND MANAGEMENT DENGAN METODE PASSIVE SCANNING DAN CVSS

GUSTI NGURAH SATRIA ARYAWAN
NRP 05111540000066

Dosen Pembimbing
Royyana Muslim Ijtihadie, S.Kom.,M.Kom.,Ph.D.
Bagus Jati Santoso, S.Kom.,Ph.D.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2019

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESES – IF184802

DESIGN AND DEVELOPMENT APPLICATION VULNERABILITY ASSESSMENT AND MANAGEMENT WITH PASSIVE SCANNING METHOD AND CVSS

GUSTI NGURAH SATRIA ARYAWAN
NRP 05111540000066

Advisor
Royyana Muslim Ijtihadie, S.Kom.,M.Kom.,Ph.D.
Bagus Jati Santoso, S.Kom.,Ph.D.

DEPARTMENT OF INFORMATICS ENGINEERING
Faculty of Intelligent Electrical and Informatics Technology
Sepuluh Nopember Institute of Technology
Surabaya 2019

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

RANCANG BANGUN APLIKASI *VULNERABILITY ASSESSMENT AND MANAGEMENT* DENGAN METODE *PASSIVE SCANNING* DAN *CVSS*

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Arsitektur dan Jaringan Komputer
Program Studi S1 Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh:
GUSTI NGURAH SATRIA ARYAWAN
NRP: 05111540000066

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Royyana Muslim Ijtihadie, S.Kom., M.Eng., Ph.D. .....
(NIP. 197708242003041001) (Pembimbing 1)

Bagus Jati Santoso, S.Kom., Ph.D. .....
(NIP. 198611252018031001) (Pembimbing 2)

SURABAYA
Desember, 2019

(Halaman ini sengaja dikosongkan)

RANCANG BANGUN APLIKASI *VULNERABILITY AND MANAGEMENT* DENGAN METODE *PASSIVE SCANNING* DAN *CVSS*

Nama Mahasiswa	**: GUSTI NGURAH SATRIA ARYAWAN**
NRP	**: 05111540000066**
Jurusan	**: Departemen Teknik Informatika FTEIC-ITS**
Dosen Pembimbing 1	**: Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.**
Dosen Pembimbing 2	**: Bagus Jati Santoso, S.Kom.,Ph.D.**

Abstrak

Vulnerability Assessment adalah proses untuk mengidentifikasi, menilai dan memprioritaskan kerentanan dalam sistem komputer, aplikasi, dan infrastruktur jaringan. Dalam melakukan vulnerability assessment dapat menggunakan berbagai macam automated testing tools, seperti Nessus, Acunetix, Netsparker. Proses ini memberikan informasi tentang kerentanan yang ditemukan beserta cara menanggulanginya. Terdapat beberapa tipe dalam melakukan vulnerability assessment, salah satunya adalah application scans yang dapat digunakan untuk melakukan test pada sebuah website guna menemukan kerentanan yang umum.

Proses untuk menemukan kerentanan dapat menggunakan metode passive scanning. Passive scanning adalah metode dalam vulnerability detection dengan memanfaatkan informasi yang didapatkan dari data yang terkirim tanpa melakukan interaksi langsung. Passive scanning dapat memberikan informasi seperti sistem operasi yang digunakan, ports yang sedang berjalan dan aplikasi yang terinstall.

CVSS adalah sistem penilaian kerentanan berdasarkan karakteristik dan menghasilkan nilai numerik yang mencerminkan

tingkat kesulitan. Nilai numerik ini nantinya akan diubah menjadi nilai kualitatif seperti low, medium, high, dan critical untuk membantu dalam melakukan prioritas perbaikan. Sistem ini memiliki berbagai kelompok metrik untuk menghasilkan nilai seperti base score, temporal score, dan environmental score.

Pada tugas akhir ini, pemanfaatan vulnerability assessment diharapkan dapat membantu pihak pengembang dalan proses perbaikan kerentanan yang ditemukan pada website yang sedang dikembangkan ataupun yang telah dibuat. Berdasarkan hasil uji coba, sistem yang dibangun pada tugas akhir ini mampu menjalankan vulnerability assessment secara langsung ataupun terjadwal, serta mampu memberikan laporan kerentanan yang ditemukan saat proses ini selesai. Kelemahan yang terdapat pada sistem ini adalah konsumsi resource yang tinggi untuk setiap proses dari masing-masing tools yang terdapat pada vulnerability assessment ini.

Kata kunci: Passive scanning, Vulnerability Assessment, CVSS.

DESIGN AND DEVELOPMENT APPLICATION VULNERABILITY ASSESSMENT AND MANAGEMENT WITH PASSIVE SCANNING METHOD AND CVSS

Student's Name	**: GUSTI NGURAH SATRIA ARYAWAN**
Student's ID	**: 05111540000066**
Department	**: Department of Informatics Engineering ELECTICS-ITS**
First Advisor	**: Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.**
Second Advisor	**: Bagus Jati Santoso, S.Kom.,Ph.D.**

Abstract

Vulnerability assessment is the process of defining, identifying and prioritizing vulnerabilities in computer systems, applications and network infrastructures. In the process of vulnerability assessment the use of automated testing tools, such as Nessus, Acunetix, Netsparker. This process provides information about vulnerabilities discovered and how to overcome them. There are several types of vulnerability assessment, one of which is an application scans can be used to test websites in order to detect common vulnerability.

The process of finding vulnerability can use the passive scanning method. Passive scanning is a method of vulnerability detection that utilizes information obtained from data sent without direct interaction. Passive scanning can provide information such as the operation system used, ports that are running and installed applications.

CVSS is a system that considers characteristics and produces numerical score that determine the level of difficulty. This numerical score will be changed to qualitative score such as low, medium, high and critical to assist in making priority improvements. This system has various metric groups to produce

scores such as base scores, temporal scores, and environmental scores.

In this undergraduate thesis, the use of vulnerability assessment is expected to help the developer in the process of repairing the vulnerabilities found on website that are being developed or that have been created. Based on trial results, the system built in this undergraduate thesis is able to run the vulnerability assessment directly or on a scheduled basis and is able to provide report of the vulnerabilies found when this process is completed. The weekness of this system is high resource consumption for each process of each tool contained in this vulnerability assessment.

Keywords : Passive scanning, Vulnerability assessment, CVSS.

KATA PENGANTAR

Segala puji bagi Tuhan Yang Maha Esa, yang telah melimpahkan rahmat-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul **RANCANG BANGUN APLIKASI *VULNERABILITY ASSESSMENT AND MANAGEMENT* DENGAN METODE *PASSIVE SCANNING* DAN *CVSS***. Dengan pengerjaan Tugas Akhir ini, penulis bisa memperdalam dan meningkatkan apa yang telah didapatkan selama menempuh perkuliahan di Departemen Teknik Informatika ITS. Selain itu, penulis juga dapat menghasilkan suatu implementasi dari apa yang telah dipelajari selama proses perkuliahan. Selesainya Tugas Akhir ini tidak terlepas dari bantuan dan dukungan beberapa pihak. Pada kesempatan ini penulis mengucapkan syukur dan terima kasih kepada:

1. Tuhan Yang Maha Esa, atas anugerah dan pencerahan-Nya penulis dapat menyelesaikan Tugas Akhir dengan baik.
2. Kedua orang tua, Gusti Nyoman Arya serta Ni Ketut Adnyani, dan kedua saudara I Gusti Ayu Phuja Wahyuni serta I Gusti Lanang Wira Pradnya terima kasih atas doa dan bantuan moral serta material selama penulis belajar di Departemen Teknik Informatika ITS.
3. Bapak Royyana Muslim Ijtihadie,S.Kom.,M.Kom.,Ph.D. selaku pembimbing I dan Bapak Bagus Jati Santoso,S.Kom.,Ph.D. selaku pembimbing II yang telah membantu, membimbing, dan memotivasi penulis mulai dari pengerjaan proposal hingga terselesainya Tugas Akhir ini.
4. Ibu Dr.Eng. Chastine Fatichah, S.Kom, M.Kom, selaku kepala Departemen Teknik Informatika ITS pada masa pengerjaan Tugas Akhir. Bapak Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D, selaku koordinator Tugas Akhir, dan segenap dosen Departemen Informatika yang telah memberikan ilmu dan pengalamannya kepada

penulis sebagai bekal menjalani kehidupan yang sebenarnya.

5. Seluruh adminstrator Laboratorium AJK serta seluruh rekan–rekan TC 2015 yang sudah mendukung penulis selama perkuliahan.
6. Teman–teman Asrama Tirtha Gangga yang telah menemani dan bersedia menjadi teman berbagi cerita serta bermain selama penulis berada di Surabaya.
7. Teman-teman penulis dalam tim perlombaan *CTF*, Ronald Andrean, Rangga Senatama Putra, Drajad Bima Ajipangestu dan Aditya Pratama yang membantu penulis dalam setiap perlombaan yang diikuti.
8. Teman-teman *Seeds for the Future* 2019 yang menemani penulis selama dua minggu di Tiongkok saat penulis sedang mengerjakan tugas akhir ini.
9. Serta semua pihak yang turut membantu dalam penyelesaian Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan sehingga dengan kerendahan hati penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan kedepannya.

Surabaya, Desember 2019

Gusti Ngurah Satria Aryawan

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

DAFTAR KODE SUMBER

(Halaman ini sengaja dikosongkan)

BAB I
PENDAHULUAN

1.1 Latar Belakang

Cepatnya pertumbuhan dan perkembangan teknologi serta banyaknya informasi yang perlu disampaikan, membutuhkan sebuah aplikasi untuk menyampaikan informasi tersebut kepada masyarakat luas. Kemudahan menyebarkan informasi menyebabkan semakin banyak informasi diperoleh dari sumber yang tidak dapat dipertanggungjawabkan. Oleh karena itu diperlukan sebuah aplikasi yang dikelola oleh instansi yang bertanggung jawab. Pada kasus ini, Institut Teknologi Sepuluh Nopember (ITS) memiliki situs informasi berbasis web dengan nama domain its.ac.id sebagai penyebar informasi kepada masyarakat luas. Selain nama domain its.ac.id, ITS juga memiliki situs informasi berbasis web dengan *subdomain* yang jumlahnya lebih dari 1500.

Banyaknya situs informasi berbasis web yang dimiliki akan mempersulit pihak terkait dalam hal pemeliharaan dan pemeriksaan situs informasi tersebut. Hal ini dapat dimanfaatkan oleh pihak yang tidak bertanggung jawab untuk merusak atau menyebarkan informasi yang tidak benar melalui situs informasi milik ITS. Kesalahan dalam menyampaikan informasi ini dapat merugikan pihak ITS dan masyarakat luas. Untuk mencegah hal tersebut terjadi, dibutuhkan seseorang dengan pengetahuan dan pengalaman dalam menggunakan *automated testing tools* untuk memberikan masukan kepada pihak pengembang aplikasi maupun pihak yang mengelola aplikasi tersebut. Namun tidak semua orang memiliki kemampuan serta pengetahuan dalam pengoperasian *automated testing tools*, dan jenis *automated testing tools* yang berbeda-beda juga membutuhkan pemahaman dan cara pengoperasian yang berbeda pula.

Oleh karena itu, penulis membuat aplikasi yang dapat melakukan *vulnerability assessment* untuk aplikasi berbasis web

yang dalam tahap pengembangan ataupun sudah di *deploy* oleh pengembang aplikasi. Tujuan tugas akhir ini adalah untuk memberikan peringatan dini terhadap kerentanan yang ditemukan. Serta dalam pembuatan aplikasi ini menggunakan berbagai jenis *automated scanning tools* yang telah diintegrasikan kedalam satu aplikasi dan penggunaan metode *passive scanning* bertujuan agar selama proses *scanning* tidak mengganggu atau membebani kinerja dari target [1] serta metode ini lebih cepat dibandingkan metode *active scanning*.

Untuk menentukan tingkat kerentanan yang ditemukan, penulis menggunakan sistem penilaian *Common Vulnerability Scroring System* (*CVSS*) yang didalam sistem penilaian ini telah ada berbagai parameter untuk menentukan nilai suatu kerentanan [2]. Penulis juga menggunakan *OWASP TOP 10* untuk panduan dalam pembuatan beberapa skenario dalam proses *scanning*.

Hasil yang diharapkan dalam proses *scanning* ini adalah untuk menampilkan kerentanan yang ditemukan dan memberikan nilai berdasarakan kerentanan tersebut. Adapun metode yang digunakan diharapkan tidak akan menimbulkan beban kerja yang besar kepada target yang diuji selama proses *scanning*. Penggunaan *automated scanning tools* diharapkan dapat membantu dalam menemukan kerentanan yang umum ditemukan. Serta dengan aplikasi ini, pengguna tidak diharuskan memiliki keahlian khusus dalam pengoperasian *automated scanning tools* yang terdapat dalam aplikasi ini.

1.2 Rumusan Masalah

Tugas akhir ini mengangkat beberapa rumusan masalah sebagai berikut:

1. Bagaimana mendesain dan mengimplementasikan infrastruktur *vulnerability assessment and management* menggunakan *Docker Container*?
2. Bagaimana mengelola *multi-user* dalam proses *scanning*?
3. Bagaimana menemukan *vulnerability* pada situs web berdasarkan skenario yang telah dibuat?

4. Bagaimana penggunaan *resource* sistem dalam proses *vulnerability assessment* ?

1.3 Batasan Permasalahan

Permasalahan yang dibahas pada tugas akhir ini memiliki batasan sebagai berikut:

1. Menggunakan *automated scanning tools* yang telah ada seperti *Burp Scanner, Nmap,* dan *Shodan.*
2. Pembuatan skenario berdasarkan jenis *vulnerability* yang dibahas pada *OWASP Top 10*.
3. Web yang diuji memiliki domain its.ac.id.

1.4 Tujuan

Tujuan dari tugas akhir ini adalah sebagai berikut:

1. Mengimplementasikan infrastruktur *vulnerability assessment and management* menggunakan *Docker Container*.
2. Mengelola *multi-user* dalam proses *scanning.*
3. Menemukan *vulnerability* pada situs web berdasarkan skenario yang telah dibuat.
4. Menentukan penggunaan *resource* sistem dalam proses *vulnerability assessment*.

1.5 Manfaat

Dengan dibuatnya tugas akhir ini diharapkan dapat memberikan manfaat pada pihak pengelola dalam pemeliharaan serta pemeriksaan aplikasi web dengan memberikan laporan temuan kerentanan dan prioritas perbaikan.

1.6 Metodologi

Pembuatan tugas akhir ini dilakukan dengan menggunakan metodologi sebagai berikut :

1.6.1 Penyusunan Proposal

Tahap awal tugas akhir ini adalah menyusun proposal tugas akhir. Pada proposal, diajukan gagasan untuk menerapkan metode *passive scanning* dalam *vulnerability detection* dan penggunaan *CVSS* dalam penilaian kerentanan untuk menentukan prioritas perbaikan.

1.6.2 Studi Literatur

Pada tahap ini dilakukan untuk mencari informasi dan studi literatur apa saja yang dapat dijadikan referensi untuk membantu pengerjaan Tugas Akhir ini. Informasi didapatkan dari buku dan literatur yang berhubungan dengan *vulnerability assessment*, *passive scanning*, *CVSS*, dan *automated scanning tools*. Tugas akhir ini juga menggunakan *OWASP Top 10* sebagai panduan dalam pembuatan skenario pada proses *scanning* yang diterbitkan pada tahun 2017.

1.6.3 Analisis dan Desain Perangkat Lunak

Pada tahap ini akan dilakukan analisa, perancangan, dan pendefinisian kebutuhan sistem untuk mengetahui permasalahan yang akan dihadapi pada tahap implementasi. Kemudian akan dijabarkan kebutuhan – kebutuhan tersebut ke dalam perancangan fitur sistem.

1.6.4 Implementasi Perangkat Lunak

Implementasi merupakan tahap untuk membangun metode-metode yang sudah diajukan pada proposal Tugas Akhir. Implementasi dilakukan dengan menggunakan suatu perangkat lunak yaitu Visual Studio Code sebagai *Text Editor* dan *Docker Container*.

1.6.5 Pengujian dan Evaluasi

Pada tahap ini perangkat lunak yang telah dibangun diuji coba dengan menggunakan data uji coba yang ada. Data uji coba tersebut diuji coba pada perangkat lunak dengan tujuan mengetahui kemampuan metode yang digunakan untuk menemukan *vulnerability*. Data *vulnerability* akan dibandingkan dengan data yang telah ada pada *National Vulnerability Database*. Hasil evaluasi juga mencakup *running time* dari setiap proses *scanning*.

1.6.6 Penyusunan Buku

Pada tahap ini disusun buku sebagai dokumentasi dari pelaksanaan tugas akhir yang mencangkup seluruh konsep, teori, implementasi, serta hasil yang telah dikerjakan.

1.7 Sistematika Penulisan Laporan

Sistematika penulisan laporan tugas akhir adalah sebagai berikut:

1. Bab I. Pendahuluan
   Bab ini berisikan penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika penulisan dari pembuatan tugas akhir.
2. Bab II. Tinjauan Pustaka
   Bab ini berisi kajian teori dari metode dan algoritma yang digunakan dalam penyusunan Tugas Akhir ini. Secara garis besar, bab ini berisi tentang *vulnerability assessment, automated scanning tools, CVSS, OWASP Top 10,* bahasa pemrograman *python.*
3. Bab III. Perancangan Sistem
   Bab ini berisi mengenai implementasi dari arsitektur sistem yang akan diimplementasikan dalam pembuatan tugas akhir ini.
4. Bab IV. Implementasi

Bab ini menjelaskan implementasi dari arsitektur sistem yang dibuat sebelumnya. Penjelasan berupa kode program yang digunakan untuk implementasi arsitektur sistem.

5. Bab V. Hasil Uji Coba dan Evaluasi
   Bab ini membahas uji coba fungsionalitas sistem dengan cara melihat keluaran yang dihasilkan oleh sistem, analisis, dan evaluasi terhadap keberhasilan fungsi serta pengguaan sumber daya selama proses pengoperasian sistem.
6. Bab VI. Kesimpulan dan Saran
   Bab ini merupakan bab yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan dan evaluasi terhadap sistem, serta saran untuk pengembangan solusi ke depannya.
7. Daftar Pustaka
   Bab ini berisi daftar pustaka yang dijadikan literatur dalam tugas akhir.
8. Lampiran
   Dalam lampiran terdapat kode sumber program secara keseluruhan

BAB II
TINJAUAN PUSTAKA

Bab ini berisi pembahasan mengenai teori-teori dasar yang digunakan dalam tugas akhir. Teori-teori tersebut diantaranya adalah *OWASP Top 10*, metode *passive scanning*, *CVSS*, *Vulnerability Assesment*, *Automated Scanning Tools*, *Docker Container* dan beberapa teori lain yang mendukung pembuatan tugas akhir.

2.1 *OWASP Top 10*

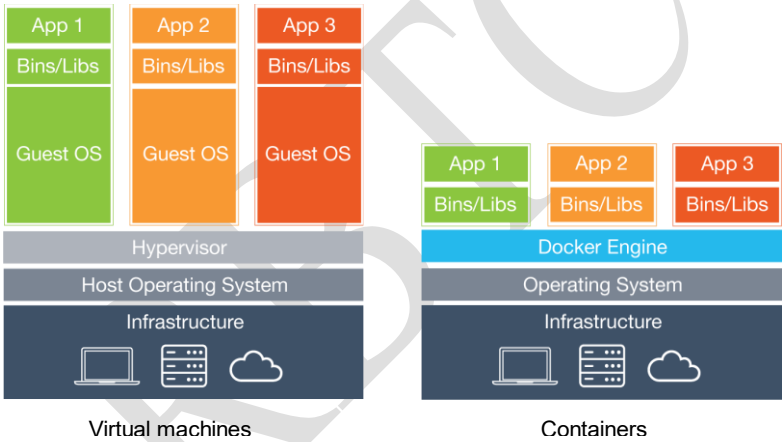
OWASP (*The Open Web Application Security Project*) merupakan organisasi atau komunitas *open source* yang berfokus dibidang keamanan aplikasi. *OWASP* memiliki beberapa proyek diantaranya *WebGoat*, *Webscarab*, dan *OWASP Top 10*. *OWASP Top 10* merupakan dokumen yang berisi rangkuman sepuluh kerentanan keamanan aplikasi yang ditemukan pada aplikasi web [3]. Dengan adanya dokumen ini diharapkan dapat membantu pengembang aplikasi dalam memahami sepuluh kerentanan keamanan serta dapat mencegah masalah ini pada aplikasinya. Versi terbaru dari *OWASP Top 10* adalah *OWASP Top 10-2017*.

2.2 *Common Vulnerability Scoring System* (*CVSS*)

CVSS adalah sebuah framework yang dirancang untuk menilai dan mengukur berbagai dampak kerentanan dalam sebuah perangkat lunak [4]. Sistem penilaian ini memiliki beberapa *metric* yang digunakan untuk mendapatkan nilai kuantitatif. Nilai kuantitatif yang diperoleh dari pengukuran dengan sistem ini akan dirubah dalam bentuk kualitatif (seperti *low*, *medium*, *high*, *critical*). Organisasi yang saat ini mengembangkan *CVSS* adalah *Cisco*, *Oracle*, *Qualys*, dan *Tenable Network Security*.

2.3 *Docker Container*

Container adalah unit standar perangkat lunak yang mengemas kode dan semua dependensinya sehingga aplikasi berjalan cepat dan andal dari suatu lingkungan komputasi ke yang lain [5]. *Image* dari *Docker Container* memiliki ukuran *file* yang jauh lebih kecil karena tidak perlu menyimpan sistem operasi secara penuh. Sehingga aplikasi yang berjalan menggunakan *Container* jauh lebih cepat dan efisien. Untuk melihat perbedaan antara *Container* dan *Virtual Machine* (*VM*) dapat dilihat pada Gambar 2.1.



Gambar 2.1 Perbandingan *Container* dan *Virtual Machine* [5]

Docker Container dibuat oleh sebuah *Docker Images*. *Docker Images* hanya mencakup dasar dari operasi sistem atau hanya memuat set dari *prebuild* aplikasi yang sudah siap dijalankan. Ketika membuat *Docker Images*, bisa menjalankan perintah (yaitu `apt-get install`) membentuk lapisan baru diatas lapisan sebelumnya. Perintah tersebut bisa dijalankan manual satu-persatu atau secara otomatis menggunakan *Dockerfile*.

Setiap *Dockerfile* adalah kombinasi beberapa perintah yang dibuat menjadi satu atau *script* yang bisa dijalankan secara otomatis sebagai *Docker Images* utama atau untuk membuat *Docker Images* yang baru. *Docker* juga menyediakan layanan untuk mengunduh dan mengupload *Docker Images* melalui `https://hub.docker.com/`.

2.4 *Passive Scanning*

Passive scanning adalah metode dari *vulnerability detection* yang memperoleh informasi dari arus data yang ditangkap tanpa perlu melakukan interaksi secara langsung. Metode ini memungkingkan interaksi yang seminimal mungkin sehingga tidak menggangu atau menghidupkan *alert* yang dipasang pada sistem yang diuji. Informasi yang didapatkan berupa informasi mengenai sistem operasi yang digunakan, *port* yang menjalankan *service* dan aplikasi yang berjalan pada sistem. Metode ini memiliki keterbatasan pada proses deteksi aplikasi yang akan mengakibatkan *false information*. Hal ini dikarenakan sifat dari *passive scanning* yang menangkap informasi dari arus data, sehingga aplikasi yang tidak melakukan pertukaran data tidak dapat dideteksi.

2.5 *Vulnerabiltiy Assessment*

Vulnerability assessment merupakan rangkaian proses untuk mengidentifikasi kerentanan yang terdapat pada infrastruktur [6]. Proses ini dapat dilakukan menggunakan *automated scanning tools* sepersi *Nessus*, menggunakan *set of plugin* yang sesuai dengan jenis kerentanan yang ingin diperiksa. Proses ini juga dapat dilakukan secara pendekatan manual menggunakan daftar kerentanan yang dibuat berdasarkan situs daftar kerentanan yang telah diketahui. Pendekatan manual tidak begitu komprehensif dibandingakn menggunakan pendekatan dengan *tools*, hal ini dikarenakan proses ini memakan waktu yang lebih lama

dibandingkan *automated scanning tools*. Untuk melihat perbedaan antara *vulnerability assessments* dan *penetration testing* dapat dilihap pada Tabel 2.1.

Tabel 2.1 Perbandingan *vulnerability assessments* dan *penetration testing* [7]

	Vulnerability Assessment	Penetration Testing
Attributes	List-oriented	Goal-oriented
Types of Reports	Prioritised list of vulnerabilities categorised by criticality for remediation	Specific information of what data was compromised and vulnerabilities exploited
Purpose	Identify security vulnerabilities in system that may be exploited	Determine whether an application can withstand an intrusion attempt

2.6 *Shodan*

Shodan adalah mesin pencari untuk perangkat yang terhubung dengan internet. *Shodan* mengumpulkan informasi tentang semua perangkat lunak yang terhubung langsung ke internet [8]. Jika suatu perangkat terhubung dengan internet, maka *shodan* akan mencari informasi yang umum didapatkan dari perangkat tersebut. Jenis perangkat yang diindeks sangat bervariasi, dari *CCTV* yang menggunakan *ip address* hingga pembangkit listrik tenaga nuklir. *Indeks* yang digunakan *shodan* berasal dari metadata yang didapatkan dari *banners* perangkat lunak tersebut. Informasi yang didapatkan dari metadata ini dapat berupa informasi tentang jenis *server* apa yang digunakan, *welcome message* atau apa saja yang didapatkan *client* sebelum berinteraksi.

2.7 *Network Mapper* (*NMAP*)

Nmap adalah sebuah aplikasi *open source* untuk kegiatan *network discovery* dan audit keamanan [9]. *Nmap* menggunakan paket *IP* mentah untuk menemukan *hosts* yang ada pada suatu jaringan, *service* apa saja yang sedang berjalan, sistem operasi yang digunakan, *firewalls* yang sedang digunakan, dan lain sebagainya. Aplikasi ini didesain untuk dapat digunakan pada jaringan yang besar ataupun hanya untuk satu *host*, selain itu *Nmap* dapat berjalan pada semua sistem operasi yang ada seperti *Windows*, *Linux*, dan *Mac OS X*. *Nmap* umumnya dijalankan menggunakan *command-line*, namun terdapat juga versi *nmap* yang memiliki *graphical user interface*.

2.8 *Python*

Python adalah bahasa pemrograman tingkat tinggi yang didukung oleh struktur data *build-in* sematik dinamis, selain itu *python* mendukung pemograman *procedural*, *object-oriented* dan *functional*. *Python* merupakan bahasa pemograman *interpreted*, oleh sebab itu, *python* tidak memakan biaya untuk kompilasi, sehingga proses pengembangan, pengujian dan *debug* menjadi lebih cepat.

Kelebihan bahasa pemograman ini adalah memiliki modul dan *package*, serta memiliki banyak standar pustaka yang didistribusikan secara bebas dan gratis. Selain itu *python* mudah dibaca karena memiliki sintaksis yang sederhana, sehingga dapat mengurangi biaya *maintenance*. *Debug* pada *python* juga mudah karena tidak akan terjadi *segmentation fault*, namun akan memberi umpan balik berupa *exception* apabila terdapat kesalahan atau *error* [10].

(Halaman ini sengaja dikosongkan)

BAB III
PERANCANGAN SISTEM

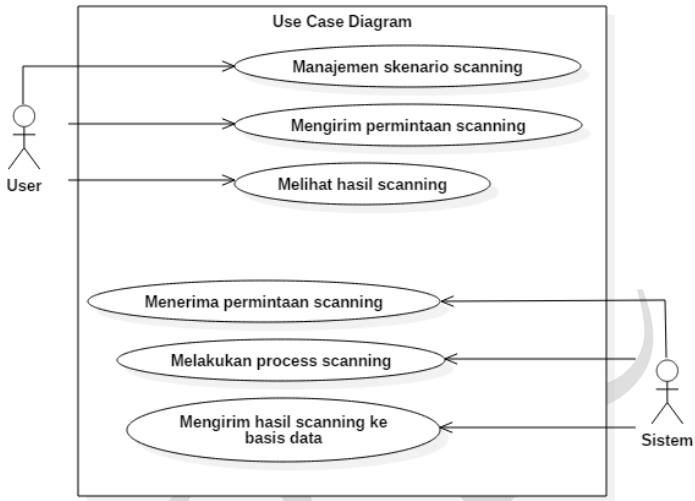
Bab ini membahas mengenai perancangan dan pembuatan sistem. Rancang bangun aplikasi *vulnerability assessment and management* yang dibuat pada tugas akhir ini adalah menentukan kerentanan dan nilainya dengan metode *passive scanning* serta menggunakan *cvss* sebagai indikator penilaian.

3.1 Deskripsi Umum Sistem

Sistem yang akan dibangun pada tugas akhir ini adalah sistem yang dapat melakukan *vulnerability assessment* terhadap suatu situs web. Uji coba pada sistem ini akan menggunakan beberapa web dengan domain its.ac.id menggunakan metode *passive scanning* dan *cvss*. Hasil yang didapatkan nanti akan ditampilkan dalam bentuk *workspace* yang berisi nilai *cvss*, ringkasan dari kerentanan, dan refrensi untuk mempelajari lebih lanjut kerentanan tersebut. Sistem ini juga menggunakan *Docker* sebagai infrastruktur, sehingga lebih mudah dalam melakukan *deployment*.

Proses uji coba akan diproses ketika pengguna melakukan proses *scanning* pada *command line interface* yang telah disediakan sistem. Kemudian *core logic* pada sistem akan memanggil *automation scanner* yang telah tersedia pada sistem menggunakan *subprocess*. Setelah proses *scanning* selesai, sistem akan menampilkan hasilnya pada *workspace* dan menyimpannya pada *database*. Pengguna dapat melihat hasil tersebut melalui *user interface* yang telah disediakan. Untuk proses *scanning* yang berkala, pengguna dapat menggunakan fitur *scheduling* yang tersedia pada sistem.

3.2 Kasus Penggunaan



Gambar 3.1 Diagram kasus penggunaan

Terdapat dua aktor dalam sistem yang akan dibuat yaitu *User* dan Sistem. *User* merupakan actor (pengguna) yang bisa melakukan manajemen pada skenario yang akan digunakan dalam proses *scanning* dan dapat melihat hasilnya, sedangkan Sistem merupakan aktor yang akan digunakan sebagai *framework* untuk melakukan proses *vulnerability scanning*. Diagram kasus penggunaan digambarkan pada Gambar 3.1 dan dijelaskan masing-masing pada Tabel 3.1

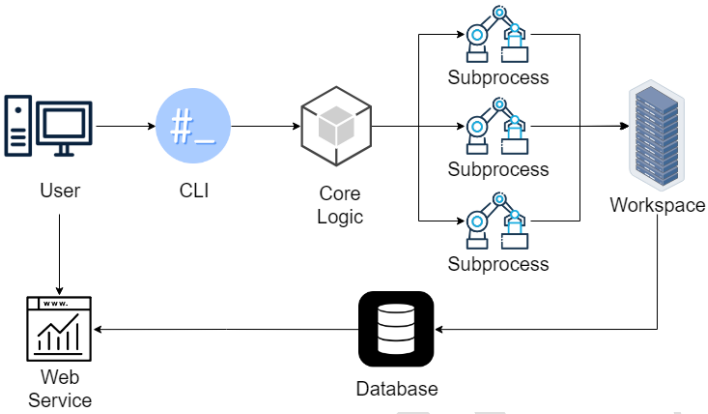
Tabel 3.1 Daftar kode kasus penggunaan

Kode Kasus Penggunaan	Nama Kasus Penggunaan	Keterangan
UC-0001	Manajemen skenario *scanning.*	*User* memilih skenario yang telah tersedia

		untuk digunakan dalam proses *scanning*
UC-0002	Mengirim permintaan *scanning*.	*User* dapat mengirim permintaan *scanning* ke Sistem
UC-0003	Melihat hasil *scanning*.	Ketika proses *scanning* selesai, *user* dapat melihat hasilnya pada *workspace* yang telah tersedia.
UC-0004	Menerima permintaan *scanning*.	Proses dimana Sistem akan menerima permintaan *scanning* dari *User*.
UC-0005	Melakukan proses *scanning*.	Proses dimana Sistem akan melakukan proses *scanning* sesuai skenario yang dikirim.
UC-0006	Mengirim hasil *scanning* ke basis data.	Ketika sistem telah selesai melakukan proses *scanning*, data yang didapatkan akan dikirim ke basis data *SQLite*.

3.3 Arsitektur Sistem

Pada sub-bab ini, akan dibahas mengenai tahap analisis arsitektur, analisis teknologi dan desain sistem yang akan dibangun. Arsitektur sistem secara umum ditunjukan pada Gambar 3.2.



Gambar 3.2 Desain arsitektur sistem

3.3.1 Desain Umum Sistem

Berdasarkan yang dijelaskan pada deskripsi umum sistem, dapat diperoleh beberapa kebutuhan sistem antara lain:

1. *Web service* sebagai antarmuka pengguna.
2. *Core logic* sebagai pengatur *subprocess* yang dijalankan.
3. *Subprocess* untuk menjalankan proses *scanning*.
4. *Workspace* untuk menggabungkan hasil dari proses *scanning*.
5. *Database* untuk menyimpan data sistem.
6. *Task queue* untuk menangani kasus *request* lebih dari satu proses *scanning*.

Untuk memenuhi kebutuhan sistem yang dijelaskan sebelumnya, penulis membagi menjadi beberapa komponen sistem yang akan digunakan pada tugas akhir ini.

1. *Service Controller*
   Berfungsi sebagai pengatur sistem *scanning* yang terdiri :
   - *Web Service*

Berfungsi sebagai tampilan antarmuka pengguna untuk menggunakan sistem.

- *Workspace*
  Berfungsi untuk menggabungkan data yang diperoleh dari *subprocess* untuk nantinya dimasukan ke *database*.
- *Database*
  Berfungsi untuk menyimpan segala data yang diperoleh oleh sistem.
- *Task Queue*
  Berfungsi untuk membuat antrian dalam menangani pembuatan proses *scanning* untuk menghindari pengguaan *resource* berlebih.

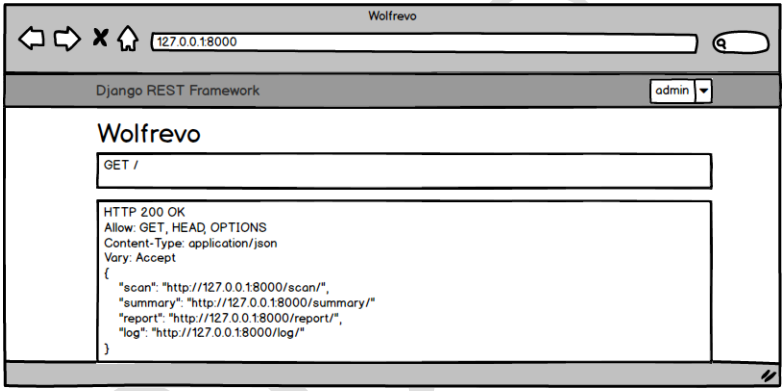
2. *Core Logic*
   Berfungsi untuk mengatur dan memilih *subprocess* yang dijalankan oleh sistem berdasarkan skenario yang dipilih oleh *user*.

3.3.2 Perancangan *Service Controller*

Komponen ini akan digunakan untuk mengatur proses sebelum dan sesudah *scanning* pada sistem. Pada komponen ini akan terdapat 4 jenis sub-komponen yaitu *web service*, *workspace*, *database*, dan *task queue* atau antrian.

3.3.2.1 Desain *Web Service*

Web service akan berfungsi sebagai antarmuka *user* dan sebagai penghubung antara *user* dengan *core logic*. Antarmuka ini berfungsi memudahkkan pengguna untuk menentukan skenario yang digunakan dalam proses *scanning* dan *subprocess* yang sesuai dengan skenario akan menjalankan *tools* yang sesuai pada proses *scanning* tersebut. Desain antarmuka dapat ditunjukkan pada Gambar 3.3



Gambar 3.3 Desain antarmuka *user*

3.3.2.2 Desain *Workspace*

Workspace akan berfungsi untuk menggabungkan hasil dari setiap *subprocess* yang telah dijalankan untuk satu proses *scanning*. Hal ini dilakukan agar data yang dimasukan kedalam *database* untuk satu proses lebih mudah dan tidak tercampur dengan proses *scanning* yang berjalan setelahnya. Hasil yang terdapat pada *workspace* berbeda tergantung dari skenario yang dipilih pada proses *scanning*.

3.3.2.3 Desain *Database*

Database diperlukan untuk menyimpan data yang tersedia pada *workspace* yang diperoleh dari proses *scanning*. Data yang tersimpan adalah data *username*, data nama *container*, data target, data *task queue*, dan data *vulnerability*. Dari data-data tersebut dibutuhkan *tabel*, diantaranya yaitu:

- *Tabel Scan*
  Menyimpan *username*, nama *container*, skenario yang dipilih, dan target.
- *Tabel Summary*
  Menyimpan *username*, nama *container*, teknologi yang digunakan, dan *vulnerability*.
- *Tabel Report*
  Menyimpan nama *container*, target, dan lokasi *file* pada *workspace*
- *Tabel Logs*
  Menyimpan nama *container*, perintah yang dijalankan, dan waktu.
- *Tabel Task Queue*
  Menyimpan nama *container*, status, dan waktu.

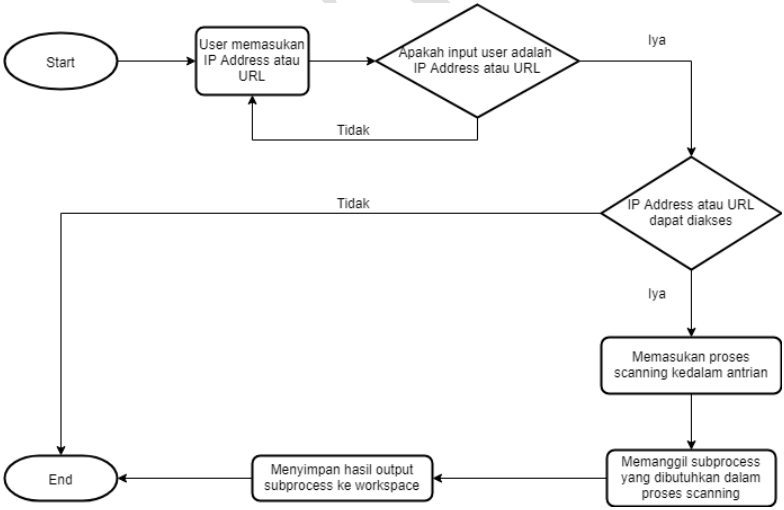
3.3.2.4 Desain *Task Queue*

Terdapat kemungkinan *user* melakukan lebih dari satu proses *scanning*, dimana untuk satu proses *scanning* memakan waktu cukup lama. Proses *scanning* akan membuat satu *container* tersendiri dalam menjalankan prosesnya, sehingga didalam sistem akan berjalan beberapa *container* secara bersamaan. Jika proses ini tidak dibuatkan antrian maka setiap *request* untuk melakukan proses *scanning* akan dijalankan langsung tanpa memperhatikan apakah proses *scanning* sebelumnya telah selesai. Hal ini akan mengakibatkan *container* yang menjalankan proses *scanning* akan terus bertambah dan akan menguras *resource*, sehingga akan ada

kondisi dimana sistem kehabisan *resource*. Untuk mengantisipasi hal ini, akan dirancang sebuah program dalam bentuk *python* atau bisa disebut *task queue*. *Task queue* akan berjalan sebagai program *daemon* dan hasilnya disimpan dalam basis data *SQLite*.

3.3.3 Perancangan *Core Logic*

Komponen ini akan digunakan untuk mengatur ketika proses *scanning* pada sistem berjalan. Hal yang pertama dilakukan setelah *user* memasukan *url* atau *ip address* dari *target* adalah mengecek apakah *user* telah memasukan *input* secara benar, apakah *input user* tersebut dapat diakses, memasukan proses *scanning* kedalam antrian, dan skenario apa saja yang diinginkan oleh *user* selama proses *scanning* tersebut berjalan. Diagram alir proses dari *core logic* dapat dilihat pada Gambar 3.4.



Gambar 3.4 Diagram alir *core logic*

BAB IV
IMPLEMENTASI

Bab ini membahas mengenai implementasi dari sistem yang sudah dirancang pada bab sebelumnya. Pembahasan secara rinci akan dijelaskan pada setiap komponen yang ada yaitu *core logic* dan *service controller* yang meliputi *web service*, *workspace*, *database*, dan *task queue*.

4.1 Lingkungan Implementasi

Lingkungan implementasi dan pengembangan dilakukan didalam komputer dengan spesifikasi *processor* Intel Core i5-7400 CPU @ 3.00Ghz dan *memory* 8 GB. Perangkat lunak yang digunakan dalam pengembangan dipaparkan pada Tabel 4.1.

Tabel 4.1 Perangkat Lunak yang Digunakan

No	Perangkat Lunak	Versi	Keterangan
1	Linux Ubuntu	18.04	Sistem Operasi
2	Visual Studio Code	1.41.0	Text Editor
3	Python	3.6.9	Bahasa Pemograman
4	Git	2.17.1	Pengolahan Versi Program
5	Docker	18.09.7	Virtualisasi Tingkat OS
6	Django	2.2.0	*Python web framework*
7	Django Rest Framework	3.11.0	*Web API* untuk *Django*

4.2 Implementasi Dasar Sistem

Berdasarkan perancangan dan desain pada bab sebelumnya, keseluruhan sistem akan diletakan pada satu kontainer. Hal ini dilakukan untuk mempermudah dalam tahap *deployment* dan dapat diperbanyak sesuai keperluan selanjutnya. Pembuatan dasar sistem menggunakan *Docker*, namun diperlukan beberapa tahap untuk bisa menggunakan *Docker* atau kontainer sebagai dasar sistem, yaitu tahap pemasangan dan konfigurasi. Tahap pemasangan *Docker* melalui *default repository* dapat dilihat di Kode Sumber 8.1 atau melalui *official repository* pada Kode Sumber 8.2, dan untuk menjalankan *Docker* tanpa menggunakan *user root* dapat dilihat pada Kode Sumber 8.3. Untuk aplikasi dan *tools* yang diperlukan dalam menjalankan sistem dapat ditunjukkan pada Kode Sumber 8.4. Sedangkan untuk konfigurasi akan dijelaskan pada implementasi pembuatan *Docker Image*.

4.3 Implementasi Pembuatan *Docker Image*

Docker Image diperlukan dalam pembuatan kontainer yang akan digunakan sebagai dasar sistem pada tugas akhir ini. Untuk membuat *Docker Image* diperlukan beberapa tahapan yaitu konfigurasi `Dockerfile`, dan konfigurasi untuk unggah *Docker Image* ke *Docker Hub*. Konfigurasi `Dockerfile` dapat dilihat pada Kode Sumber 4.1, kemudian jalankan perintah *docker build* untuk membuat *Docker Image* dari `Dockerfile` pada Kode Sumber 4.2. Setelah *Docker Image* terbuat, lakukan *commit* agar dapat di *push* ke *Docker Hub* seperti pada Kode Sumber 4.3.

```
1  FROM ubuntu:18.04
2  ENV LC_ALL=C.UTF-8 \
3      LANG=C.UTF-8
4  RUN mkdir /root/wolfrevo
5  WORKDIR /root/wolfrevo
6  # Update and upgrade
7  RUN apt-get update && apt-get upgrade -y
```

```
8   # Install requirement tools
9   RUN apt-get install --quiet --yes --fix-
10  missing \
        software-properties-common \
11      whois \
12      dnsutils \
13      python3 \
14      python3-pip \
15      npm \
16      nmap \
17      curl \
18      wget \
19      masscan \
20      golang \
21      chromium-browser \
22      git && \
23      git clone
24  https://github.com/satriaaryawan/wolfrevo.
    git . && \
        chmod +x install.sh && \
25      ./install.sh && \
26      git clone
27  https://github.com/scipag/vulscan
    scipag_vulscan && \
        ln -s `pwd`/scipag_vulscan
28  /usr/share/nmap/scripts/vulscan && \
        apt-get autoremove --purge -y && \
29      apt-get clean
```

Kode Sumber 4.1 Konfigurasi *Dockerfile*

```
1   docker build -t tugas_akhir .
```

Kode Sumber 4.2 Perintah untuk membuat *Docker Image*

```
1   docker tag testing:latest
    wolfrevo/tugas_akhir:latest
2   docker push wolfrevo/tugas_akhir:latest
```

Kode Sumber 4.3 Perintah untuk mengunggah *Docker Image*

4.4 Implementasi *Core Logic*

Core logic memiliki peran untuk mengatur *subprocess* yang akan dijalankan dalam proses *scanning*. Selain itu dalam *core logic* juga akan ada pengecekan sebelum dan sesudah proses *scanning*. *Core logic* akan dijalankan pada masing-masing *container* yang menjalankan proses *scanning*. Sehingga untuk setiap proses *scanning* akan memiliki *core logic* tersendiri yang akan mengatur proses *scanning* itu sendiri. Isi berkas untuk *core logic* dapat ditunjukkan pada Kode Sumber 8.5.

4.5 Implementasi *Service Controller*

Berdasarkan desain dan perancangan, *service controller* terdiri dari komponen *web service*, *workspace*, *database*, dan *task queue*. Keseluruhan kompenen ini akan diimplementasikan untuk satu buah server.

4.5.1 Implementasi *Web Service*

Untuk menggunakan fitur-fitur dalam sistem yang dibuat pada *web service*, pengguna perlu mengakses rute-rute yang dijelaskan pada Tabel 4.2.

Tabel 4.2 Rute Fitur Pada *Web Service*

No	Rute	Method	Keterangan
1	/	POST	Login sebagai user administrator.
2	/scan	GET	Menampilkan halaman *scan*.
3	/summary	GET	Menampilkan halaman *summary*.

4	/reports	GET	Menampilkan halaman *reports*.
5	/logs	GET	Menampilkan halaman *logs*.

4.5.2 Implementasi *Workspace*

Workspace adalah tempat meletakan seluruh hasil keluaran dari *tools* yang dijalankan oleh *subprocess*. Penyimpannya dalam berbagai format *file* seperti dalam bentuk *".txt"*, *"json"* dan *".xml"* dalam *folder* yang telah diatur agar dapat diproses lebih lanjut. Setiap proses *scanning* yang dijalankan pada *container* memiliki *workspace* tersendiri yang nantinya keseluruhan *file* yang terdapat didalamnya akan dikirim ke *workspace* pada *server*. Hal ini dilakukan karena *workspace* pada *container* akan hilang saat proses *scanning* selesai, sehingga diperlukan *workspace* utama untuk menyimpan *file* agar tidak hilang ketika *container* dari proses *scanning* dihapus.

4.5.3 Implementasi *Database*

Berdasarkan hasil perancangan *database* pada bab sebelumnya. Data yang dibutuhkan dan digunakan oleh sistem di dalam basis data *SQLite*. Penggunaan *SQLite* dipilih karena penggunaan *Django REST Framework*, sehingga dapat memudahkan dalam memasukan data kedalam *database*. Lebih jelasnya dapat dilihat pada Kode Sumber 4.4

```
1  from django.db import models
2
3  class Scan(models.Model):
4      username =
   models.TextField(default='')
5      nama_container =
   models.TextField(default='')
```

```
6       skenario =
    models.TextField(default='')
7       target = models.TextField(default='')
8
9   class Summary(models.Model):
10      username =
    models.TextField(default='')
11      nama_container =
    models.TextField(default='')
12      teknologi =
    models.TextField(default='')
13      vulnerability =
    models.TextField(default='')
14
15  class Report(models.Model):
16      nama_container =
    models.TextField(default='')
17      target = models.TextField(default='')
18      lokasi_file =
    models.TextField(default='')
19
20  class Logs(models.Model):
21      nama_container =
    models.TextField(default='')
22      cmd = models.TextField(default='')
23      waktu = models.TextField(default='')
24
25  class Taskqueue(models.Model):
26      nama_container =
    models.TextField(default='')
27      status = models.TextField(default='')
28      waktu = models.TextField(default='')
```

Kode Sumber 4.4 Implementasi *database* pada *Django REST Framework*

4.5.4 Implementasi *Task Queue*

Task Queue akan digunakan untuk mengatur antrian permintaan proses *scanning* dari pengguna, pada tugas akhir ini implementasi *task queue* akan dipasang pada *server* yang sama dengan *web service*. Untuk membuat *task queue* penulis menggunakan bahasa pemrograman *python* versi 3.6.9. Sedangkan untuk basis data menggunakan *SQLite* yang terpasang pada *server* yang sama dengan *task queue*. Selama *task queue* berjalan, algoritmanya akan selalu mengecek apakah terdapat antrian dalam permintaan *scanning* yang dapat dieksekusi. Isi berkas untuk *task queue* dapat ditunjukkan pada Kode Sumber 4.5.

```
1  import json
2  import subprocess
3  import sys
4  import time
5  import requests
6
7  from datetime import datetime
8
9  create_container = []
10 start_container = []
11 die_container = []
12 destroy_container = []
13
14 def start_docker(docker_id):
15     subprocess.call(['docker', 'start',
   docker_id])
16
17 def remove_docker(docker_id):
18     subprocess.call(['docker', 'rm'
   ,docker_id])
19
20 def docker_run(container_status,
   container_id):
21     if container_status == 'create':
```

```
22  create_container.append(container_id)
23      # start container
24      while len(start_container) < 5:
25          try:
26              run_container =
    create_container.pop(0)
27              start_docker(run_container)

28  start_container.append(run_container)
29          except:
30              break
31      # die container
32      if container_status == 'die':
33          die_container.append(container_id)
34          for i in die_container:
35              if i in start_container:
36                  start_container.remove(i)
37      # destroy container
38      while len(die_container) > 0:
39          try:
40              exited_container =
    die_container.pop(0)

41  remove_docker(exited_container)

42  destroy_container.append(exited_container)
43          except:
44              break
45
46  def run():
47      cmd = "docker events --format '{{json
    .}}' --filter 'image=tugas_akhir'"
48      program = subprocess.Popen(cmd,
    shell=True, stdout=subprocess.PIPE,
    stderr=subprocess.STDOUT)
49      while True:
50          nextline =
    program.stdout.readline().decode('utf-8')
```

```
51          json_data = json.loads(nextline)
52          if nextline == '' and
    program.poll() is not None:
53              break
54          event_detail = {'nama_container':
    None, 'status': None, 'waktu': None}
55          event_detail['status'] =
    json_data['status']
56          event_detail['nama_container'] =
    json_data['Actor']['Attributes']['name']
57          event_detail['waktu'] =
    json_data['time']
58          docker_run(event_detail['status'],
    event_detail['nama_container'])

59  requests.post('http://127.0.0.1:8000/TaskQ
    ueue/', json=(event_detail))
60          print(json.dumps(event_detail))
61
62  if __name__ == "__main__":
63      run()
```

Kode Sumber 4.5 Isi berkas *docker_events.py*

(Halaman ini sengaja dikosongkan)

BAB V
HASIL UJI COBA DAN EVALUASI

Pada bab ini akan dibahas uji coba dan evaluasi dari sistem yang sudah dibuat. Sistem akan diuji coba fungsionalitasnya dengan menjalankan skenario pengujian fitur-fitur yang terdapat pada sistem. Uji coba dilakukan untuk mengevaluasi kinerja sistem dengan lingkungan uji coba yang ditentukan.

5.1 Lingkungan Uji Coba

Lingkungan uji coba sistem ini adalah sebuah *server* yang didalamnya terdapat komponen *core logic*, *web service*, dan *database*. Sedangkan untuk proses *scanning* akan berjalan menggunakan *Docker Container* yang dibuat dari *Docker Images*, sehingga jumlah *container* yang dibuat berjumlah keseluruhan proses *scanning* yang dapat berjalan berdasarkan *task queue*. *Server* dan *Docker Images* yang digunakan sistem menggunakan sebuah pc dengan spesifikasi untuk komponen yang digunakan dapat dilihat pada Tabel 5.1.

Tabel 5.1 Spesifikasi Komponen

No	Komponen	Perangkat Keras	Perangkat Lunak
1	*Server*	*Processor* Intel Core i5-7400 CPU @ 3.00GHz, *Memory* 8GB.	Ubuntu 18.04, Python 3.6.9, Git 2.17, Docker 19.03.5., Django 2.2.0, Django Rest Framework 3.11.0.
2	*Docker Images*	*Size* 2.49GB	Python 3.6.9, Shodan 1.20, Nmap 7.80, Subfinder 2.3.0, Go 13.6, Amass 3.4.2.

5.2 Skenario Uji Coba

Uji coba ini dilakukan untuk menguji fungsionalitas dari sistem yang telah dibuat berdasarkan perancangan dan sistem benar-benar diimplementasikan dan bekerja sesuai seharusnya. Skenario pengujian dibedakan menjadi 2 bagian yaitu:

- **Uji Fungsionalitas**
  Pengujian yang dilakukan didasarkan pada fungsionalitas yang disajikan sistem.
- **Uji Performa**
  Pengujian ini dilakukan untuk mengetahui ketersediaan *CPU* dan *RAM* yang digunakan oleh sistem selama proses *scanning* dilakukan.

5.2.1 Skenario Uji Fungsionalitas

Uji fungsionalitas pada sistem ini akan menguji beberapa bagian yang terdapat pada *core logic* yaitu *user* melihat proses *scanning, user* melihat *summary* proses *scanning, user* melihat *reports* proses *scanning*, dan *user* melihat *logs* proses *scanning*. Untuk lebih detailnya dapat melihat Tabel 5.2.

Tabel 5.2 Skenario uji fungsionalitas

No	Rute	Uji Coba	Harapan
1	/scan	Mengirim *request* menuju rute *web service* melalui *browser*.	*Request* berhasil diterima oleh *web service*, kemudian *web service* menampilkan halaman *scan.*
2	/summary	Mengirim *request* menuju rute *web*	*Request* berhasil diterima oleh *web service*, kemudian

		service melalui *browser*.	*web service* menampilkan halaman *summary*.
3	/reports	Mengirim *request* menuju rute *web service* melalui *browser*.	*Request* berhasil diterima oleh *web service*, kemudian *web service* menampilkan halaman *reports*.
4	/logs	Mengirim *request* menuju rute *web service* melalui *browser*.	*Request* berhasil diterima oleh *web service*, kemudian *web service* menampilkan halaman *logs*.

5.2.2 Skenario Uji Performa

Uji performa digunakan untuk mengetahui besar penggunaan *memory* dan *CPU* pada saat sistem berjalan. Skenario uji coba dilakukan dengan cara menjalankan proses *scanning* terhadap *IP address* atau *URL* yang berbeda dalam satu waktu, kemudian mengevaluasi penggunaan *memory* dan *CPU*. Skenario uji coba performa adalah sebagai berikut:

- Menjalankan proses *scanning* untuk satu *ip address* atau *url* pada satu waktu.
- Menjalankan proses *scanning* untuk 10 *ip address* atau *url* pada satu waktu.
- Menjalankan proses *scanning* untuk 20 *ip address* atau *url* pada satu waktu.
- Menjalankan proses *scanning* untuk 30 *ip address* atau *url* pada satu waktu.
- Melakukan *stress test* pada sistem dengan menjalankan 50 proses *scanning*.

5.3 Hasil Uji Coba dan Evaluasi

Berikut dijelaskan hasil uji coba dan evaluasi berdasarkan skenario yang sudah dijelaskan pada bab 5.2.

5.3.1 Uji Fungsionalitas

Berikut dijelaskan hasil pengujian fungsionalitas pada sistem yang sudah dibangun. Uji coba ini dilakukan dengan mengakses sistem melalui rute yang telah ditentukan pada Tabel 5.2. Detail hasil uji coba fungsionalitas pada sistem dapat dilihat pada Tabel 5.3.

Tabel 5.3 Hasil uji fungsionalitas

No	Rute	Harapan	Hasil
1	/scan	*Request* berhasil diterima oleh *web service*, kemudian *web service* menampilkan halaman *scanning.*	OK
2	/summary	*Request* berhasil diterima oleh *web service*, kemudian *web service* menampilkan halaman *summary.*	OK
3	/reports	*Request* berhasil diterima oleh *web service*, kemudian *web service* menampilkan halaman *reports.*	OK
4	/logs	*Request* berhasil diterima oleh *web service*, kemudian *web*	OK

		service menampilkan halaman *logs*.	

5.3.2 Uji Performa

Seperti yang sudah dijelaskan pada bab 5.2.2, uji coba penggunaan sumber daya dilakukan secara bertahap dengan menggunakan jumlah proses *scanning* yang berbeda-beda dan dengan target yang sama. Pengujian ini akan membandingkan penggunaan *memory* dan *CPU* serta waktu untuk masing-masing skenario. Terdapat lima kondisi berbeda yang digunakan dalam proses uji coba sumber daya.

Pengujian ini dilakukan dengan memanfaatkan *file python* untuk membantu menjalankan proses *scanning* dengan target yang memiliki subdomain its.ac.id. Pemilihan target sendiri memanfaatkan informasi yang dapat diakses melalui `http://pantau-web.its.ac.id/` dan untuk menampilkan datanya menggunakan *Grafana* serta *Prometheus*.

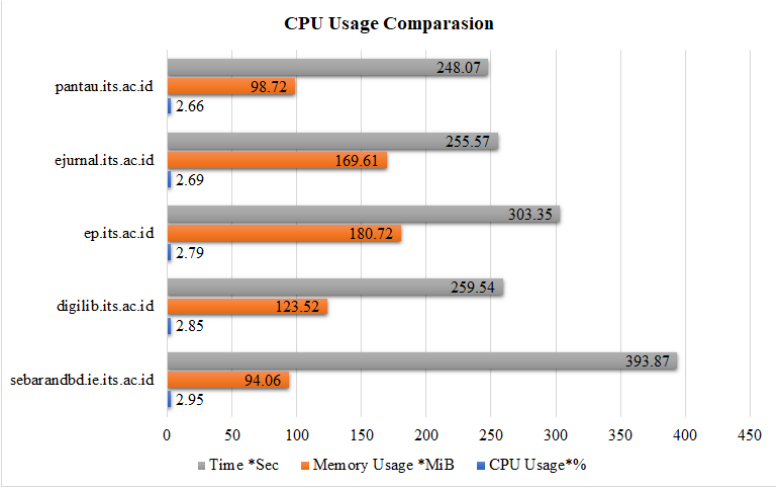
Kondisi pertama adalah ketika menjalankan satu proses *scanning* dalam satu waktu yang setiap prosesnya menggunakan *container* tersendiri. Hasil uji coba pada kondisi ini tertera pada Tabel 5.4. Pada Tabel 5.4 ditampilkan penggunaan *CPU*, *memory*, dan total waktu dari proses *scanning* dari 100 target.

Tabel 5.4 Hasil uji coba satu proses *scanning* pada satu waktu

No	Container Id	Target	Container CPU Usage *%			Container Memory Usage *MiB			Time *Sec
			min	max	avg	min	max	avg	
1	administrator-d105bce4	humas.its.ac.id	0.79	6.11	2.21	21.66	336.56	167.29	341.17
2	administrator-d105bce5	amu.its.ac.id	0.23	2.76	1.27	21.91	344.98	130.33	235.98
3	administrator-d291b1c6	fasor.its.ac.id	0.79	4.99	2.09	21.87	325.91	122.03	198.22
4	administrator-d3353198	enviro.its.ac.id	0.58	4.75	1.14	15.98	383.57	105.31	892.21
5	administrator-d3353199	tlingkungan.enviro.its.ac.id	0	3.52	1	15.79	127.97	66.33	962.71
96	satria-e1c748e0	digilib.its.ac.id	0.14	6.7	2.85	16.98	413.76	123.52	259.54
97	satria-e1c748e1	share.its.ac.id	0.21	4.77	1.94	24.24	329.62	131.05	237.33
98	satria-e336dbe6	boyo.its.ac.id	0.52	2.69	1.54	21.94	130.34	96.09	206.37
99	satria-e40fdb30	geofisika.its.ac.id	0.5	7.98	2.12	22.15	137.76	88.87	391.01
100	satria-e40fdb31	marineicon.ne.its.ac.id	0.56	3.22	1.63	22.05	355.59	126.83	258.16

Untuk mempermudah membaca hasil dari Tabel 5.4, dapat melihat Gambar 5.1 dan Gambar 5.2 yang melakukan

perbandingan hubungan antara *CPU*, *memory* dan waktu untuk lima proses *scanning* yang menggunakan *CPU* dan *memory* terbesar. Gambar 5.1 menampilkan perbandingan untuk lima proses *scanning* yang memerlukan *CPU* dalam jumlah yang besar, sedangkan untuk Gambar 5.2 menampilkan perbandingan untuk lima proses scanning berdasarkan penggunaan *memory* terbesar.



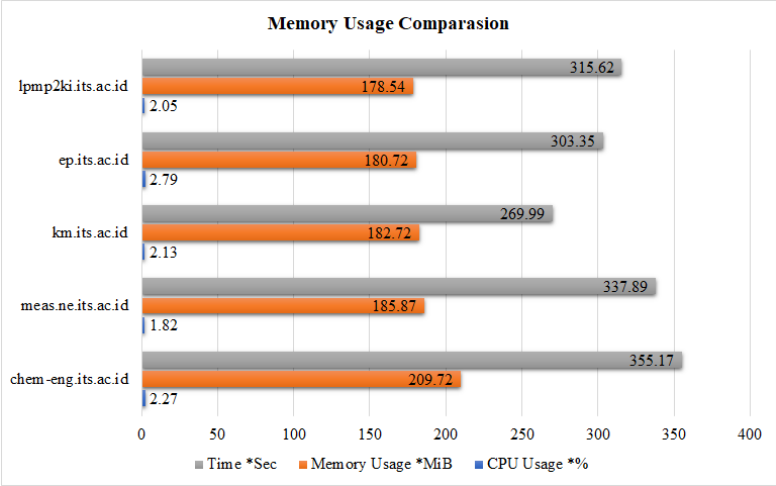
Gambar 5.1 Penggunaan *CPU* terbesar untuk 5 proses *scanning* pada skenario 1

Pada skenario pertama setiap proses *scanning* akan berjalan sendiri dan *resource* yang ada pada sistem akan digunakan untuk proses itu sendiri. Pada skenario awal atau skenario 0 *CPU load* pada sistem hanya 1,73% dengan *memory load* mencapai 23% serta penggunaan *memory* pada keadaan awal mencapai 449,75 MiB. Ketika skenario pertama dijalankan *CPU load* naik menjadi 3,16% dengan penggunaan *memory load* mencapai 40% serta peggunaan *memory* mencapai 1,04 GiB. Penggunaan *memory* yang mencapai 1,04 GiB disebabkan karena ketika proses *scanning* berjalan, terdapat beberapa *automated scanning tools* yang melakukan operasi pada *memory* sehingga proses untuk melepaskan memory

ketika proses *scanning* selesai tetap berjalan walaupun proses scanning telah berakhir. Perbandingan pengguaan *CPU* dan *memory* dapat ditunjukkan pada Tabel 5.5.

Tabel 5.5 Penggunaan *CPU* dan *memory* pada skenario 1

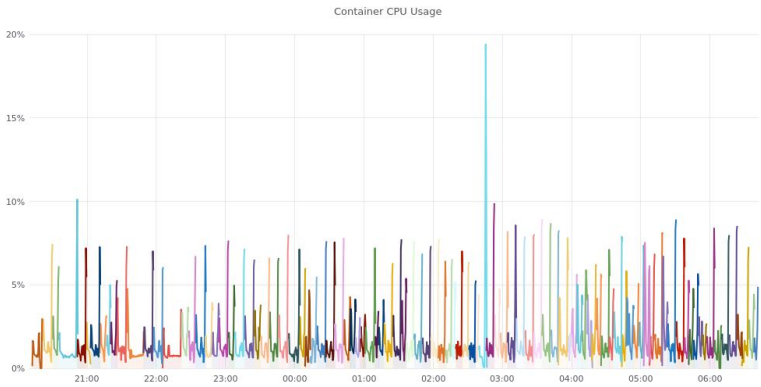
Skenario	CPU Load	Memory Load	Used Memory
0	1,73%	23%	449,75 MiB
1	3,16%	40%	1,04 GiB



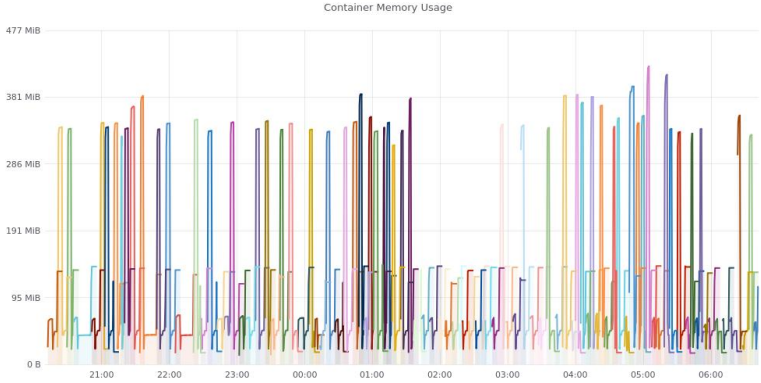
Gambar 5.2 Penggunaan *memory* terbesar untuk 5 proses *scanning* pada skenario 1

Penggunaan *CPU* pada sistem selama kondisi pertama ini berjalan dapat dilihat pada Gambar 5.3 dan penggunaan *memory* pada Gambar 5.4. Pada Gambar 5.3 dapat dilihat bahwa penggunaan *CPU* pada proses *scanning* hanya berjalan sendiri tanpa ada penggunaan *CPU* untuk proses lain. Sehingga pada skenario pertama penggunaan *CPU load* hanya mencapai 3,16% atau 60,03% digunakan oleh proses *scanning*. Sedangkan pada Gambar 5.4 terlihat bahwa penggunaan *memory* tetap berjalan walaupun proses *scanning* telah selesai, sehingga menyebabkan

penggunaan *memory* pada sistem mencapai 1,04 GiB atau 40% dari *memory* yang terdapat pada sistem. Pada skenario pertama dibutuhkan waktu 6 jam untuk melakukan proses *scanning* terhadap 100 target, dengan rata-rata waktu yang diperlukan untuk melakukan satu proses *scanning* mencapai 375 detik. Dengan skenario ini, pengguna harus menunggu 6 menit untuk dapat menjalankan proses *scanning* yang baru.



Gambar 5.3 Penggunaan *CPU* sistem pada skenario 1



Gambar 5.4 Penggunaan *memory* sistem pada skenario 1

Kondisi kedua adalah ketika menjalankan 10 proses *scanning* dalam satu waktu yang setiap prosesnya menggunakan *container* tersendiri. Hasil uji coba pada kondisi ini tertera pada Tabel 5.6. Pada Tabel 5.6 ditampilkan penggunaan *CPU*, *memory*, dan total waktu dari proses *scanning* dari 100 target.

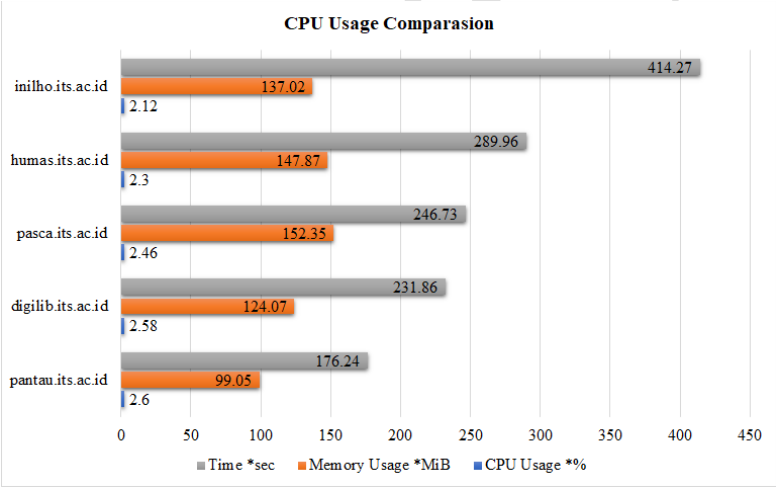
Tabel 5.6 Hasil uji coba 10 proses *scanning* pada satu waktu

No	Container Id	Target	Container CPU Usage *%			Container Memory Usage *MiB			Time *Sec
			min	max	avg	min	max	avg	
1	administrator-00771454	humas.its.ac.id	0.59	7.79	2.3	21.82	323.93	147.87	289.96
2	administrator-019f508a	amu.its.ac.id	0	2.43	1.03	21.9	344.43	102.12	194.22
3	administrator-031626be	fasor.its.ac.id	0.04	4.64	1.86	22.09	329.48	131.49	193.26
4	administrator-03d6ebd8	enviro.its.ac.id	0.11	4.32	0.93	22.05	382.5	134.75	1035.98
5	administrator-053aec9a	tlingkungan.enviro.its.ac.id	0.34	2.7	0.89	15.55	127.69	71.14	555.53
96	satria-1b0b62a3	boyo.its.ac.id	0.11	2.26	1.08	12.07	62.45	46.81	242.38
97	satria-1c34ed92	geofisika.its.ac.id	0.34	7.78	2.01	11.36	140.86	88.99	336.82
98	satria-1c34ed93	marineicon.ne.its.ac.id	0.4	3.17	1.42	18.73	349.77	126.07	294.43
99	satria-ffcd2804	pilrek.its.ac.id	0	3.56	1.18	21.59	127.71	85.61	358.9
100	satria-ffcd2805	hima.na.its.ac.id	0.32	8.21	1.94	21.79	136.13	89.61	348.69

Untuk mempermudah membaca hasil dari Tabel 5.6, dapat melihat Gambar 5.5 dan Gambar 5.6 yang melakukan perbandingan hubungan antara *CPU*, *memory* dan waktu untuk lima proses *scanning* yang menggunakan *CPU* dan *memory* terbesar. Gambar 5.5 menampilkan perbandingan untuk lima proses *scanning* yang memerlukan *CPU* dalam jumlah yang besar, sedangkan untuk Gambar 5.6 menampilkan perbandingan untuk lima proses berdasarkan penggunaan *memory* terbesar.

Pada skenario kedua terdapat 10 proses *scanning* yang berjalan secara bersamaan pada satu waktu. Penggunaan *CPU* dan *memory* untuk setiap proses *scanning* tidak mempengaruhi penggunaan *CPU* ataupun *memory* pada proses *scanning* yang lain. Pada skenario ini penggunaan *CPU load* mencapai 13,79% dengan penggunaan *memory load* mencapai 44% serta *memory* yang digunakan 2,40 GiB. Penggunaan *resource* pada skenario dua, lebih tinggi dibandingkan pada skenario pertama karena pada skenario pertama hanya terdapat satu proses *scanning* yang berjalan pada satu waktu. Sedangkan pada skenario kedua terdapat 10 proses scanning yang berjalan pada satu waktu. Perbandingan penggunaan *CPU* dan *memory* dapat dilihat pada Tabel 5.7.

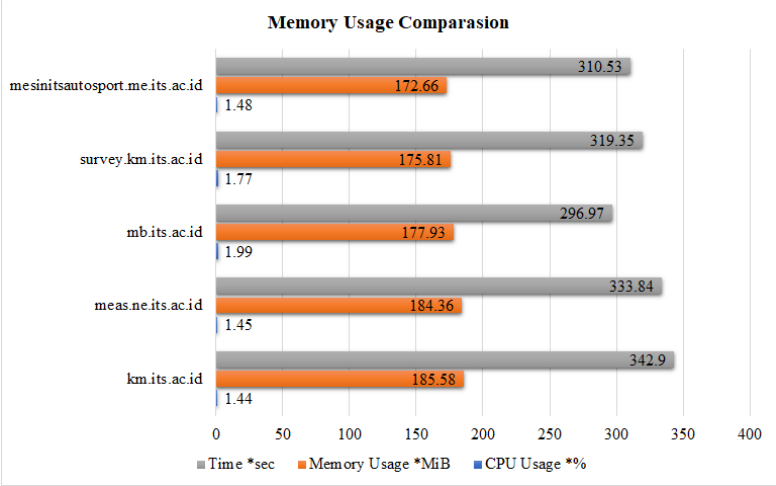
CPU load yang digunakan pada proses *scanning* memiliki nilai yang sama dengan skenario pertama. Pada skenario ini *CPU load* untuk setiap proses *scanning* sama dengan skenario pertama, namun presentase penggunaan *CPU load* untuk setiap proses *scanning* mengalami penurunan dibandingkan skenario pertama yang awalnya 60.03% menjadi 12.14%. Hal ini dikarenakan terdapat 10 proses *scanning* yang berjalan pada satu waktu. Dengan skenario ini, pengguna dapat menyelesaikan 100 proses *scanning* dalam waktu 1 jam dengan waktu untuk satu proses *scanning* selesai adalah 363 detik. Skenario ini memungkinkan dalam waktu 6 menit mendapatkan 10 hasil proses *scanning*.



Gambar 5.5 Penggunaan *CPU* terbesar untuk 5 proses *scanning* pada skenario 2

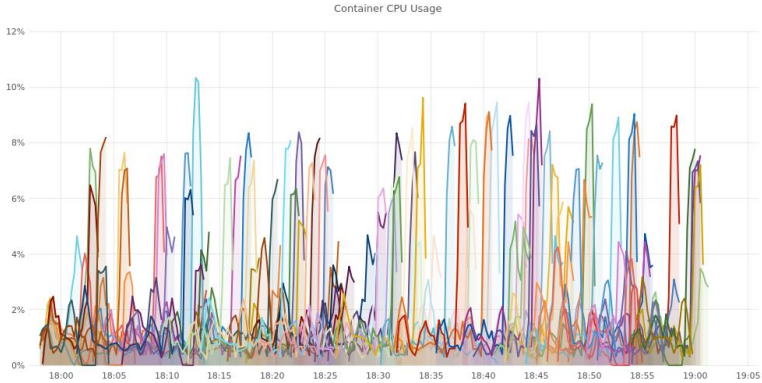
Tabel 5.7 Penggunaan *CPU* dan *memory* pada skenario 2

Skenario	CPU Load	Memory Load	Used Memory
0	1,73%	23%	449,75 MiB
1	3,16%	40%	1,04 GiB
2	13,79%	44%	2,40 GiB

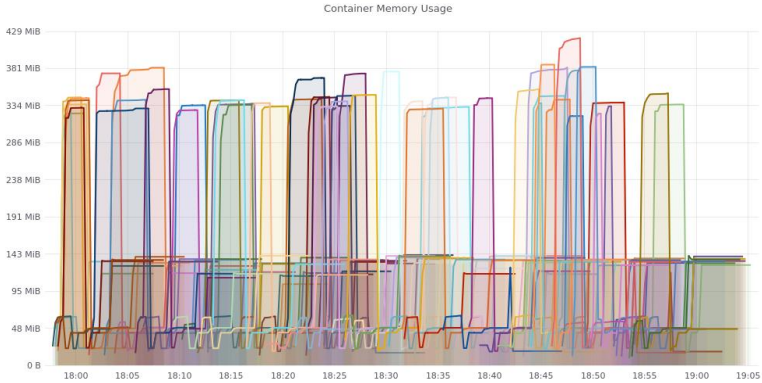


Gambar 5.6 Penggunaan *memory* terbesar untuk 5 proses *scanning* pada skenario 2

Penggunaan *CPU* pada sistem selama kondisi kedua ini berjalan dapat dilihat pada Gambar 5.7 dan penggunaan *memory* pada Gambar 5.8. Pada Gambar 5.7 dapat dilihat bahwa terjadi *CPU load* antara satu proses *scanning* dengan proses *scanning* lainya berjalan bersamaan, hal ini mengakibatkan penggunaan *CPU* pada sistem naik menjadi 13,79% atau naik 10% dibandingkan dengan *CPU load* pada skenario pertama. Dengan kenaikan 10% ini, proses *scanning* dapat berjalan untuk 10 target dalam satu waktu. Sedangkan penggunaan *memory* naik menjadi 2.40 GiB dibandingkan dengan penggunaan *memory* pada skenario pertama. Kenaikan penggunaan *memory* disebabkan karena ketika proses *scanning* berjalan terdapat proses *scanning* lain yang ikut berjalan dan dibutuhkan waktu untuk melepaskan *memory* yang digunakan seperti kondisi pada skenario pertama. Pada Gambar 5.8 diperlihatkan terdapat irisan penggunaan memory antara satu proses *scanning* dengan proses *scanning* yang lain. Walaupun demikian irisan dari penggunaan *memory* ini hanya mempengaruhi penggunaan dari *memory* pada sistem.



Gambar 5.7 Penggunaan *CPU* sistem pada skenario 2



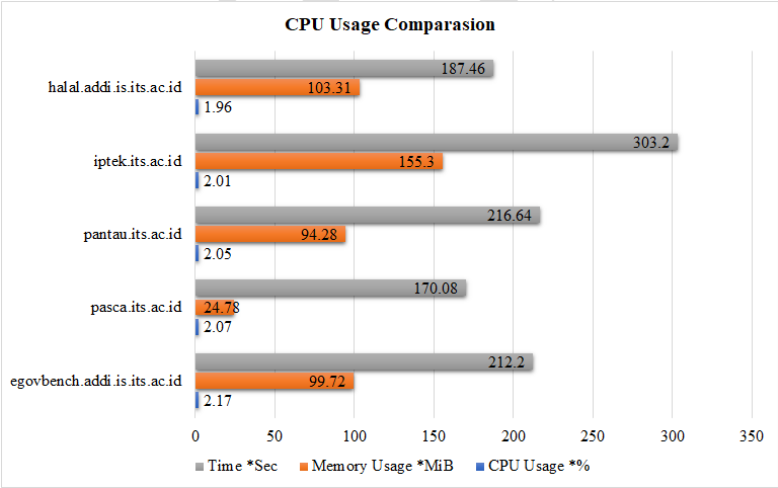
Gambar 5.8 Penggunaan *memory* sistem pada skenario 2

Kondisi ketiga adalah ketika menjalankan 20 proses *scanning* dalam satu waktu yang setiap prosesnya menggunakan *container* tersendiri. Hasil uji coba pada kondisi ini tertera pada Tabel 5.8. Pada Tabel 5.8 ditampilkan penggunaan *CPU*, *memory*, dan total waktu dari proses *scanning* dari 100 target.

Tabel 5.8 Hasil uji coba 20 proses *scanning* pada satu waktu

No	Container Id	Target	Container CPU Usage *%			Container Memory Usage *MiB			Time *Sec
			min	max	avg	min	max	avg	
1	administrator-4b5dff40	humas.its.ac.id	0.04	7.58	1.57	23.38	341.59	110.26	345.14
2	administrator-4c645fd8	amu.its.ac.id	0.02	1.58	0.44	10.73	316.97	77.15	235.25
3	administrator-4d908206	fasor.its.ac.id	0	4.49	1.14	11.3	307.39	68.42	304.35
4	administrator-4e9eb82a	enviro.its.ac.id	0.03	3.53	0.77	4.44	337.8	53.74	863.57
5	administrator-505359a0	tlingkungan.enviro.its.ac.id	0	3.76	0.76	9.84	212.5	35.22	866.7
96	satria-6fbc1a48	digilib.its.ac.id	0.01	5.81	1.81	24.18	399.97	201.37	320.67
97	satria-728c0882	share.its.ac.id	0	3.89	1.4	15.67	133.8	99.52	206.49
98	satria-753a580e	boyo.its.ac.id	0.26	2.85	1.19	19.56	129.81	100.58	160.78
99	satria-77f491f4	geofisika.its.ac.id	0.09	7.83	1.46	23.71	138.54	85.92	417.49
100	satria-7abdbb7c	marineicon.ne.its.ac.id	0.4	2.28	0.84	22.04	352.41	156.6	404.99

Untuk mempermudah membaca hasil dari Tabel 5.8, dapat melihat Gambar 5.9 dan Gambar 5.10 yang melakukan perbandingan hubungan antara *CPU*, *memory* dan waktu untuk lima proses *scanning* yang menggunakan *CPU* dan *memory* terbesar. Gambar 5.9 menampilkan perbandingan untuk lima proses *scanning* yang memerlukan *CPU* dalam jumlah yang besar, sedangkan untuk Gambar 5.10 menampilkan perbandingan untuk lima proses *scanning* berdasarkan penggunaan *memory* terbesar.



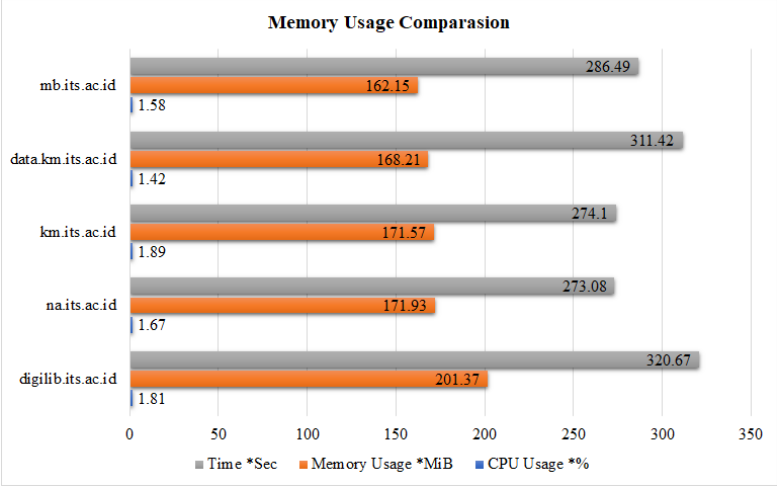
Gambar 5.9 Penggunaan *CPU* terbesar untuk 5 proses *scanning* pada skenario 3

Pada skenario ketiga terdapat 20 proses *scanning* yang berjalan secara bersamaan pada satu waktu. Pada skenario ini waktu yang dibutuhkan untuk menyelesaikan 100 proses *scanning* 12 kali lebih cepat dibandingkan dengan skenario pertama dan 2 kali lebih cepat dibandingkan dengan skenario kedua dengan perbandingan penggunaan *CPU* dan *memory* yang dapat dilihat pada Tabel 5.9. Pada skenario ini *CPU load* mencapai 21,18% dan *memory load* mencapai 47% dengan penggunaan *memory* 2,94 GiB. Kenaikan *CPU load* mencapai 7,39% dibandingkan dengan pada skenario kedua. Sedangkan untuk *memory load* pada skenario ketiga mengalami kenaikan dibandingkan dengan skenario pertama yang mencapai 3%. Hal ini disebabkan setiap proses *scanning* memerlukan *memory* yang tetap walaupun pada waktu itu terdapat proses *scanning* lain yang sedang berjalan.

Pada skenario ketiga terdapat penurunan penggunaan *CPU load* untuk setiap proses *scanning*. Penurunan ini terjadi pada jumlah *CPU load* maupun presentase penggunaan *CPU load* dibandingkan dengan sistem. Jumlah *CPU load* menurun dibandingkan dengan skenario pertama dan skenario kedua, serta presentase penggunaan *CPU load* pada skenario ini menjadi 6,94% dibandingkan *CPU load* pada sistem. Hal ini dikarenakan pada skenario ketiga terdapat 20 proses *scanning* yang berjalan dan untuk setiap proses *scanning* memerlukan *CPU load* yang harus dibagi dengan proses *scanning* yang lain.

Tabel 5.9 Penggunaan *CPU* dan *memory* pada skenario 3

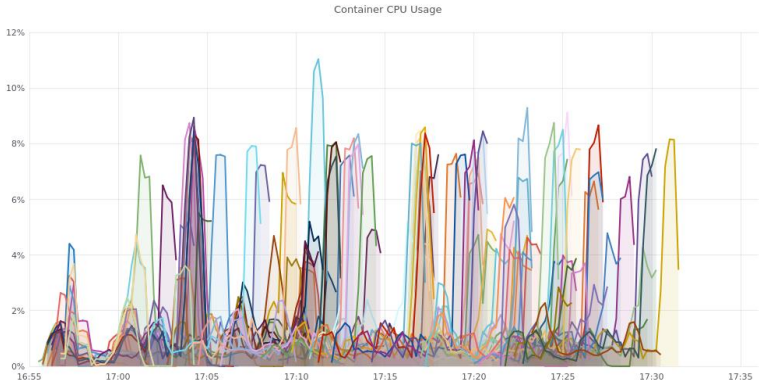
Skenario	CPU Load	Memory Load	Used Memory
0	1,73%	23%	449,75 MiB
1	3,16%	40%	1,04 GiB
2	13,79%	44%	2,40 GiB
3	21,18%	47%	2,94 GiB



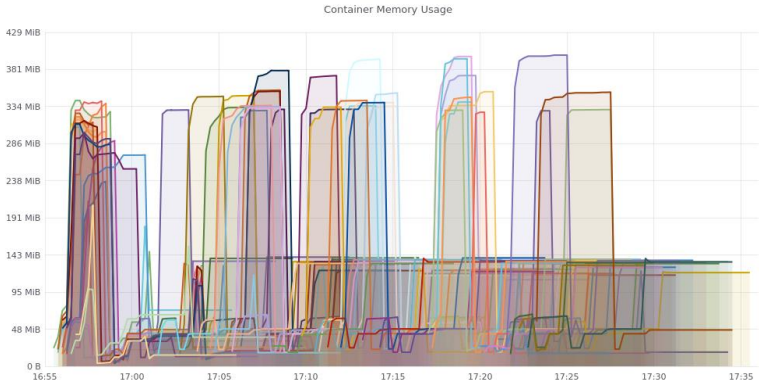
Gambar 5.10 Penggunaan *memory* terbesar untuk 5 proses *scanning* pada skenario 3

Penggunaan *CPU* pada sistem selama kondisi ketiga ini berjalan dapat dilihat pada Gambar 5.11 dan penggunaan *memory* pada Gambar 5.12. Pada Gambar 5.11 dapat dilihat bahwa terjadi *CPU load* antara satu proses *scanning* dengan proses *scanning* lainya berjalan bersamaan, hal ini mengakibatkan penggunaan *CPU* pada sistem naik menjadi 21.18% atau naik 7,39% dibandingkan dengan *CPU load* pada skenario kedua dan naik 18,02% dibandingkan dengan skenario pertama. Dengan kenaikan 18,02% ini, proses *scanning* dapat berjalan untuk 20 target dalam satu waktu. Sedangkan penggunaan *memory* naik menjadi 2.94 GiB dibandingkan dengan penggunaan *memory* pada skenario pertama. Kenaikan penggunaan *memory* disebabkan karena ketika proses *scanning* berjalan terdapat proses *scanning* lain yang ikut berjalan dan dibutuhkan waktu untuk melepaskan *memory* yang digunakan seperti kondisi pada skenario pertama. Pada Gambar 5.12 diperlihatkan terdapat irisan penggunaan memory antara satu proses *scanning* dengan proses *scanning* yang lain. Walaupun

demikian irisan dari penggunaan *memory* ini hanya mempengaruhi penggunaan dari *memory* pada sistem. Kenaikan penggunaan *memory* pada skenario ini mencapai 3% dibandingkan dengan skenario kedua dan 7% dibandingkan skenario pertama. Penggunaan *memory* yang mencapai 47% ini merupakan batas aman yang dapat digunakan dalam tahap *deployment* pada sistem dengan spesifikasi yang telah disebutkan pada bab 4.1.



Gambar 5.11 Penggunaan *CPU* sistem pada skenario 3



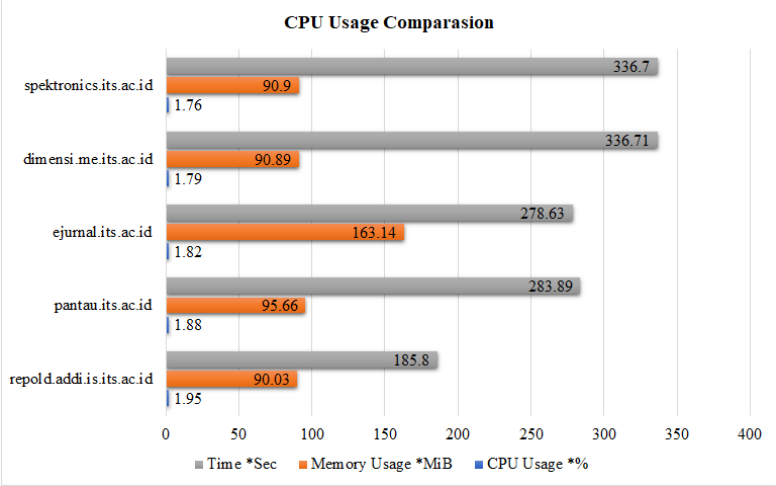
Gambar 5.12 Penggunaan *memory* sistem pada skenario 3

Kondisi keempat adalah ketika menjalankan 30 proses *scanning* dalam satu waktu yang setiap prosesnya menggunakan *container* tersendiri. Hasil uji coba pada kondisi ini tertera pada Tabel 5.10. Pada Tabel 5.10 ditampilkan penggunaan *CPU*, *memory*, dan total waktu dari proses *scanning* dari 100 target.

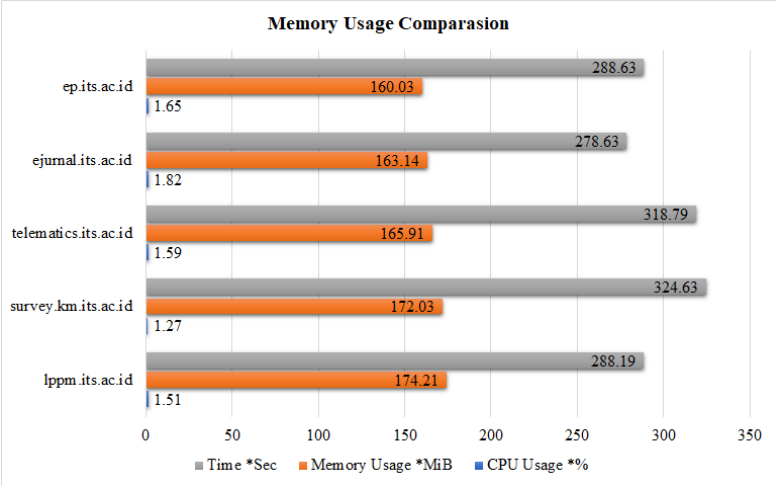
Tabel 5.10 Hasil uji coba 30 proses *scanning* pada satu waktu

No	Container Id	Target	Container CPU Usage *%			Container Memory Usage *MiB			Time *Sec
			min	max	avg	min	max	avg	
1	administrator-62f39b34	humas.its.ac.id	0.01	7.82	1.51	5.37	338.94	97.3	394
2	administrator-62f39b35	amu.its.ac.id	0.01	2.06	0.43	7.8	343.09	86.58	277.1
3	administrator-6441c880	fasor.its.ac.id	0	3.89	0.89	13.93	166.52	100.48	362.7
4	administrator-64fc545c	enviro.its.ac.id	0	3.51	0.63	11.41	330.3	56.46	748.76
5	administrator-64fc545d	tlingkungan.enviro.its.ac.id	0	3.53	0.64	10.29	63.4	31.44	707.01
96	satria-856f2eb2	digilib.its.ac.id	0.16	3.38	1.15	15.34	132.75	96.14	162.87
97	satria-88a1fad8	share.its.ac.id	0.09	4.32	1.44	22.06	133.94	97.62	177.58
98	satria-8b88f4d6	boyo.its.ac.id	0.1	3.19	1.19	22.36	131.69	98.45	159.45
99	satria-8e64dc24	geofisika.its.ac.id	0.19	7.18	1.55	22.84	137	87.92	339.45
100	satria-909e13b6	marineicon.ne.its.ac.id	0.18	1.28	0.58	22.27	41.74	37.94	177.67

Untuk mempermudah membaca hasil dari Tabel 5.10, dapat melihat Gambar 5.13 dan Gambar 5.14 yang melakukan perbandingan hubungan antara *CPU*, *memory* dan waktu untuk lima proses *scanning* yang menggunakan *CPU* dan *memory* terbesar. Pada skenario keempat terdapat 30 proses *scanning* yang berjalan secara bersamaan pada satu waktu. Pada skenario ini waktu yang dibutuhkan untuk menyelesaikan 100 proses *scanning* 15 kali lebih cepat dibandingkan dengan skenario pertama dan 2,5 kali lebih cepat dibandingkan dengan skenario kedua serta 1,25 kali lebih cepat dibandingkan dengan skenario ketiga dengan perbandingan penggunaan *CPU* dan *memory* yang dapat dilihat pada Tabel 5.11.



Gambar 5.13 Penggunaan *CPU* terbesar untuk 5 proses *scanning* pada skenario 4



Gambar 5.14 Penggunaan *memory* terbesar untuk 5 proses *scanning* pada skenario 4

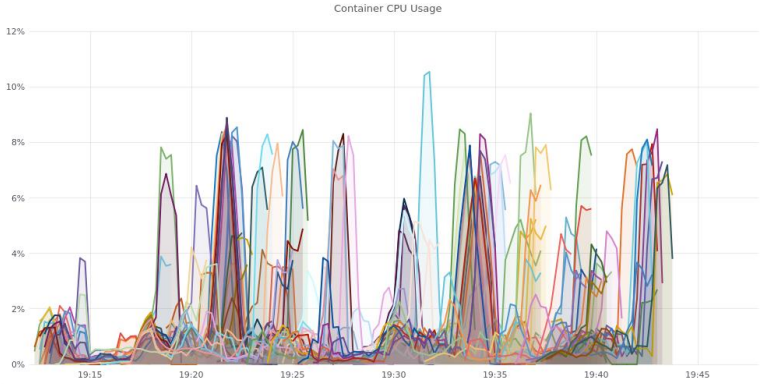
Penggunaan *CPU* dan *memory* untuk setiap proses *scanning* tidak mempengaruhi penggunaan *CPU* ataupun *memory* pada proses *scanning* yang lain. Pada skenario ini penggunaan *CPU load* mencapai 23,77% dengan penggunaan *memory load* mencapai 55% serta *memory* yang digunakan 3,27 GiB. Penggunaan *resource* pada skenario keempat lebih tinggi dibandingkan skenario pertama, kedua, dan ketiga karena skenario keempat terdapat 30 proses scanning yang berjalan pada satu waktu. Perbandingan penggunaan *CPU* dan *memory* dapat dilihat pada Tabel 5.11. Pada skenario ini *CPU load* untuk setiap proses *scanning* mengalami penurunan dibandingkan skenario pertama, kedua dan ketiga baik dalam presentase penggunaan *CPU load* maupun nilai *CPU load* untuk setiap proses *scanning*. Setiap proses *scanning* mengalami penurunan dibandingkan skenario pertama yang memiliki nilai 60,03%, skenario kedua dengan nilai 12,14% dan skenario ketiga dengan nilai 6,94% menjadi 5,22% pada skenario keempat. Hal ini dikarenakan penggunaan *CPU load* pada skenario keempat digunakan oleh 30 proses *scanning*. Namun pada skenario keempat penggunaan *memory* juga tidak mengalami perubahan, sehingga setiap proses *scanning* menggunakan *memory* tanpa ada gangguan dari proses *scanning* yang lain.

Tabel 5.11 Penggunaan *CPU* dan *memory* pada skenario 4

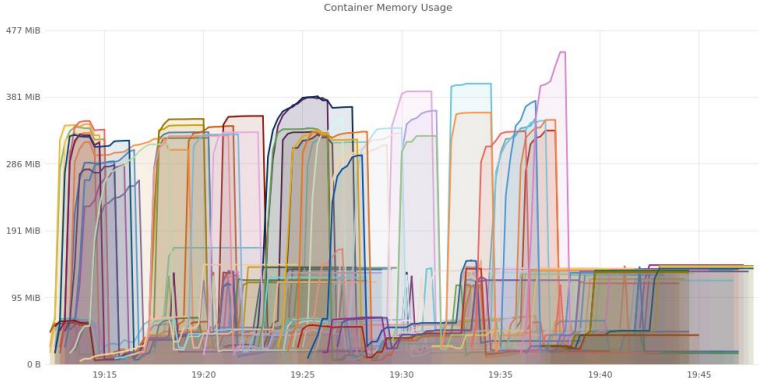
Skenario	CPU Load	Memory Load	Used Memory
0	1,73%	23%	449,75 MiB
1	3,16%	40%	1,04 GiB
2	13,79%	44%	2,40 GiB
3	21,18%	47%	2,94 GiB
4	23,77%	55%	3,27 GiB

Penggunaan *CPU* pada sistem selama kondisi keempat ini berjalan dapat dilihat pada Gambar 5.15 dan penggunaan *memory* pada Gambar 5.16 Pada Gambar 5.15 dapat dilihat bahwa terjadi *CPU load* antara satu proses *scanning* dengan proses *scanning* lainya berjalan bersamaan, hal ini mengakibatkan penggunaan

CPU pada sistem naik menjadi 23.77% atau naik 2,59% dibandingkan dengan *CPU load* pada skenario ketiga dan naik 19,98% dibandingkan dengan skenario kedua, serta naik 20,61% dibandingkan dengan *CPU load* pada skenario pertama. Dengan kenaikan 20,61% ini, proses *scanning* dapat berjalan untuk 30 target dalam satu waktu. Sedangkan penggunaan *memory* naik menjadi 3,27 GiB dibandingkan dengan penggunaan *memory* pada skenario pertama. Kenaikan penggunaan *memory* disebabkan karena ketika proses *scanning* berjalan terdapat proses *scanning* lain yang ikut berjalan dan dibutuhkan waktu untuk melepaskan *memory* yang digunakan seperti kondisi pada skenario pertama. Pada Gambar 5.16 diperlihatkan terdapat irisan penggunaan memory antara satu proses *scanning* dengan proses *scanning* yang lain. Walaupun demikian irisan dari penggunaan *memory* ini hanya mempengaruhi penggunaan dari *memory* pada sistem. Kenaikan penggunaan *memory* pada skenario ini mencapai 8% dibandingkan dengan skenario ketiga dan 11% dibandingkan skenario kedua serta 15% dibandingkan skenario pertama. Penggunaan *memory* yang mencapai 55% ini merupakan batas kurang aman yang dapat digunakan dalam tahap *deployment* pada sistem dengan spesifikasi yang telah disebutkan pada bab 4.1. Hal ini dikarenakan pada saat menjalankan proses keempat terdapat *delay* dalam penggunaan sistem.



Gambar 5.15 Penggunaan *CPU* sistem pada skenario 4



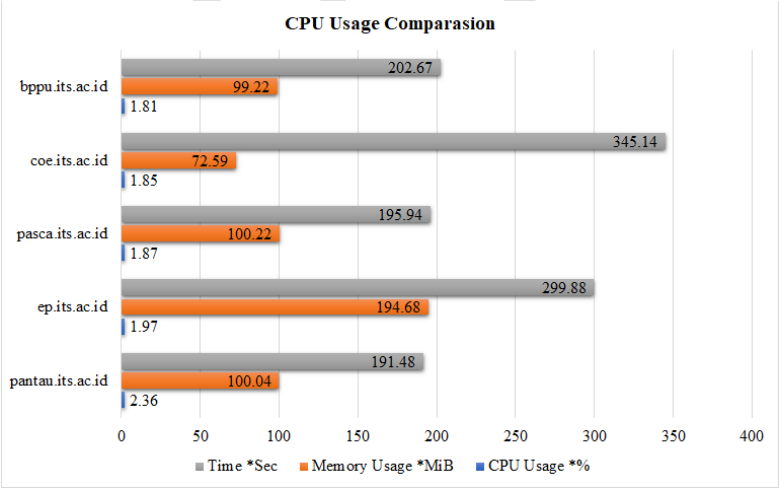
Gambar 5.16 Penggunaan *memory* sistem pada skenario 4

Kondisi *strees test* adalah ketika menjalankan 50 proses *scanning* dalam satu waktu yang setiap prosesnya menggunakan *container* tersendiri. Hasil uji coba pada kondisi ini tertera pada Tabel 5.12. Pada Tabel 5.12 ditampilkan penggunaan *CPU*, *memory*, dan total waktu dari proses *scanning* dari 100 target.

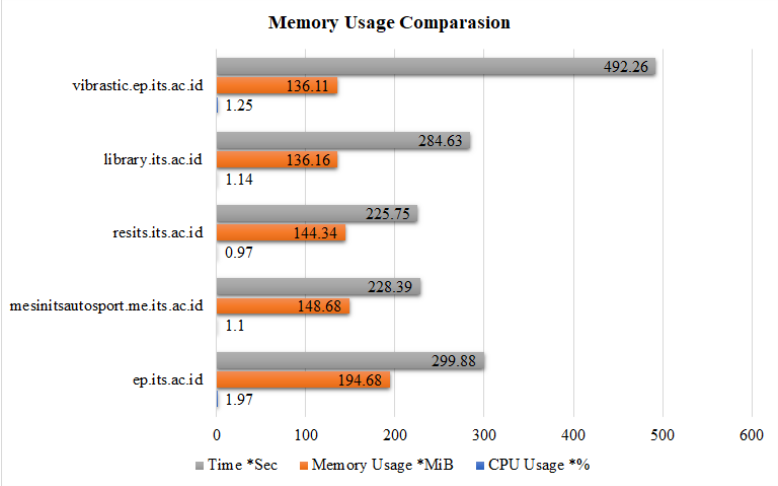
Tabel 5.12 Hasil uji coba 50 proses *scanning* pada satu waktu

No	Container Id	Target	Container CPU Usage *%			Container Memory Usage *MiB			Time *Sec
			min	max	avg	min	max	avg	
1	administrator-fb45f192	humas.its.ac.id	0	2.52	0.32	6.65	328.6	67.73	1152.53
2	administrator-fb45f193	amu.its.ac.id	0	1.93	0.35	7.24	335.67	86.4	305.34
3	administrator-fcd2c56c	fasor.its.ac.id	0	1.59	0.2	11.48	314.1	59.56	1038.59
4	administrator-fcd2c56d	enviro.its.ac.id	0	2.2	0.26	6.08	336.5	42.97	997.63
5	administrator-feb00930	tlingkungan.enviro.its.ac.id	0	1.22	0.19	6.16	53.84	30.77	1617.66
96	praktikum-25c6342c	m.library.its.ac.id	0.13	3.76	1.35	18.02	122.38	93.4	206.81
97	praktikum-2783bd84	resits.its.ac.id	0.13	2.09	0.97	18.84	335.3	144.34	225.75
98	praktikum-2aec9e64	spmi.its.ac.id	0	1.98	0.7	17.98	135.67	79.24	310.98
99	praktikum-2e0e2810	spektronics.its.ac.id	0.2	7.84	1.64	18.23	123	68.5	360.32
100	praktikum-30a68b76	baak.its.ac.id	0.12	3.13	1.26	18.24	134.41	101.61	184.6

Untuk mempermudah membaca hasil dari Tabel 5.12, dapat melihat Gambar 5.17 dan Gambar 5.18 yang melakukan perbandingan hubungan antara *CPU*, *memory* dan waktu untuk lima proses *scanning* yang menggunakan *CPU* dan *memory* terbesar. Pada skenario *stress test* terdapat 50 proses *scanning* yang berjalan secara bersamaan pada satu waktu. Pada saat menjalankan skenario *stress test* ini sistem tidak dapat dijalankan dengan optimal, terdapat kendala ketika mengakses halaman *web service* yang tersedia.



Gambar 5.17 Penggunaan *CPU* terbesar untuk 5 proses *scanning* pada skenario *stress test*



Gambar 5.18 Penggunaan *memory* terbesar untuk 5 proses *scanning* pada skenario *stress test*

Pada skenario *stress test* terdapat penurunan penggunaan *CPU load* untuk setiap proses *scanning*. Penurunan ini terjadi pada jumlah *CPU load* maupun presentase penggunaan *CPU load* dibandingkan dengan sistem. Jumlah *CPU load* menurun dibandingkan dengan keempat skenario sebelumnya, serta presentase penggunaan *CPU load* pada skenario ini menjadi 4,12% dibandingkan *CPU load* pada sistem. Hal ini dikarenakan pada skenario *stress test* terdapat 50 proses *scanning* yang berjalan dan untuk setiap proses *scanning* memerlukan *CPU load* yang harus dibagi dengan proses *scanning* yang lain. Untuk detailnya dapat ditunjukkan pada Tabel 5.13.

Tabel 5.13 Penggunaan *CPU* dan *memory* pada skenario *stress test*

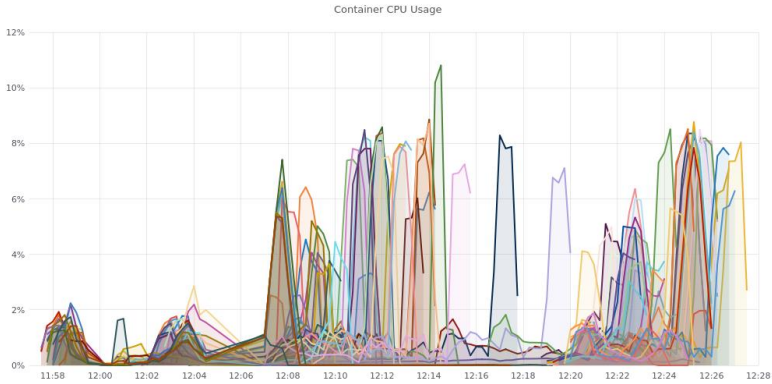
Skenario	CPU Load	Memory Load	Used Memory
0	1,73%	23%	449,75 MiB
1	3,16%	40%	1,04 GiB
2	13,79%	44%	2,40 GiB

3	21,18%	47%	2,94 GiB
4	23,77%	55%	3,27 GiB
5	25,26 %	55%	3,51 GiB

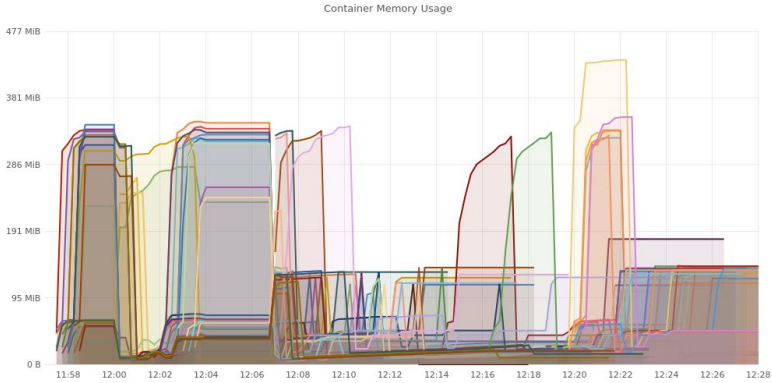
Penggunaan *CPU* pada sistem selama kondisi *stress test* ini berjalan dapat dilihat pada Gambar 5.19 dan penggunaan *memory* pada Gambar 5.20. Pada Gambar 5.19 dapat dilihat bahwa terjadi *CPU load* antara satu proses *scanning* dengan proses *scanning* lainya berjalan bersamaan, hal ini mengakibatkan penggunaan *CPU* pada sistem naik menjadi 25.26% atau naik 1,49% dibandingkan dengan *CPU load* pada skenario keempat, naik 4,08% dibandingkan dengan skenario ketiga, naik 11,47% dibandingkan dengan skenario kedua, dan naik 22,10% dibandingkan dengan skenario pertama. Dengan kenaikan 22,10% ini, proses *scanning* dapat berjalan untuk 50 target dalam satu waktu. Sedangkan penggunaan *memory* naik menjadi 3,51 GiB dibandingkan dengan penggunaan *memory* pada skenario pertama. Kenaikan penggunaan *memory* disebabkan karena ketika proses *scanning* berjalan terdapat proses *scanning* lain yang ikut berjalan dan dibutuhkan waktu untuk melepaskan *memory* yang digunakan seperti kondisi pada skenario pertama. Pada Gambar 5.20 diperlihatkan terdapat irisan penggunaan memory antara satu proses *scanning* dengan proses *scanning* yang lain. Pada skenario ini tidak ada perubahan kenaikan pada penggunaan *memory* secara keseluruhan, namun terdapat ketika menjalankan 50 proses *scanning* pada satu waktu penggunaan *memory* sempat mencapai 79%. Penggunaan *memory* yang mencapai 79% ini merupakan kondisi yang sangat tidak ideal untuk dapat digunakan dalam tahap *deployment* pada sistem dengan spesifikasi yang telah disebutkan pada bab 4.1.

Sehingga dengan sistem dengan spesifikasi yang tertera pada bab 4.1, penulis menyarankan penggunaan 20 proses *scanning* pada satu waktu. Hal ini untuk menjaga agar sistem dapat berjalan dengan optiomal dan untuk mengantisipasi apabila dalam

pengembangannya nanti akan ditambahkan beberapa *automated testing tools* yang secara langsung mempengaruhi *CPU load* dan *memory load* pada sistem. Pada tahap *deployment* aplikasi ini, yang perlu diperhatikan adalah kapasitas *memory* pada sistem untuk mendapatkan jumlah proses *scanning* yang ideal bagi sistem.



Gambar 5.19 Penggunaan *CPU* sistem pada skenario *stress test*



Gambar 5.20 Penggunaan *memory* sistem pada skenario *stress test*

(Halaman ini sengaja dikosongkan)

BAB VI
KESIMPULAN DAN SARAN

Bab ini berisikan kesimpulan yang dapat dapat diambil dari hasil uji coba yang telah dilakukan. Selain kesimpulan, terdapat juga saran yang ditujukan untuk pengembangan perangkat lunak selanjutnya.

6.1 Kesimpulan

Kesimpulan yang didapatkan berdasarkan hasil uji coba penerapan *vulnerability assessment and management* dapat diambil beberapa kesimpulan sebagai berikut:

1. Penerapan *vulnerability assessment and management* dapat dilakukan menggunakan *Docker container*, sehingga dapat dilakukannya *deployment* dan digunakan untuk melakukan proses *vulnerability assessment* terhadap website tujuan.
2. Penerapan satu *container* untuk satu proses *scanning* memungkinkan sistem dapat dijalankan oleh *mutli-user* yang menjalankan proses *scanning*, dan memanfaatkan *task queue* yang ada pada sistem agar proses *scanning* dapat dijalankan berdasarkan waktu *user* melakukan *request* proses *scanning*.
3. Proses *vulnerability assessment* dapat dilakukan berdasarkan skenario yang ditentukan, sehingga memperoleh informasi yang nantinya digunakan untuk melakukan tahapan selanjutnya.

4.a Penggunaan fitur *task queue* pada proses *scanning*, memungkinkan sistem menyesuaikan jumlah proses *scanning* yang dapat dijalankan berdasarkan *resource* yang tersedia pada sistem.

4.b Penggunaan *automated tools* dalam tugas akhir ini dapat ditambah atau dikurangi sesuai dengan keperluan pengembangan perangkat lunak selanjutnya, serta menyesuaikan dengan *resource* pada sistem agar dapat digunakan semaksimal mungkin.

6.2 Saran

Saran yang diberikan terkait pengembangan pada Tugas Akhir ini adalah:

1. Melakukan penambahan refrensi *vulnerability* pada *vulnerability assessment* agar memperoleh jenis *vulnerability* yang lebih beragam untuk dilakukannya perbandingan.
2. Melakukan penambahan dan perbaikan skenario dalam sistem secara berkala, sehingga hasil dari proses *scanning* dapat mengikuti perubahan teknologi pembuatan website.
3. Melakukan perubahan pada halaman antarmuka agar lebih mudah digunakan dan lebih informatif bagi *user* yang menggunakan.

DAFTAR PUSTAKA

[1] M. Rouse, "Passive Scanning," August 2014. [Online]. Available: https://whatis.techtarget.com/definition/passive-scanning. [Accessed 17 October 2019].

[2] "Common Vulnerability Scoring System," [Online]. Available: https://www.first.org/cvss/. [Accessed 17 October 2019].

[3] "OWASP Top 10 Application Security Risks," [Online]. Available: https://www.owasp.org/index.php/Top_10-2017_Top_10. [Accessed 17 October 2019].

[4] P. Mell, K. Scarfone and S. Romanosky, "Common Vulnerability Scoring System," *IEEE Security & Privacy*, vol. 4, no. 6, pp. 85-89, 2006.

[5] "What is a Container?," [Online]. Available: https://www.docker.com/resources/what-container. [Accessed 17 October 2019].

[6] R. Margaret, "What is a Vulnerability Assessment," April 2018. [Online]. Available: https://searchsecurity.techtarget.com/definition/vulnerability-assessment-vulnerability-analysis. [Accessed 17 October 2019].

[7] "Vulnerability Assessment and Penetration Testing," 18 March 2014. [Online]. Available: https://www.csa.gov.sg/gosafeonline/go-safe-for-business/smes/vulnerability-assessment-and-penetration-testing. [Accessed 10 October 2019].

[8] "What is Shodan?," 18 May 2019. [Online]. Available: https://help.shodan.io/the-basics/what-is-shodan. [Accessed 17 October 2019].

[9] "Nmap: the Network Mapper," [Online]. Available: https://nmap.org/. [Accessed 17 October 2019].

[10] "What is Python? Executive Summary," [Online]. Available: https://www.python.org/doc/essays/blurb/. [Accessed 20 October 2019].

LAMPIRAN
Kode Sumber Perangkat Lunak

Untuk melakukan instalasi *Docker* dari *default repository*, dapat menjalankan perintah berikut:

```
1   sudo apt-get update
2   sudo apt-get remove docker docker-engine
    docker.io

3   sudo apt install docker.io
4   sudo systemctl start docker
5   sudo systemctl enable docker
```

Kode Sumber 8.1 Install *Docker* dari *default repository*

Sedangkan untuk melakukan instalasi *Docker* dari *official repository* dapat menjalankan perintah berikut:

```
1   sudo apt-get update
2   sudo apt-get install apt-transport-https
    ca-certificates curl software-properties-
    common

3   curl -fsSL
    https://download.docker.com/linux/ubuntu/g
    pg | sudo apt-key add -

4   sudo add-apt-repository "deb [arch=amd64]
    https://download.docker.com/linux/ubuntu
    $(lsb_release -cs)  stable"

5   sudo apt-get update
6   sudo apt-get install docker-ce
```

Kode Sumber 8.2 Install *Docker* dari *official repository*

Agar dapat menjalankan *command Docker* tanpa menggunakan *command "sudo"*, dapat menggunakan perintah berikut:

```
1  sudo groupadd docker
2  sudo usermod -aG docker $USER
```

Kode Sumber 8.3 Perintah Menjalankan *Docker* tanpa *command "sudo"*

```
1  #!/bin/bash
2
3  #Global variabel
4  CWD=$(pwd)
5  DEFAULT_SHELL="$HOME/.bashrc"
6  PACKAGE_MANAGER="apt-get"
7  GO_DIR=~/go/bin
8  DATA_PATH="$CWD/data"
9  PLUGINS_PATH="$CWD/plugins"
10 WORKSPACE_PATH="$CWD/workspace"
11
12 #Make directory
13 mkdir -p $GO_DIR 2>/dev/null
14 mkdir -p $DATA_PATH 2>/dev/null
15 mkdir -p $PLUGINS_PATH 2>/dev/null
16 mkdir -p $PLUGINS_PATH/nmap 2>/dev/null
17 mkdir -p $PLUGINS_PATH/go 2>/dev/null
18 mkdir -p $WORKSPACE_PATH 2>/dev/null
19
20 #Download wordlist for "dns"
21 wget -q -O $DATA_PATH/commonspeak2-
   subdomains.txt
   https://raw.githubusercontent.com/assetnot
   e/commonspeak2-
   wordlists/master/subdomains/subdomains.txt
22 wget -q -O $DATA_PATH/shorts.txt
   https://raw.githubusercontent.com/danielmi
   essler/SecLists/master/Discovery/DNS/subdo
   mains-top1million-20000.txt
```

```
23  wget -q -O $DATA_PATH/short-
    permutation.txt
    https://raw.githubusercontent.com/subfinde
    r/goaltdns/master/words.txt
24  #Combine to one file
25  cat $DATA_PATH/commonspeak2-subdomains.txt
    $DATA_PATH/shorts.txt $DATA_PATH/short-
    permutation.txt | sort -u >
    $DATA_PATH/all-dns.txt
26
27  #Download wordlist for "content"
28  wget -q -O $DATA_PATH/quick.txt
    https://raw.githubusercontent.com/maurosor
    ia/dirsearch/master/db/dicc.txt
29  wget -q -O $DATA_PATH/top10000.txt
    https://raw.githubusercontent.com/danielmi
    essler/SecLists/master/Discovery/Web-
    Content/RobotsDisallowed-Top1000.txt
30  #Combine to one file
31  cat $DATA_PATH/quick.txt
    $DATA_PATH/top10000.txt | sort -u>
    $DATA_PATH/all-content.txt
32
33  #Download wordlist for "params"
34  wget -q -O $DATA_PATH/param-miner.txt
    https://raw.githubusercontent.com/PortSwig
    ger/param-miner/master/resources/params
35  wget -q -O $DATA_PATH/parameth.txt
    https://raw.githubusercontent.com/maK-
    /parameth/master/lists/all.txt
36  #Combine to one file
37  cat $DATA_PATH/param-miner.txt
    $DATA_PATH/parameth.txt | sort -u >
    $DATA_PATH/all-params.txt
38
39  #Download data for "subdomain takeover"
40  wget -q -O $DATA_PATH/providers-data.csv
    https://raw.githubusercontent.com/anshuman
    bh/tko-subs/master/providers-data.csv
```

```
41  wget -q -O $DATA_PATH/fingerprints.json
    https://raw.githubusercontent.com/haccer/s
    ubjack/master/fingerprints.json
42
43  #Download data for "sensitive key and
    passwords"
44  wget -q -O $DATA_PATH/keywords.txt
    https://raw.githubusercontent.com/random-
    robbie/keywords/master/keywords.txt
45  wget -q -O $DATA_PATH/resolvers.txt
    https://raw.githubusercontent.com/Abss0x7t
    bh/bass/master/resolvers/public.txt
46
47  #Download data for "Nmap"
48  wget -q -O $DATA_PATH/apps.json
    https://raw.githubusercontent.com/AliasIO/
    Wappalyzer/master/src/apps.json
49  wget -q -O $PLUGINS_PATH/nmap/vulners.nse
    https://raw.githubusercontent.com/vulnersC
    om/nmap-vulners/master/vulners.nse
50  wget -q -O $PLUGINS_PATH/nmap/nmap-
    bootstrap.xsl
    https://raw.githubusercontent.com/honze-
    net/nmap-bootstrap-xsl/master/nmap-
    bootstrap.xsl
51  wget -q -O
    $PLUGINS_PATH/nmap/masscan_xml_parser.py
    https://raw.githubusercontent.com/laconicw
    olf/Masscan-to-
    CSV/master/masscan_xml_parser.py
52  wget -q -O $PLUGINS_PATH/nmap/nmaptocsv.py
    https://raw.githubusercontent.com/maaaaz/n
    maptocsv/master/nmaptocsv.py
53
54  pip3 install setuptools 2>/dev/null
55  pip3 install wheel 2>/dev/null
56  pip3 install -r requirement.txt
57  chmod +x wolfrevo.py
58  shodan init "$(cat shodan_api_key.txt)"
```

```
59
60  #Add gopath
61  if ! grep -Fxq "GOPATH" "$DEFAULT_SHELL";
    then
62      echo 'export GOPATH=$HOME/go'
    >>$DEFAULT_SHELL
63      echo 'PATH=$GOPATH/bin:$PATH'
    >>$DEFAULT_SHELL
64      source $DEFAULT_SHELL
65  fi
66  source $DEFAULT_SHELL
67
68  #Update golang version
69  wget -qO-
    https://raw.githubusercontent.com/udhos/up
    date-golang/master/update-golang.sh | bash
    2>/dev/null
70
71  #Install tools with golang
72  GO_BIN=/usr/local/go/bin/go
73  $GO_BIN get -u
    github.com/projectdiscovery/subfinder/cmd/
    subfinder
74  export GO111MODULE=on; $GO_BIN get -u
    github.com/OWASP/Amass/v3/... 2>/dev/null
75  $GO_BIN get -u
    github.com/tomnomnom/httprobe
76  $GO_BIN get -u
    github.com/rverton/webanalyze/...
77
78  #Copy to directory go in "plugins path"
79  cp $GO_DIR/* "$PLUGINS_PATH/go/"
    2>/dev/null
80
81  #Install tool from "git"
82  cd $PLUGINS_PATH
83  git clone https://github.com/RUB-
    NDS/CORStest 2>/dev/null
```

Kode Sumber 8.4 Isi berkas *install.sh*

```
1  import argparse
2  import socket
3
4  from lib.utilities import banner
5  from lib.utilities import utils
6  from lib.utilities import process
7
8  from lib.core import shodan
9
10 start_time = utils.generate_timestamp()
11
12 def scanning(args):
13     utils.print_target(args.target)
14     process.scanning_process(args.target,
   args.module, start_time)
15
16 def main():
17     banner.banner_()
18     parser = argparse.ArgumentParser(
           description="Vulnerability
   Assessment and Management")
19     parser.add_argument('-t', '--target',
   action='store', dest='target',
   help='target')
20     parser.add_argument('-m', '--module',
   action='store', dest='module',
   help='select module to use in scanning')
21
22     args = parser.parse_args()
23     scanning(args)
24
25 if __name__ == "__main__":
26     main()
```

Kode Sumber 8.5 Isi berkas *wolfrevo.py*

```
1  from lib.utilities import utils
2
3  from lib.core import httprobe
4  from lib.core import shodan
5  from lib.core import nmap
6  from lib.core import subfinder
7  from lib.core import amass
8  from lib.core import webanalyze
9  from lib.core import combine
10 from lib.core import corscan
11
12 def scanning_process(target, options,
   time):
13     #Full Scan
14     if options == '0':
15         httprobe.scanning(target, time)
16         amass.scanning(target, time)
17         subfinder.scanning(target, time)
18         combine.combine_(target, time)
19         corscan.scanning(target, time)
20         webanalyze.scanning(target, time)
21         nmap.scanning(target, time)
22         shodan.scanning(target, time)
23     #Subdomain Scan
24     elif options == '1':
25         amass.scanning(target, time)
26         subfinder.scanning(target, time)
27         combine.combine_(target, time)
28     #Fingerprint
29     elif options == '2':
30         httprobe.scanning(target, time)
31         corscan.scanning(target, time)
32         webanalyze.scanning(target, time)
33     #Vulnerability Scan
34     elif options == '3':
35         nmap.scanning(target, time)
36         shodan.scanning(target, time)
```

```
37      else:
38          print('please select module!')
```

Kode Sumber 8.6 Isi berkas *process.py*

```
1   import os
2   import sys
3   import time
4   import ipaddress
5   import socket
6   import urllib
7   import subprocess
8
9   from datetime import datetime
10
11  # define path
12  current_path =
    os.path.dirname(os.path.realpath(__file__
    ))
13  root_path =
    os.path.dirname(os.path.dirname(current_p
    ath))
14  workspace_path =
    os.path.join(root_path,'workspace')
15  plugins_path =
    os.path.join(root_path,'plugins')
16  data_path =
    os.path.join(root_path,'data')
17
18  # file and folder
19  def root_directory():
20      return root_path
21
22  def plugins_directory():
23      return plugins_path
24
25  def data_directory():
26      return data_path
27
```

```
28  def workspace(module_name, target,
    timestamp, ext_file):
29      file_name =
    os.path.join(workspace_path,
    str(timestamp) + '_' + module_name + '_'
    + target + '.' + ext_file)
30      return file_name
31
32  def delete_file(filename):
33      cmd = 'rm '+filename
34      run_program(cmd)
35
36  def running_time_file(target, timestamp,
    start_time, banner):
37      run_time = time.perf_counter() -
    start_time
38      running_time_file =
    open(workspace('running_time',target,time
    stamp,'txt'), 'a+')
39      running_time_file.write(f'[RUNNING
    TIME] {banner} executed in
    {run_time:0.2f} seconds.''\n')
40
41  # color
42  W = '\033[1;0m'     #white
43  R = '\033[1;31m'    #red
44  G = '\033[1;32m'    #green
45  B = '\033[1;34m'    #blue
46
47  info = '{0}[*]{1}'.format(B, W)
48  good = '{0}[+]{1}'.format(G, W)
49  bad = '{0}[-]{1}'.format(R, W)
50
51  def print_target(text):
52      print('{1}---<---<---@ {2}Target: {0}
    {1}@--->--->---'.format(text, W, G))
53
54  def print_line():
55      print(f'{W}' + '-'*70)
```

```
56
57  def print_good(text):
58      print(good + " " + text)
59
60  def print_bad(text):
61      print(bad + " " + text)
62
63  def print_load(text, tag='LOAD'):
64      print(f'{W}' + '-'*70)
65      print(f'{W}[{B}{tag}{W}] {G}{text}')
66      print(f'{W}' + '-'*70)
67
68  # timestamp
69  def generate_timestamp():
70      timestamp = int(time.time())
71      return timestamp
72
73  def readable_timestamp():
74      timestamp = int(time.time())
75      return
    str(datetime.utcfromtimestamp(timestamp).
    strftime('%d/%m/%Y')) + f"__{timestamp}"
76
77  def get_start_time():
78      return time.perf_counter()
79
80  def get_running_time(start_time, banner):
81      print_line()
82      running_time = time.perf_counter() -
    start_time
83      print(f'{W}[{B}RUNNING TIME]
    {W}{banner} executed in
    {B}{running_time:0.2f}{W} seconds.\n')
84      print_line()
85
86  # check ip address and domain name
87  def check_ip_address(input_user):
88      try:
```

```
89  ipaddress.ip_interface(str(input_user).st
    rip())
90              return True
91          except:
92              return False
93
94  def get_ip_address(input_user):
95      if check_ip_address(input_user):
96          return input_user
97      else:
98          try:
99              ip_address =
    socket.gethostbyname(get_domain_name(inpu
    t_user))
100             return ip_address
101         except:
102             return False
103
104 def get_domain_name(input_user):
105     parsed =
    urllib.parse.urlparse(input_user)
106     domain_name = parsed.netloc if
    parsed.netloc else parsed.path
107     return domain_name
108
109 # subprocess to run program
110 def run_program(cmd):
111     stdout = ''
112     try:
113         program = subprocess.Popen(cmd,
    shell=True, stdout=subprocess.PIPE,
    stderr=subprocess.STDOUT)
114         while True:
115             nextline =
    program.stdout.readline().decode('utf-8')
116             if nextline == '' and
    program.poll() is not None:
117                 break
```

```
118                 print(nextline, end='')
119                 stdout += nextline
120                 sys.stdout.flush()
121 
122             exitcode = program.returncode
123 
124             if (exitcode == 0):
125                 return stdout
126             else:
127                 print_line()
128                 print('Something wrong: ')
129                 print(cmd)
130                 print_line()
131                 return None
132         except:
133             print_line()
134             print('Something wrong: ')
135             print(cmd)
136             print_line()
137             return None
```

Kode Sumber 8.7 Isi berkas *utils.py*

```
1   import os
2   import subprocess
3   from lib.utilities import utils
4   
5   module_name = "amass"
6   banner = 'Scanning with Amass'
7   start_time = utils.get_start_time()
8   
9   def scanning(text, uuid):
10      target = utils.get_domain_name(text)
11      utils.print_load(banner)
12      writefile =
    open(utils.workspace(module_name, target,
    uuid, 'txt'),'w+')
13      try:
            cmd =
14  '/root/wolfrevo/plugins/go/amass enum -
```

```
    timeout 10 -d '+target+' -o
    '+writefile.name
            utils.run_program(cmd)
15          utils.print_good(banner)
16      except:
17          utils.delete_file(writefile.name)
18          utils.print_bad(banner)
19
20      utils.get_running_time(start_time,
21  banner)
        utils.running_time_file(target, uuid,
22  start_time, banner)
```

Kode Sumber 8.8 Isi berkas *amass.py*

```
1   import os
2   import subprocess
3   from lib.utilities import utils
4
5   module_name = 'combine'
6   banner = 'Combine to one file subdomain'
7   start_time = utils.get_start_time()
8
9   def combine_(text, uuid):
10      target = utils.get_domain_name(text)
11      utils.print_load(banner)
12      writefile =
        open(utils.workspace(module_name,target,uu
        id, 'txt'), 'w+')
13      file_amass =
        os.path.join(utils.workspace_path,str(uuid
        )+'_amass_*')
14      file_subfinder =
        os.path.join(utils.workspace_path,str(uuid
        )+'_subfinder_*')
15      try:
16          cmd = 'cat '+file_amass+'
        '+file_subfinder+' | sort -u >
        '+writefile.name
17          utils.run_program(cmd)
```

```
18          utils.print_good(banner)
19      except:
20          utils.delete_file(writefile.name)
21          utils.print_bad(banner)
22
23      utils.get_running_time(start_time,
    banner)
24      utils.running_time_file(target, uuid,
    start_time, banner)
```

Kode Sumber 8.9 Isi berkas *combine.py*

```
1   import os
2   import subprocess
3   from lib.utilities import utils
4
5   module_name = "cors"
6   banner = 'Scanning with CORS'
7   start_time = utils.get_start_time()
8
9   def scanning(text, uuid):
10      target = utils.get_domain_name(text)
11      utils.print_load(banner)
12      httprobe_file =
    open(utils.workspace('httprobe',target,uui
    d,'txt'), 'r')
13      writefile =
    open(utils.workspace(module_name,target,uu
    id,'txt'), 'w+')
14      try:
15          cors_plugins =
    os.path.join(utils.plugins_directory(),'CO
    RStest','corstest.py')
16          cmd = cors_plugins+'
    '+httprobe_file.name+' > '+writefile.name
17          utils.run_program(cmd)
18          utils.print_good(banner)
19      except:
20          utils.delete_file(writefile.name)
21          utils.print_bad(banner)
```

```
22
23      utils.get_running_time(start_time,
    banner)
24      utils.running_time_file(target, uuid,
    start_time, banner)
```

Kode Sumber 8.10 Isi berkas *corscan.py*

```
1   import os
2   import subprocess
3   from lib.utilities import utils
4
5   module_name = "httprobe"
6   banner = 'Processing httprobe'
7   start_time = utils.get_start_time()
8
9   def scanning(text, uuid):
10      target = utils.get_domain_name(text)
11      httprobe_file =
    open(utils.workspace('httprobe',target,uui
    d,'txt'), 'w+')
12      utils.print_load(banner)
13      try:
14          utils.print_good(banner)
15          cmd_httprobe = 'echo "'+target+'"
    | /root/wolfrevo/plugins/go/httprobe >
    '+httprobe_file.name
16          utils.run_program(cmd_httprobe)
17      except:
18          utils.print_bad(banner)
19
20      utils.get_running_time(start_time,
    banner)
21      utils.running_time_file(target, uuid,
    start_time, banner)
```

Kode Sumber 8.11 Isi berkas *httprobe.py*

```
1   import os
2   import subprocess
3   from lib.utilities import utils
```

```
4
5   module_name = "nmap"
6   banner = 'Scanning with Nmap'
7   start_time = utils.get_start_time()
8
9   def scanning(text, uuid):
10      target = utils.get_ip_address(text)
11      target_domain =
    utils.get_domain_name(text)
12      utils.print_load(banner)
13      write_file =
    open(utils.workspace(module_name,
    target_domain, uuid, 'xml'),'w+')
14      try:
15          cmd = 'nmap --open -T4 -v0 -Pn -n
    -sSV -p- '+target+' --script
    vulscan/vulscan.nse --oX '+write_file.name
16          utils.run_program(cmd)
17          utils.print_good(banner)
18      except:
19          utils.print_bad(banner)
20          utils.delete_file(write_file.name)
21
22      utils.get_running_time(start_time,
    banner)
23      utils.running_time_file(target_domain,
    uuid, start_time, banner)
```

Kode Sumber 8.12 Isi berkas *nmap.py*

```
1   import socket
2   import json
3   import subprocess
4   from shodan import Shodan
5
6   from lib.utilities import utils
7
8   api =
    Shodan('V8eQMSxtuxTeotGRVcSEjZPrKDkE4s6E')
9
```

```
10 module_name = "shodan"
11 banner = 'Scanning with Shodan'
12 start_time = utils.get_start_time()
13
14 def scanning(text, uuid):
15     target = utils.get_ip_address(text)
16     target_domain =
   utils.get_domain_name(text)
17     vulnerability_list=[]
18     utils.print_load(banner)
19     try:
20         result = api.host(target)
21         if result['vulns'] is not None:
22             write_file =
   open(utils.workspace(module_name,
   target_domain, uuid, 'json'),'w+')
23             vulns = result['data'][-
   1]['vulns']
24             for detail in vulns:
25                 vulnerability_detail =
   {'cve':None, 'cvss':None, 'summary':None}

26 vulnerability_detail['cve'] = detail

27 vulnerability_detail['cvss'] =
   vulns[detail]['cvss']

28 vulnerability_detail['summary'] =
   vulns[detail]['summary']

29 vulnerability_list.append(vulnerability_de
   tail)

30 write_file.write(json.dumps(vulnerability_
   list))
31             utils.print_good(banner)
32     except:
33         utils.print_bad(banner)
34
```

```
35      utils.get_running_time(start_time,
    banner)
36      utils.running_time_file(target_domain,
    uuid, start time, banner)
```

Kode Sumber 8.13 Isi berkas *shodan.py*

```
1   import os
2   import subprocess
3   from lib.utilities import utils
4
5   module_name = "subfinder"
6   banner = 'Scanning with Subfinder'
7   start_time = utils.get_start_time()
8
9   def scanning(text, uuid):
10      target = utils.get_domain_name(text)
11      utils.print_load(banner)
12      writefile =
    open(utils.workspace(module_name, target,
    uuid, 'txt'),'w+')
13      try:
14          cmd =
    '/root/wolfrevo/plugins/go/subfinder -d
    '+target+' -t 100 -o '+writefile.name
15          utils.run_program(cmd)
16          utils.print_good(banner)
17      except:
18          utils.delete_file(writefile.name)
19          utils.print_bad(banner)
20
21      utils.get_running_time(start_time,
    banner)
22      utils.running_time_file(target, uuid,
    start time, banner)
```

Kode Sumber 8.14 Isi berkas *subfinder.py*

```
1   import os
2   import subprocess
3   from lib.utilities import utils
```

```
4
5   module_name = "webanalyze"
6   banner = 'Scanning with Webanalyze'
7   start_time = utils.get_start_time()
8   apps_path =
    os.path.join(utils.data_directory(),'apps.
    json')
9
10  def scanning(text, uuid):
11      target = utils.get_domain_name(text)
12      utils.print_load(banner)
13      writefile =
    open(utils.workspace(module_name, target,
    uuid, 'json'),'w+')
14      try:
15          cmd =
    '/root/wolfrevo/plugins/go/webanalyze -
    apps '+apps_path+' -host '+target+' -
    output json >> '+writefile.name
16          utils.run_program(cmd)
17          utils.print_good(banner)
18      except:
19          utils.delete_file(writefile)
20          utils.print_bad(banner)
21
22      utils.get_running_time(start_time,
    banner)
23      utils.running_time_file(target, uuid,
    start_time, banner)
```

Kode Sumber 8.15 Isi berkas *webanalyze.py*

(Halaman ini sengaja dikosongkan)

BIODATA PENULIS

Gusti Ngurah Satria Aryawan merupakan anak dari pasangan Bapak Gusti Nyoman Arya dan Ibu Ni Ketut Adnyani. Lahir di Singaraja pada tanggal 17 November 1996. Penulis menempuh pendidikan formal dimulai dari SDN 2 Banjar Bali, Buleleng (2003-2009), SMPN 1 Singaraja (2009-2012), SMAN 1 Singraja (2012-2015) dan S1 Departemen Teknik Informatika ITS (2015-2020). Bidang studi yang diambil oleh penulis pada saat berkuliah di Departemen Teknik Informatika ITS adalah Arsitektur dan Jaringan Komputer (AJK). Penulis aktif dalam organisasi seperti Himpunan Mahasiswa Teknik Computer-Informatika (2016-2018) dan TPKH-ITS (2015-2016). Penulis juga aktif dalam berbagai kegiatan perlombaan dibidang jaringan komputer dan keamanan siber seperti perlombaan *capture the flag* (CTF) dan Huawei ICT Competition. Penulis memiliki hobi mendengarkan musik dan bermain game. Selama berkuliah di Departemen Informatika, penulis telah menjadi asisten untuk beberapa mata kuliah seperti sistem operasi, jaringan komputer dan sistem terdistribusi. Penulis dapat dihubungi melalui email: satriaaryawan.gusti@gmail.com.