PERANCANGAN SISTEM KEAMANAN PADA UBUNTU SERVER TERHADAP *BUFFER OVERFLOW* DAN *FORMAT STRING* MENGGUNAKAN FAIL2BAN DAN APPARMOR

SKRIPSI

WAHYU PRIAMBODO

2207421048

PROGRAM STUDI TEKNIK MULTIMEDIA DAN JARINGAN

JURUSAN TEKNIK INFORMATIKA DAN KOMPUTER

POLITEKNIK NEGERI JAKARTA

2026

PERANCANGAN SISTEM KEAMANAN PADA UBUNTU SERVER TERHADAP *BUFFER OVERFLOW* DAN *FORMAT STRING* MENGGUNAKAN FAIL2BAN DAN APPARMOR

SKRIPSI

Dibuat untuk Melengkapi Syarat-Syarat yang Diperlukan untuk Memperoleh Diploma Empat Politeknik

WAHYU PRIAMBODO

2207421048

PROGRAM STUDI TEKNIK MULTIMEDIA DAN JARINGAN

JURUSAN TEKNIK INFORMATIKA DAN KOMPUTER

POLITEKNIK NEGERI JAKARTA

2026

SURAT PERNYATAAN BEBAS PLAGIARISME

Saya yang bertanda tangan di bawah ini:

Nama	:	Wahyu Priambodo
NIM	:	2207421048
Jurusan / Program Studi	:	Teknik Informatika dan Komputer / Teknik Multimedia dan Jaringan
Judul Skripsi	:	Perancangan Sistem Keamanan pada Ubuntu Server Terhadap *Buffer Overflow* dan *Format String* Menggunakan Fail2Ban dan AppArmor

Menyatakan dengan sebenarnya bahwa skripsi ini benar-benar merupakan hasil karya saya sendiri, bebas dari peniruan terhadap karya dari orang lain. Kutipan pendapat dan tulisan orang lain ditunjuk sesuai dengan cara-cara penulisan karya ilmiah yang berlaku.

Apabila di kemudian hari terbukti atau dapat dibuktikan bahwa dalam skripsi ini terkandung ciri-ciri plagiat dan bentuk-bentuk peniruan lain yang dianggap melanggar peraturan, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Depok, 21 Juni 2026

Yang membuat pernyataan

(Wahyu Priambodo)

NIM. 2207421048

LEMBAR PENGESAHAN

Skripsi diajukan oleh :

Nama	:	Wahyu Priambodo
NIM	:	2207421048
Jurusan / Program Studi	:	Teknik Informatika dan Komputer / Teknik Multimedia dan Jaringan
Judul Skripsi	:	Perancangan Sistem Keamanan pada Ubuntu Server Terhadap *Buffer Overflow* dan *Format String* Menggunakan Fail2Ban dan AppArmor

Telah diuji oleh tim penguji dalam sidang skripsi pada hari Kamis, Tanggal 2, Bulan Juli, Tahun 2026 dan dinyatakan **LULUS**.

Disahkan oleh

Pembimbing I	: Iik Muhamad Malik Matin, S.Kom., M.T.	( )
Penguji I	: Defiana Arnaldy, S.Tp., M.Si.	( )
Penguji II	: Ayu Rosyida Zain, S.ST, M.T.	( )
Penguji III	: Ariawan Andi Suhandana, S.Kom., M.T.I.	( )

Mengetahui :

Jurusan Teknik Informatika dan Komputer

Ketua

Dr. Anita Hidayati, S.Kom., M.Kom.

NIP. 197908032003122003

KATA PENGANTAR

Puji dan syukur selalu penulis panjatkan kepada Allah *subhanahu wa ta'ala*, Tuhan Yang Maha Esa, karena atas rahmat dan karunia-Nya, penulis mampu menyelesaikan skripsi ini yang berjudul "Perancangan Sistem Keamanan pada Ubuntu Server Terhadap *Buffer Overflow* dan *Format String* menggunakan Fail2Ban dan AppArmor". Skripsi ini dibuat dengan tujuan untuk memenuhi syarat dalam memperoleh gelar Diploma Empat (D4) Politeknik. Penulis menyadari bahwa terdapat berbagai pihak yang telah mendukung, membantu, dan membimbing selama proses penelitian. Oleh karena itu, izinkan penulis untuk menyampaikan terima kasih yang sebesar-besarnya kepada:

1. Bapak Iik Muhamad Malik Matin, S.Kom., M.T., selaku dosen pembimbing penulis yang telah mendedikasikan waktu, kesabaran, dan ilmunya selama proses penelitian dan menyusun skripsi ini.
2. Komunitas Xyl0n PNJ yang senantiasa mengajak penulis untuk belajar ilmu-ilmu yang baru, terutama di dunia keamanan siber, seperti *Capture The Flag* (CTF) dan Linux *binary exploitation* (*pwn*).
3. Kedua orang tua, saudara, dan kerabat yang telah memberikan doa dan dukungan, baik moral ataupun material, serta motivasi untuk tetap semangat dalam menuntaskan skripsi ini.
4. Sahabat dan teman-teman kelas yang sudah mau berbagi ilmu dan berjuang bersama-sama selama proses mengerjakan skripsi.

Penulis menyadari bahwa masih terdapat keterbatasan pengetahuan dan pengalaman selama proses penelitian, sehingga skripsi ini jauh dari kata sempurna. Oleh karena itu, penulis sangat mengharapkan kritik dan saran yang membangun dari pihak-pihak lain supaya skripsi ini bisa lebih baik lagi ke depannya. Akhir kata, penulis berharap agar skripsi ini dapat bermanfaat dan menambah wawasan baru bagi orang lain, terutama *security researcher*.

Depok, 21 Juni 2026

Wahyu Priambodo

SURAT PERNYATAAN PERSETUJUAN PUBLIKASI SKRIPSI UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Politeknik Negeri Jakarta, saya bertanda tangan di bawah ini:

Nama	:	Wahyu Priambodo
NIM	:	2207421048
Jurusan / Program Studi	:	Teknik Informatika dan Komputer / Teknik Multimedia dan Jaringan

Demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Politeknik Negeri Jakarta Hak Bebas Royalti Non-Eksklusif atas karya ilmiah saya yang berjudul :

Perancangan Sistem Keamanan pada Ubuntu Server Terhadap *Buffer Overflow* dan *Format String* Menggunakan Fail2Ban dan AppArmor

Beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Non-Eksklusif ini Politeknik Negeri Jakarta berhak menyimpan, mengalihmediakan/formatkan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan mempublikasikan skripsi saya tanpa meminta izin dari saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Depok, 21 Juni 2026

Yang menyatakan,

(Wahyu Priambodo)

NIM. 2207421048

Perancangan Sistem Keamanan pada Ubuntu Server Terhadap *Buffer Overflow* dan *Format String* Menggunakan Fail2Ban dan AppArmor

ABSTRAK

Kerentanan memori seperti buffer overflow dan format string masih sering ditemukan hingga saat ini. Mitigasi bawaan pada binary di Linux, seperti NX, RELRO, PIE, dan Stack Canary menjadi tidak efektif apabila tidak dikonfigurasi maksimal dan dikombinasikan dengan lemahnya praktik secure coding. Penelitian ini bertujuan untuk merancang sistem keamanan berlapis yang mengombinasikan AppArmor sebagai lapisan pencegah eksploitasi di tingkat sistem operasi dan Fail2Ban sebagai lapisan pemblokiran alamat IP di tingkat jaringan. Rancangan sistem keamanan diimplementasikan pada tiga mesin virtual Ubuntu Server 24.04 dengan proteksi bertingkat, yaitu VM-1 (tanpa proteksi), VM-2 (proteksi sebagian), dan VM-3 (proteksi penuh). Penelitian ini juga memanfaatkan alat monitoring, seperti Grafana, Loki, dan Prometheus. Objek penelitian menggunakan 10 file ELF binary yang diskenariokan memiliki celah dengan variasi proteksi dan alur logika program berbeda-beda. Tahap pengujian dilakukan melalui 4 skenario berbeda, yaitu fungsionalitas, pembatasan dampak oleh AppArmor, pemblokiran alamat IP oleh Fail2Ban, dan pembanjiran koneksi. Hasil pengujian pada binary dengan profil AppArmor longgar masih memungkinkan penyerang memperoleh server secara penuh, karena tidak adanya pembatasan ruang lingkup. Di sisi lain, Fail2Ban mampu memblokir alamat IP berdasarkan pola crash berulang dan pembanjiran koneksi yang melebihi ambang batas. Pengembangan lebih lanjut bisa dengan menambahkan notifikasi alert dan memperdalam pengujian di sisi jaringan.

Kata Kunci: AppArmor, *Buffer Overflow*, CTFd, Fail2Ban, *Format String*, Grafana, *Hardening Server*, Loki, Prometheus, Ubuntu Server

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

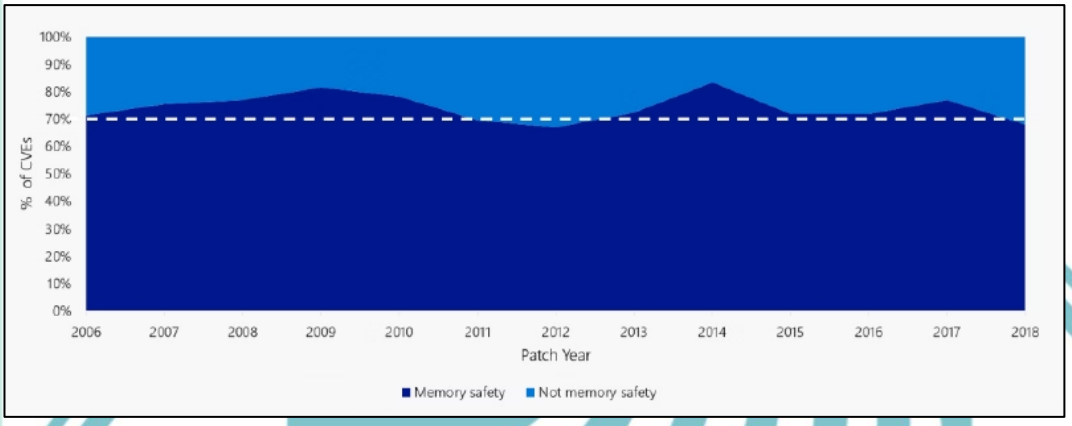
BAB I
PENDAHULUAN

1.1 Latar Belakang Masalah

Setiap aplikasi atau *binary* yang berjalan di sistem operasi berbasis Linux, termasuk Ubuntu Server, umumnya telah dilengkapi mekanisme mitigasi bawaan seperti *No Execute* (NX), *Relocation Read-Only* (RELRO), *Stack Canary* (StackGuard), dan *Position Independent Executable* (PIE) (Butt et al., 2022). Mekanisme mitigasi tersebut berfungsi untuk melindungi program dari berbagai upaya eksploitasi keamanan memori, seperti celah *buffer overflow*, *format string*, dan kerentanan memori lainnya (Butt et al., 2022). Selain itu, terdapat pula mitigasi pada level sistem operasi, yaitu *Address Space Layout Randomization* (ASLR), yang bekerja dengan merandomisasi tata letak ruang alamat proses, sehingga membuat upaya eksploitasi *binary* menjadi semakin sulit (Marco-Gisbert dan Ripoll, 2019). Permasalahan muncul ketika mekanisme mitigasi tersebut tidak dikonfigurasi secara maksimal, sehingga eksploitasi *buffer overflow* dan *format string* tetap berpeluang terjadi. Kondisi ini diperburuk apabila pengembang aplikasi tidak menerapkan praktik *secure coding* atau lemahnya *quality control*, yang menyebabkan mitigasi bawaan pada aplikasi dan ASLR menjadi tidak efektif (Cybersecurity and Infrastructure Security Agency and Federal Bureau of Investigation, 2025). Oleh karena itu, diperlukan mekanisme mitigasi tambahan untuk menutupi keterbatasan mitigasi bawaan pada program di sistem operasi berbasis Linux.

Buffer overflow merupakan celah keamanan yang diakibatkan oleh masukan (*input*) melebihi kapasitas alokasi *buffer* yang telah ditentukan, sehingga dapat menimpa bagian lain di dalam memori (George et al., 2021). Sementara itu, *format string* adalah celah keamanan serius yang dapat dimanfaatkan untuk membocorkan (*read*) maupun mengubah (*write*) data atau alamat di dalam memori (Xu et al., 2024). Kedua celah tersebut menarik untuk dikaji lebih mendalam, karena sama-sama dapat menjadi rantai kerentanan yang berujung pada eksekusi kode arbitrer (*arbitrary code execution*) (Petrean dan Coleşa, 2024).

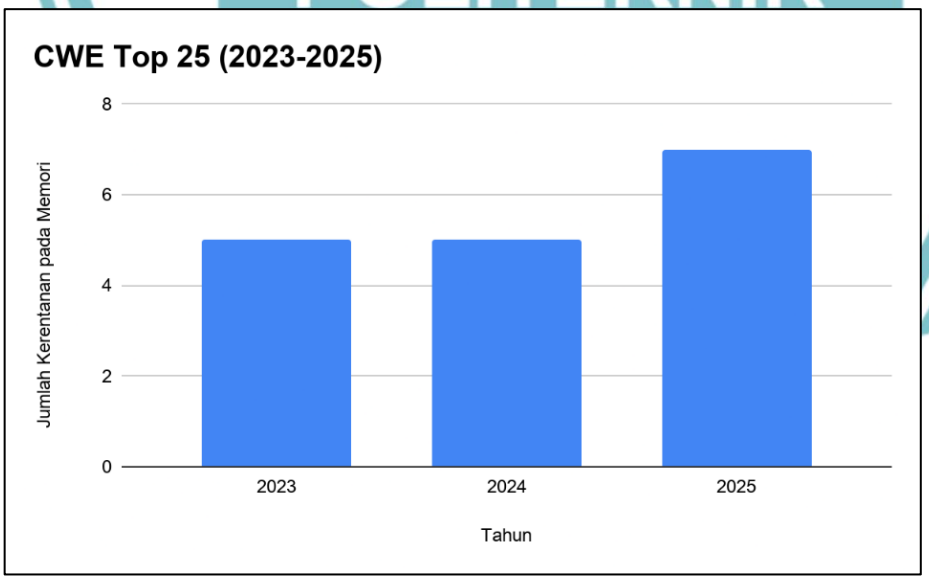
Pernyataan tersebut diperkuat oleh laporan dari Microsoft dan pada tahun 2019 yang menyatakan jika sekitar 70% isu celah keamanan didominasi oleh kerentanan pada memori (Miller, 2019). Gambar 1.1 merepresentasikan laporan dari Microsoft pada tahun 2019.



Gambar 1. 1 Dominasi *Memory Corruption Bugs* di Sistem Microsoft Tahun 2019

Sumber: (Miller, 2019)

Daftar *Top 25 Common Weakness Enumeration* (CWE) periode 2023 - 2025 menunjukkan tren peningkatan eksploitasi terhadap kerentanan memori (MITRE, 2025). Grafik tren kenaikan dapat dilihat pada Gambar 1.2. Dengan demikian, kerentanan pada memori masih menjadi ancaman yang relevan hingga saat ini.



Gambar 1. 2 Tren Kenaikan *Memory Corruption Bugs* pada 2023-2025

Sumber: (MITRE, 2025)

Objek penelitian ini menggunakan total 10 *file ELF binary* yang dirancang memiliki kerentanan *buffer overflow* dan *format string*. Kesepuluh *file binary* tersebut memiliki variasi alur logika program dan proteksi yang berbeda-beda dan harus dilewati oleh penyerang supaya bisa mendapatkan akses ke dalam sistem. Sistem keamanan yang telah dirancang akan diekspos ke penyerang dalam bentuk kompetisi *Capture The Flag* (CTF) yang dapat diakses secara publik untuk pengambilan data pengujian dan evaluasi sistem keamanan (Maulana et al., 2023). Kompetisi CTF akan memberikan kesepuluh *file binary* yang rentan ke penyerang untuk dieksploitasi hingga mendapatkan akses ke dalam sistem dan membaca file *flag*. Penyerang membutuhkan teknik-teknik yang relevan untuk eksploitasi celah *buffer overflow* dan *format string*. Teknik eksploitasi *buffer overflow* dapat berupa *Return Oriented Programming* (ROP), *Return-to-Libc* (*ret2libc*), dan lain sebagainya. Sedangkan, teknik eksploitasi *format string* dapat berupa *Global Offset Table* (GOT) *Overwrite*. Teknik-teknik apapun yang digunakan oleh penyerang dikategorikan valid selama proses pengujian asalkan tidak melakukan pelanggaran seperti *sharing flag* dan merusak platform CTF.

Ubuntu Server dipilih sebagai lingkungan pengujian pada penelitian ini karena terkenal andal dan memiliki reputasi yang stabil dan aman untuk penggunaan standar server (Erawan dan Salman, 2023). Selain itu, menurut Erawan dan Salman (2023) juga menjelaskan bahwa Ubuntu sudah memiliki fitur keamanan bawaan, salah satunya adalah AppArmor, yang dapat membantu membatasi dampak serangan. Versi 24.04 dipilih karena memiliki dukungan pemeliharaan sistem operasi yang panjang atau *Long Term Support* (LTS) hingga 5 tahun. Dukungan tersebut merepresentasikan server produksi yang stabil dan relevan dalam jangka waktu lama.

Studi literatur terhadap penelitian terdahulu dilakukan untuk menemukan celah penelitian (*research gap*). Penelitian Liu et al. (2023) dilakukan dengan tujuan mengembangkan metode deteksi dan pertahanan *buffer overflow* berbasis ekstensi *instruction set* RISC-V. Penelitian Rasyid dan Santoso (2026) kedua dilakukan dengan tujuan menemukan kerentanan *buffer overflow* pada perangkat lunak PDFCook versi 0.4.5 menggunakan teknik *fuzzing* (AFL++) yang dikombinasikan dengan *Address Sanitizer* (ASan). Penelitian Petrean dan Coleşa (2024) dilakukan

dengan tujuan merancang *tool* otomatis (PwnMaster) untuk mendeteksi sekaligus mengeksploitasi kerentanan *buffer overflow* dan *format string* pada *file ELF* menggunakan *symbolic execution* dan *pwntools*. Penelitian Xu dan Wang (2022) dilakukan dengan tujuan membangun sistem *automated exploit generation* (BofAEG) untuk mendeteksi dan menghasilkan eksploitasi *stack buffer overflow* secara otomatis berbasis *symbolic execution* dan analisis dinamis. Penelitian Kim et al. (2024) dilakukan dengan tujuan meningkatkan keamanan aplikasi berbasis bahasa C melalui metode *StackGuard*$^{+}$ sebagai alternatif interoperabel terhadap proteksi berbasis *canary* konvensional. Dari kelima literatur tersebut, belum ada penelitian yang mengkaji pendekatan *hardening* pada Ubuntu Server terhadap eksploitasi *buffer overflow* dan *format string* menggunakan Fail2Ban dan AppArmor.

Selain kelima studi literatur terkait *buffer overflow* dan *format string*, pencarian studi literatur terkait pemanfaatan alat Fail2Ban sebagai metode *Intrusion Prevention System* (IPS) juga dilakukan. Dari hasil studi literatur, Fail2Ban dapat dimanfaatkan untuk memblokir suatu alamat IP yang terindikasi melakukan pelanggaran jika melebihi ambang batas yang telah ditentukan (Balqis & Rahman, 2025). Penerapan Fail2Ban umum digunakan untuk memblokir serangan DDoS dan *brute force login* SSH (Dawamsyach et al., 2023). Studi lain yang dilakukan oleh Park et al. (2021), menunjukkan bahwa Fail2Ban mampu mendeteksi pola *brute force* melalui protokol SSH dan berhasil memblokir alamat IP penyerang. Pada penelitian ini, Fail2Ban juga dapat diterapkan untuk mendeteksi pola *crash* akibat eksploitasi *buffer overflow* atau *format string*. Di mana, pola *crash* suatu proses aplikasi akan tersimpan di *file log* sistem di Linux. Fail2Ban dapat memblokir alamat IP berdasarkan peristiwa *crash* apabila terdapat *log* lain yang menyimpan sumber alamat IP yang melakukan eksploitasi.

Mengingat eksploitasi celah *buffer overflow* dan *format string* dapat berujung pada eksekusi kode arbitrer, penelitian ini dilakukan untuk membatasi dampak dari kondisi tersebut. Penelitian ini merancang mekanisme *hardening* pada Ubuntu Server 24.04 LTS menggunakan AppArmor dan Fail2Ban. AppArmor berperan sebagai *Mandatory Access Control* (MAC) yang membatasi ruang lingkup suatu aplikasi terhadap sistem, dalam penelitian ini berfungsi mengurung (*contain*) proses

yang telah dikompromikan agar tidak dapat mengakses sumber daya di Ubuntu Server secara sembarangan (Alviano dan Sestito, 2025). Sementara itu, Fail2Ban berfungsi memantau *file log* aktivitas suatu layanan dan secara otomatis memblokir alamat IP yang terindikasi melakukan serangan (Ridho et al., 2025).

Pada penelitian ini, AppArmor bertindak sebagai aktor utama yang mencegah upaya eksploitasi *buffer overflow* dan *format string* terhadap akses sumber daya sistem yang tidak diizinkan. Di sisi lain, Fail2Ban bertugas untuk memblokir alamat IP penyerang berdasarkan pola eksploitasi *binary* yang menyebabkan *crash*, seperti spam input karakter hingga melebihi batas input yang telah dialokasikan atau kesalahan dalam skrip eksploitasi. Selain itu, Fail2Ban dapat digunakan untuk memblokir upaya pembanjiran koneksi (*TCP connection flooding*) setelah ambang batas tercapai. Dengan demikian, dampak eksploitasi yang dilakukan oleh penyerang ke Ubuntu Server dapat dibatasi di lapisan sistem operasi dan jaringan.

1.2 Perumusan Masalah

Berdasarkan latar belakang masalah, maka diperoleh rumusan masalah pada penelitian ini sebagai berikut:

1. Bagaimana merancang sistem keamanan pada Ubuntu Server menggunakan Fail2Ban dan AppArmor terhadap eksploitasi *buffer overflow* dan *format sstring*?
2. Bagaimana efektivitas Fail2Ban dan AppArmor dalam membatasi dampak eksploitasi *buffer overflow* dan *format string* di Ubuntu Server?

1.3 Batasan Masalah

Berikut adalah beberapa batasan masalah pada penelitian ini agar cakupan penelitian tidak terlalu luas:

1. Celah keamanan yang diuji terbatas pada *buffer overflow* dan *format string*.
2. Alat dan modul keamanan yang dipakai terbatas pada Fail2Ban dan AppArmor.
3. Lingkungan pengujian yang digunakan terbatas pada Ubuntu Server versi 24.04.

4. Objek penelitian ini menggunakan 10 *file ELF binary* yang dirancang memiliki kerentanan dan memiliki variasi alur logika program dan proteksi yang berbeda-beda untuk pengujian celah keamanan, Terdapat 5 *file ELF binary* yang memiliki kerentanan *buffer overflow* dan 5 *file ELF binary* yang memiliki kerentanan *format string*.
5. Mekanisme keamanan tidak mencakup *patching* terhadap file *ELF binary* yang rentan, melainkan berfokus pada proteksi di lapisan sistem operasi dan jaringan pada Ubuntu Server.

1.4 Tujuan dan Manfaat

Sejalan dengan rumusan masalah, berikut adalah beberapa tujuan yang akan dicapai pada penelitian ini:

1. Merancang sistem keamanan pada Ubuntu Server menggunakan Fail2Ban dan AppArmor guna mencegah eksploitasi celah *buffer overflow* dan *format string*.
2. Mengevaluasi efektivitas Fail2Ban dan AppArmor dalam membatasi dampak eksploitasi *buffer overflow* dan *format string* di Ubuntu Server.

Adapun, manfaat yang akan diperoleh dari penelitian ini antara lain:

1. Bagi peneliti keamanan (*security researcher*)

   Penelitian ini diharapkan dapat menambah referensi mengenai rancangan sistem keamanan dalam meminimalisir upaya eksploitasi kerentanan memori melalui celah *buffer overflow* dan *format string*.

2. Bagi pengembang (*application developer*)

   Penelitian ini diharapkan dapat membantu tim IT dalam menyusun strategi mitigasi terhadap aplikasi yang memiliki *memory corruption bugs*, terutama ketika proses *patching* aplikasi belum dapat dilakukan secara langsung. Selain itu, alat dan modul keamanan dapat digunakan secara gratis, sehingga akan menghemat biaya pengeluaran dalam perancangan sistem keamanan.

1.5 Sistematika Penulisan

Dalam melakukan penelitian ini, berikut adalah sistematika penulisan yang digunakan:

A. BAB I PENDAHULUAN

Bab ini memuat penjelasan mengenai penelitian apa dan mengapa penelitian tersebut harus dilakukan. Bagian ini terdiri dari latar belakang masalah, perumusan masalah, tujuan dan manfaat penelitian, dan batasan masalah terkait penelitian ini.

B. BAB II TINJAUAN PUSTAKA

Bab ini memuat rangkuman penelitian yang berhubungan dengan penelitian ini untuk memperjelas perbedaan dan memperoleh aspek keterbaruan. Kajian literatur yang relevan dengan topik penelitian ini juga ditambahkan untuk memperkuat landasan teori.

C. BAB III METODOLOGI PENELITIAN

Bab ini memuat penjabaran terkait bagaimana penelitian ini akan dilakukan. Metodologi pada penelitian ini meliputi studi literatur, perancangan sistem keamanan, pengujian sistem keamanan, evaluasi hasil pengujian, dan dokumentasi penelitian.

D. BAB IV HASIL DAN PEMBAHASAN

Bab ini memuat hasil dan pembahasan pada penelitian yang telah dilakukan. Parameter pengujian dapat didapatkan pada bab ini setelah metodologi penelitian dilakukan dengan benar dan berurutan.

E. BAB V PENUTUP

Bab ini memuat kesimpulan dan saran pada penelitian yang telah dilakukan. Selain itu, penjelasan pada bagian saran akan ditambahkan apabila masih ada kekurangan yang dapat dikembangkan lebih lanjut.

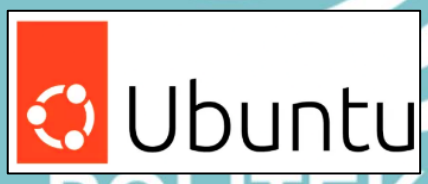
BAB II
TINJAUAN PUSTAKA

2.1 Ubuntu Server

Ubuntu Server dikenal sebagai distribusi Linux berbasis Debian yang dirancang khusus untuk infrastruktur pusat data dan layanan awan. Arsitekturnya yang modular dan sifatnya yang *open-source* memungkinkan kustomisasi sistem secara mendalam tanpa mengorbankan stabilitas operasional (Amrullah et al., 2025).

Sistem operasi ini dipilih dalam penelitian karena dukungan luasnya terhadap modul keamanan kernel Linux seperti AppArmor dan alat audit sistem lainnya. Efisiensi manajemen sumber daya di Ubuntu Server menjadikannya standar dalam pengembangan aplikasi server modern yang membutuhkan tingkat keamanan dan skalabilitas tinggi (Sahara et al., 2024).

Logo Ubuntu tertera pada Gambar 2.1.



Gambar 2. 1 Logo Ubuntu

Sumber: (Ubuntu Handbook, 2022)

2.2 Linux *Application* (ELF *Binary*)

Aplikasi di Linux umumnya dibuat dengan kode bahasa C (Amrullah et al., 2025). Penggunaan *GNU C Library* (glibc) selalu berkaitan dengan praktik pemrograman bahasa C, seperti penggunaan *input/output*, *string*, dan sebagainya. Program kompiler di Linux untuk mengonversi dari kode C menjadi suatu *file binary* adalah *GNU Compiler Collection* (GCC) (ÇeliK dan Dal, 2022). *File binary* hasil kompilasi tersebut berformat *Executable and Linkable Format* (ELF). *File binary* tersebut memerlukan hak eksekusi untuk dapat dijalankan di sistem operasi berbasis Linux.

2.2.1 Bahasa C

Sebagian besar aplikasi atau *binary* yang tersedia pada sistem operasi distribusi Linux, seperti Ubuntu dan Debian, dibangun menggunakan bahasa pemrograman C, sejalan dengan arsitektur sistem operasi berbasis Linux yang sebagian besar komponennya ditulis dalam bahasa tersebut (Amrullah et al., 2025). Bahkan, *kernel* Linux itu sendiri ditulis dan dikembangkan menggunakan bahasa C (Panter dan Eisty, 2024). Bahasa C banyak digunakan karena memberikan keleluasaan dalam mengakses dan mengelola memori secara langsung, tetapi keleluasaan tersebut justru menjadi sumber kerentanan, karena bahasa C tidak menyediakan pemeriksaan batas (*bounds checking*) secara otomatis dan bergantung sepenuhnya pada kedisiplinan pengembang (National Security Agency USA, 2022). Akibatnya, kesalahan pengelolaan memori pada program berbahasa C dapat memicu kerentanan seperti *buffer overflow* dan *format string*.

Secara garis besar, kode program berbahasa C harus melalui proses kompilasi terlebih dahulu sebelum dapat dieksekusi. Proses ini mengubah kode sumber (*source code*) yang bersifat *human-readable* menjadi *file binary* dalam format *executable* yang dapat dijalankan oleh sistem. Kode C yang telah dikonversi menjadi *file binary* sudah tidak mudah dibaca oleh manusia, tetapi *binary* tersebut masih berpotensi dibongkar kembali menggunakan teknik *reverse engineering* untuk menemukan kerentanannya. Pada sistem operasi berbasis Linux, *file binary* umumnya dijalankan dengan perintah *"./binary_name"* dan hanya dapat berjalan apabila telah memiliki izin eksekusi (*execute permission*).

Logo bahasa pemrograman C ditunjukkan pada Gambar 2.2.

Gambar 2. 2 Logo Bahasa Pemrograman C

Sumber: (Sharma, 2026)

2.2.2 *GNU Compiler Collection* (GCC)

Kompiler berfungsi untuk menerjemahkan kode bahasa pemrograman menjadi aplikasi atau bahasa mesin (Adiyoso dan Susetyo, 2023). Salah satu kompiler yang bersifat *open-source* dan populer adalah *GNU Compiler Collection* (GCC). GCC adalah alat kompiler yang dikembangkan oleh Richard Stallman pada tahun 1984, yang juga merupakan *founder* dari *GNU Project* (ÇeliK dan Dal, 2022).

Awalnya, GCC hanya mendukung kompilasi bahasa pemrograman C saja, tetapi saat ini sudah mendukung banyak bahasa pemrograman, seperti C++. *GCC* juga telah mendukung banyak arsitektur prosesor komputer, seperti ARM64, MIPS64, POWER64, s390x, RISC-V, rv64gc, dan x86_64 (64-bit) (ÇeliK dan Dal, 2022) .

Menurut Butt et al. (2022), GCC menyediakan mekanisme proteksi memori yang krusial, seperti *Stack Smashing Protector* (SSP) atau *Stack Canary* dan *Position Independent Executable* (PIE). Fitur-fitur bawaan ini secara signifikan mempersulit penyerang dalam melakukan eksploitasi kerentanan *memory corruption*, seperti *buffer overflow* pada aplikasi yang berjalan di lingkungan Linux.
Logo *GNU Compiler Collection* (GCC) tertera pada Gambar 2.3.

Gambar 2. 3 Logo *GNU Compiler Collection* (GCC)

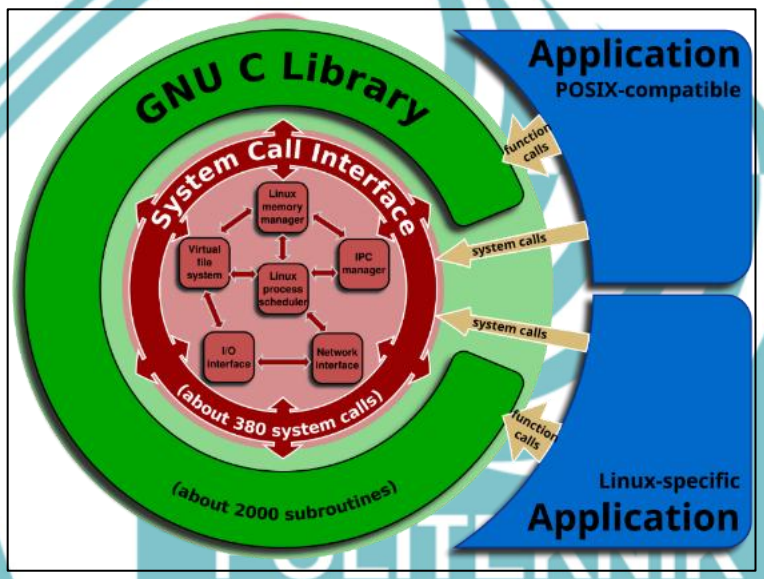
Sumber: (Machado, 2024)

2.2.3 *GNU C Library* (*glibc*)

Banyak program atau *binary* di sistem operasi distribusi Linux, baik PC atau server menggunakan *GNU C Library* (*glibc*) sebagai standar *runtime library* (T. Xie et al.,

2016). Hal itu karena glibc memiliki fungsi-fungsi standar, seperti masukan/keluaran (*input/output*) untuk menangani input pengguna, berbagai operasi string, fungsi operasi matematika, dan sebagainya. Hal yang sering tidak disadari oleh pengembang aplikasi adalah bahwa sebagian fungsi tersebut tidak melakukan pemeriksaan batas secara otomatis, sehingga penggunaannya yang tidak hati-hati dapat membuka celah *buffer overflow* maupun *format string*.

Keterkaitan antara penggunaan *GNU C Library* (*glibc*) dengan *file binary* di Linux ditunjukkan pada Gambar 2.4.



Gambar 2. 4 Keterkaitan *GNU C Library* dengan Aplikasi di Linux

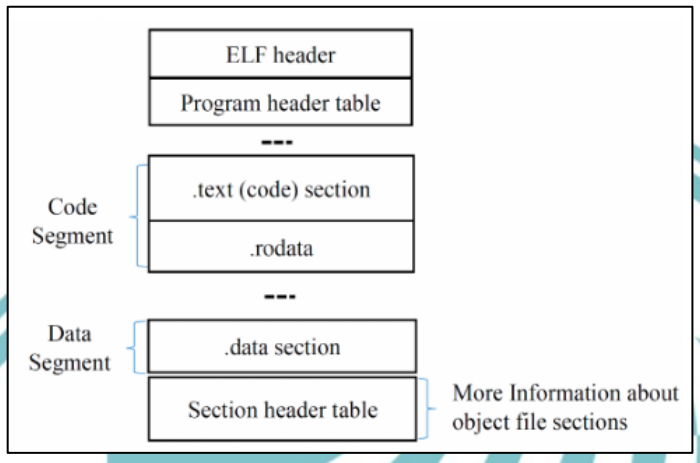
Sumber: (Machado, 2024)

2.2.4 *File Executable and Linkable Format* (ELF)

Executable and Linkable Format (disingkat *ELF*) merupakan format aplikasi atau program pada sistem operasi berbasis Linux yang mendefinisikan struktur instruksi *assembly* (hasil kompilasi program dari bahasa C), data, dan metadata aplikasi. *File ELF binary* modern dilengkapi dengan berbagai mekanisme mitigasi bawaan untuk mencegah eksploitasi kerentanan pada memori (Butt et al., 2022). Namun,

kombinasi teknik eksploitasi tertentu masih memungkinkan penyerang untuk melewati mitigasi tersebut.

Struktur *file ELF binary* di Linux hasil kompilasi bahasa pemrograman C tertera pada Gambar 2.5.



Gambar 2. 5 Struktur File ELF binary

Sumber: (Kyi et al., 2019)

2.3 *Global Offset Table* (GOT) dan *Procedure Linkage Table* (PLT)

Selain bagian kode dan metadata, *file ELF binary* juga menyimpan dua bagian penting yang saling berkaitan, yaitu *Global Offset Table* (*.got*) dan *Procedure Linkage Table* (*.plt*). GOT berfungsi menyimpan alamat memori aktual dari fungsi atau simbol eksternal yang diimpor dari pustaka bersama seperti glibc, sedangkan PLT bertugas sebagai "*trampoline*", yaitu kumpulan potongan instruksi yang mengarahkan pemanggilan fungsi eksternal menuju entri yang sesuai di GOT (Lu et al., 2021).

Pada dasarnya, sebuah *file binary* tidak menyimpan alamat absolut fungsi eksternal dari glibc secara langsung, karena alamat tersebut baru diketahui saat program dimuat ke memori. Oleh sebab itu, ketika suatu fungsi eksternal dipanggil untuk pertama kali, *dynamic linker* akan menyelesaikan (*resolve*) alamat aktual fungsi tersebut, lalu menuliskannya ke entri GOT yang bersesuaian. Pada pemanggilan berikutnya, program dapat langsung mengambil alamat fungsi dari GOT tanpa perlu proses resolusi ulang. Mekanisme penundaan resolusi alamat ini disebut *lazy binding*, dan merupakan perilaku *default* pada *file binary* (Petrean dan Coleşa, 2024).

Keterkaitan GOT dan PLT inilah yang menjadikan keduanya relevan dengan teknik eksploitasi. Karena *lazy binding* mengharuskan entri GOT bersifat dapat ditulisi (*writable*) selama program berjalan, penyerang yang memiliki kemampuan baca-tulis memori (misalnya melalui celah *format string*) dapat menimpa alamat fungsi pada GOT sehingga alur eksekusi dapat dibajak menuju alamat fungsi lain, seperti "*system()*" (Gaidis et al., 2023). Pada penelitian ini, pemahaman terhadap struktur GOT dan PLT menjadi fundamental yang penting, karena beberapa *binary* yang menjadi objek penelitian di antaranya tidak menerapkan mitigasi *Full Relocation Read-Only*, sehingga entri GOT bersifat *writable* dan dapat ditimpa dengan alamat fungsi lain yang berbahaya (istilah teknik serangannya adalah *GOT Overwrite*). Dengan demikian, ketika fungsi yang ada di entri GOT ditimpa dengan fungsi lain untuk memanggil *shell*, maka akses ke dalam sistem dapat diperoleh.

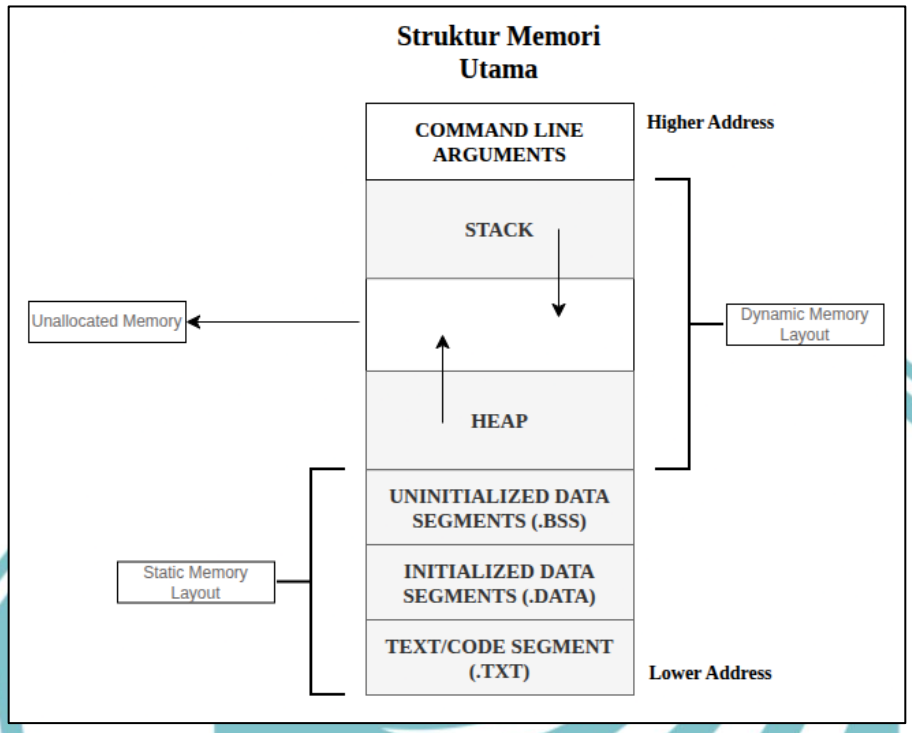
2.4 Struktur Memori Utama

Memori suatu proses pada sistem Linux terbagi menjadi beberapa segmen, di antaranya segmen *text*, *data*, *heap*, dan *stack*. Salah satu karakteristik penting yang perlu dipahami adalah arah pertumbuhan *stack* dan *heap* yang saling berlawanan.

Stack tumbuh dari alamat memori tinggi menuju alamat memori rendah, seperti "0x7fffffffe000" menuju "0x7fffffffd000" (lebih rendah dalam heksadesimal). Sedangkan, *heap* tumbuh sebaliknya, yaitu dari alamat memori rendah menuju alamat memori tinggi.

Pemahaman terhadap arah pertumbuhan ini menjadi penting karena teknik eksploitasi *buffer overflow* memanfaatkan penimpaan data pada *stack* untuk mencapai alamat *return* dengan tujuan membajak alur program ke fungsi lain yang diinginkan.

Gambar 2.6 merupakan struktur memori utama di sistem operasi berbasis Linux.



Gambar 2. 6 Struktur Memori Utama

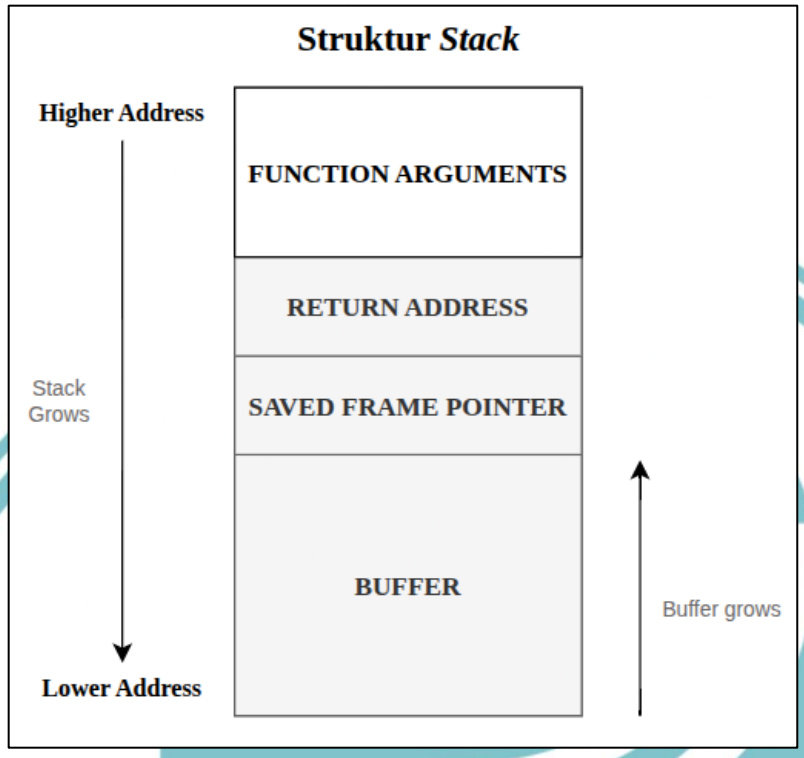
Sumber: (Karim, 2015)

2.5 Struktur *Stack*

Stack merupakan segmen memori yang dikelola secara otomatis untuk menyimpan data sementara, parameter fungsi, dan *return address* selama eksekusi aplikasi. Manajemen *stack* dilakukan menggunakan register khusus, yaitu ESP (32-bit) atau RSP (64-bit) sebagai yang penunjuk ke batas atas *stack frame* dan EBP (32-bit) atau RBP (64-bit) sebagai penunjuk batas *stack frame* (El-Zoghby et al., 2025).

Struktur *stack frame* dibangun melalui proses *prologue* saat fungsi dipanggil dan dihancurkan melalui *epilogue* setelah fungsi selesai dieksekusi. Kerentanan keamanan sering terjadi pada area ini, karena variabel lokal ditempatkan berdekatan dengan *return address*, sehingga luapan data pada *buffer* lokal dapat dimanipulasi untuk membajak atau mengalihkan alur kendali program (Anderson et al., 2023).

Struktur *stack* yang berada di dalam memori utama pada sistem operasi berbasis Linux ditunjukkan pada Gambar 2.7.



Gambar 2. 7 Struktur *Stack*

Sumber: (Kim et al., 2024)

2.6 Linux 64-bit *Calling Convention*

Pada prosesor dengan arsitektur x86-64 (64-bit), pemanggilan fungsi mengikuti aturan *System V AMD64 Application Binary Interface* (ABI), yaitu enam argumen pertama pada fungsi diteruskan melalui register "*rdi*", "*rsi*", "*rdx*", "*rcx*", "*r8*", dan "*r9*" secara berurutan (Lu et al., 2021). Konvensi ini menjadi sangat relevan dalam teknik eksploitasi karena penyerang perlu mengatur nilai register sesuai dengan argumen fungsi yang ingin dipanggil. Untuk keperluan tersebut, penyerang memanfaatkan potongan instruksi yang disebut *gadget*. Misalnya, instruksi "*pop rdi; ret*" yang berfungsi menghapus argumen pertama pada suatu fungsi (*pop rdi*), lalu melanjutkan eksekusi ke alamat berikutnya di *stack* (*ret*). Rangkaian *gadget* inilah yang menjadi dasar teknik *Return Oriented Programming* (ROP) dan *return-to-libc* (*ret2libc*).

2.7 Linux *Binary Protection*

File binary di Linux umumnya dilengkapi dengan mitigasi bawaan demi memitigasi upaya eksploitasi *buffer overflow* dan *format string*. Mitigasi tersebut meliputi *No Execute* (NX) untuk mencegah eksekusi *shellcode* di *stack memory*, *Relocation Read-Only* untuk mencegah teknik *Global Offset Table* (GOT) *Overwrite*, *Position Independent Execution* (PIE) untuk merandomisasi alamat fungsi dan objek lain pada *binary* supaya penyerang tidak mengetahui alamat memori yang aktual, dan terakhir ada *Stack Canary* (*StackGuard*) yang berfungsi mencegah upaya eksploitasi *buffer overflow* (Butt et al., 2022).

2.7.1 *No Execute* (NX)

No Execute (NX) merupakan mekanisme mitigasi yang menandai area memori tertentu, seperti *stack* dan *heap*, sebagai area yang tidak dapat dieksekusi. Mekanisme ini mencegah penyerang mengeksekusi *shellcode* yang disuntikkan langsung ke dalam *stack*, sehingga serangan *buffer overflow* klasik berbasis injeksi kode menjadi tidak efektif. Namun, keberadaan NX mendorong munculnya teknik eksploitasi yang lebih canggih seperti *Return-to-Libc* dan *Return Oriented Programming* yang memanfaatkan kode yang sudah ada di memori (Butt et al., 2022).

2.7.2 *Relocation Read-Only* (RELRO)

Relocation Read-Only (RELRO) merupakan mekanisme yang melindungi *Global Offset Table* (GOT) dari upaya penimpaan alamat atau data di memori (*overwrite*). RELRO memiliki dua tingkatan, yaitu *Partial RELRO* dan *Full RELRO*. Pada *Full RELRO*, seluruh entri GOT diselesaikan saat program dimuat dan kemudian ditandai sebagai *read-only*, sehingga penyerang tidak dapat melakukan teknik *GOT Overwrite*. Sebaliknya, *binary* yang tidak menerapkan RELRO atau hanya menerapkan *Partial RELRO* tetap rentan terhadap teknik tersebut (Butt et al., 2022).

2.7.3 *Position Independent Execution* (PIE)

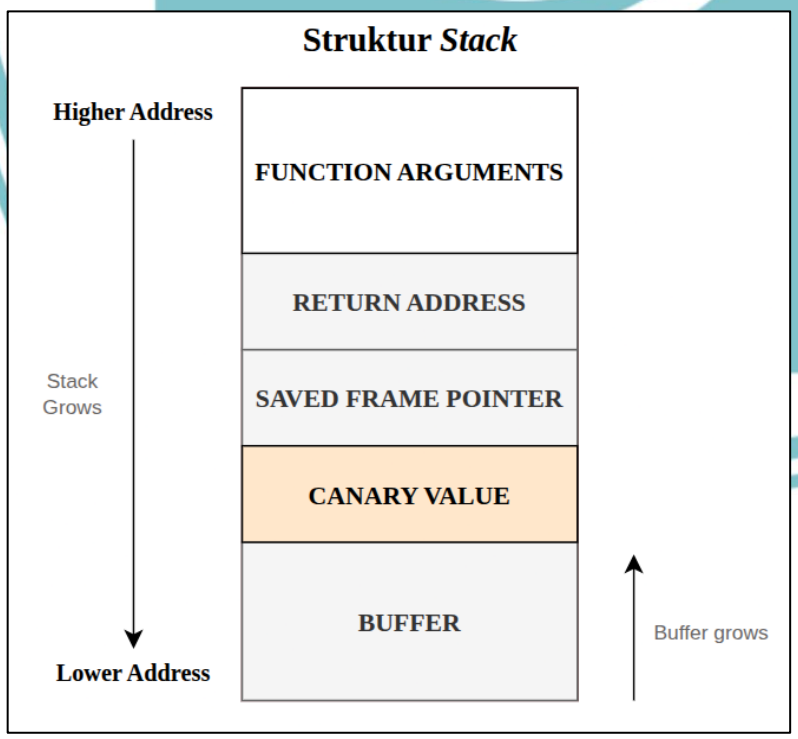
Position Independent Executable (PIE) merupakan mekanisme yang memungkinkan sebuah *binary* dimuat pada alamat memori acak setiap kali dijalankan. Dengan PIE, segmen-segmen *binary* seperti "*.text*" (kode *binary*),

"*.data*" (*initialize data*), dan "*.bss*" (*uninitialized data*) tidak lagi berada pada alamat konstan (dinamis), sehingga mempersulit penyerang dalam memprediksi lokasi instruksi atau data yang menjadi target. PIE bekerja secara berdampingan dengan ASLR untuk memberikan perlindungan yang lebih kuat terhadap eksploitasi (Butt et al., 2022).

2.7.4 *Stack Canary* (*StackGuard*)

Stack Canary atau *StackGuard* merupakan mekanisme proteksi yang menempatkan sebuah nilai penjaga (*canary value*) di antara variabel lokal dan *return address* pada *stack* (Butt et al., 2022). Ketika terjadi *buffer overflow* yang menimpa *stack*, nilai *canary* akan ikut berubah. Sebelum sebuah fungsi mengembalikan kontrol program, sistem memeriksa integritas nilai *canary*. Apabila nilai tersebut telah berubah, program akan dihentikan untuk mencegah eksekusi *return address* yang telah dimanipulasi (Butt et al., 2022). Mekanisme ini efektif menggagalkan serangan *stack smashing*, kecuali apabila penyerang berhasil membocorkan nilai *canary* terlebih dahulu, misalnya melalui celah *format string*.

Nilai *stack canary* yang tersimpan di struktur *stack* ditunjukkan pada Gambar 2.8.



Gambar 2. 8 Posisi *Stack Canary* di *Stack*

2.8 *Address Space Layout Randomization* (ASLR)

Address Space Layout Randomization (ASLR) merupakan mekanisme mitigasi yang bekerja pada level *kernel* Linux dengan cara merandomisasi tata letak ruang alamat suatu proses, termasuk lokasi *stack*, *heap*, dan pustaka bersama (*shared library* (Marco-Gisbert dan Ripoll, 2019).

Berbeda dengan PIE yang merupakan properti dari *binary* itu sendiri, ASLR diterapkan oleh sistem operasi sehingga cakupan randomisasinya lebih luas dan membuat teknik eksploitasi menjadi lebih sulit dilakukan (Marco-Gisbert dan Ripoll, 2019). Dengan randomisasi alamat ini, penyerang tidak dapat lagi mengandalkan alamat memori yang bersifat statis dan harus terlebih dahulu membocorkan alamat (*memory address leak*) untuk dapat menyusun eksploitasi yang andal.

2.9 Kerentanan pada Memori (*Memory Corruption Bugs*)

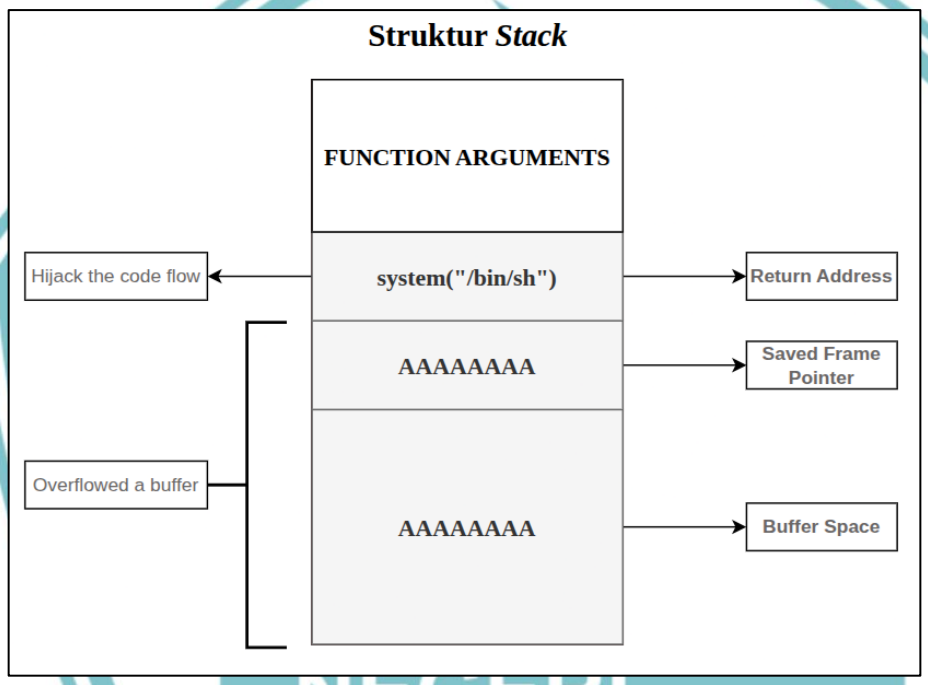
Kerentanan pada memori (*memory corruption bugs*) merupakan kategori celah keamanan yang paling dominan dan berbahaya hingga saat ini. Laporan dari Microsoft menunjukkan bahwa sekitar 70% kerentanan yang diberi CVE setiap tahunnya merupakan isu keamanan memori (Miller, 2019). Sejalan dengan hal tersebut, penelitian Imtiaz dan Williams (2021) menemukan bahwa sekitar 40% kerentanan yang dipublikasikan pada proyek C/C++ berkaitan dengan memori dan memiliki tingkat keparahan yang lebih tinggi dibandingkan kerentanan non-memori. Kedua temuan tersebut menegaskan bahwa kerentanan memori, termasuk *buffer overflow* dan *format string*, masih menjadi ancaman serius yang memerlukan perhatian khusus.

2.9.1 *Buffer Overflow*

Kerentanan *buffer overflow* muncul apabila aplikasi tidak melakukan validasi terhadap batas input pengguna, sehingga data yang dimasukkan melampaui kapasitas buffer yang telah dialokasikan di memori. Penggunaan fungsi pustaka C yang tidak aman, seperti "*gets()*" atau "*strcpy()*", menjadi penyebab utama terjadinya korupsi memori pada *region stack* maupun *heap* (George et al., 2021).

Eksploitasi celah ini memungkinkan penyerang untuk menimpa alamat kembali pada *stack* dengan alamat instruksi berbahaya, yang dapat mengakibatkan pembajakan alur eksekusi program (*control-flow hijacking*). Dampak dari serangan ini mencakup penghentian aplikasi secara paksa (*crash*) hingga pengambilalihan kendali penuh atas sistem operasi (George et al., 2021).

Ilustrasi eksploitasi celah *buffer overflow* yang terjadi di dalam *stack* (*stack-based buffer overflow*) ditunjukkan pada Gambar 2.9.



Gambar 2. 9 Ilustrasi Eksploitasi Celah *Buffer Overflow*

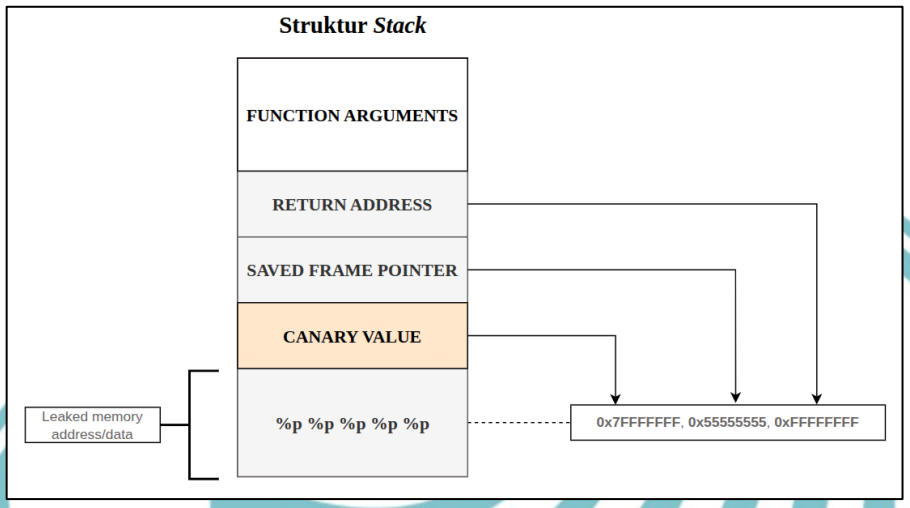
2.9.2 *Format String*

Kerentanan *format string* muncul ketika input dari pengguna digunakan secara langsung sebagai parameter dalam fungsi "*printf()*" di bahasa C. Tanpa adanya penentu format (*format specifier*) yang valid, penyerang dapat memasukkan karakter khusus seperti "*%p*" untuk membaca isi memori atau "*%n*" untuk menulis nilai tertentu ke alamat memori (Xu et al., 2024).

Celah ini sering dimanfaatkan untuk membocorkan nilai rahasia (*information disclosure*), seperti *stack canary* atau alamat basis biner guna melewati proteksi PIE dan ASLR. Dengan kemampuan membaca dan menulis memori secara ilegal,

kerentanan ini menjadi pintu masuk yang efektif bagi serangan korupsi memori yang lebih kompleks (Butt et al., 2022).

Ilustrasi eksploitasi celah *format string* yang terjadi di dalam *stack* tertera pada Gambar 2.10.



Gambar 2. 10 Ilustrasi Eksploitasi Celah *Format String*

2.10 Teknik Eksploitasi Kerentanan pada Memori

Ada banyak teknik yang bisa digunakan untuk melewati mitigasi pada *file binary*. Misalnya, penyerang tidak dapat mengeksekusi *shellcode* yang berhasil diinjeksi ke dalam memori karena ada proteksi *No Execute* (NX), maka penyerang bisa memanfaatkan teknik lain seperti *Return-to-Libc* (*ret2libc*) atau *Return Oriented Programming* (ROP). Jika ada celah *format string* dan proteksi *Relocation Read-Only* (RELRO) tidak ada atau bersifat *partial*, maka teknik *GOT Overwrite* bisa digunakan untuk *read*/*write* data di memori (Butt et al., 2022).

2.10.1 *Return-to-Libc* (ret2libc)

Return-to-Libc (*ret2libc*) merupakan teknik eksploitasi yang muncul sebagai respons terhadap mitigasi NX (Butt et al., 2022). Alih-alih menyuntikkan dan mengeksekusi *shellcode* sendiri yang dapat dicegah oleh mitigasi NX, teknik ini mengarahkan alur eksekusi program ke fungsi yang sudah tersedia di dalam *GNU C Library* (*glibc*), seperti fungsi "*system()*", untuk menjalankan perintah tertentu seperti "*/bin/sh*" (Petrean dan Coleşa, 2024). Dengan demikian, penyerang dapat memperoleh akses *shell* tanpa perlu mengeksekusi kode pada area memori yang dilindungi NX.

2.10.2 *Return Oriented Programming* (ROP)

Return Oriented Programming (ROP) merupakan bentuk lanjutan dari serangan *code-reuse* yang dapat melewati mitigasi NX dengan memanfaatkan potongan-potongan instruksi (*gadget*) yang sudah ada di dalam *binary* (Butt et al., 2022). Setiap *gadget* umumnya diakhiri dengan instruksi ret, sehingga penyerang dapat merangkai beberapa *gadget* secara berurutan untuk membentuk rantai (*ROP chain*) yang menjalankan operasi kompleks sesuai keinginannya (Butt et al., 2022). Teknik ini sangat ampuh terutama ketika mitigasi *stack canary* berhasil dilewati, dan menjadi salah satu metode utama untuk mencapai *arbitrary code execution*.

2.10.3 *Sigreturn Return Oriented Programming* (SROP)

Sigreturn Oriented Programming (SROP) merupakan teknik eksploitasi *code-reuse* yang pertama kali diperkenalkan oleh Bosman dan Bos (2014) dalam publikasi berjudul *"Framing Signals – A Return to Portable Shellcode"*. Berbeda dengan teknik *Return Oriented Programming* (ROP) konvensional, SROP membajak alur eksekusi program dengan memanfaatkan mekanisme penanganan sinyal *(signal handling)* pada sistem operasi berbasis UNIX. Ketika sebuah sinyal ditangani, kernel menyimpan seluruh konteks *register* ke dalam *stack* dalam bentuk *signal frame* (struktur *sigcontext*), kemudian memulihkan (*restore*) seluruh register tersebut saat sinyal selesai diproses melalui *system call sigreturn* (Bosman dan Bos, 2014).

Penyerang memanfaatkan mekanisme tersebut dengan memalsukan *signal frame* pada *stack*, lalu memicu *system call* "*rt_sigreturn*" (bernomor 15 pada arsitektur prosesor 64-bit), sehingga Kernel Linux memuat nilai register sesuai kehendak penyerang, termasuk register *instruction pointer*. Identik dengan teknik ROP, SROP tetap memerlukan *gadget* tambahan untuk memicu eksekusi *system call* tersebut, yang umumnya berupa rangkaian instruksi *"pop rax; ret"* dan *"syscall; ret"*. Yoon dan Lee (2022) menegaskan bahwa SROP tetap menjadi ancaman *code-reuse* yang relevan pada sistem modern, karena memanfaatkan fungsi "*restore_sigcontext()*" yang secara sah digunakan Kernel Linux untuk memproses sinyal.

2.10.4 *Shellcode Injection*

Buffer selain berfungsi menyimpan masukan pengguna secara sementara di memori (RAM) juga berpotensi menyimpan *shellcode*, yaitu rangkaian instruksi mesin yang dirancang untuk memberikan penyerang akses *shell* ke sistem. Teknik penyisipan *shellcode* ke dalam memori dikenal sebagai *shellcode injection* (Butt et al., 2022). Ketika alur eksekusi program berhasil dibajak melalui celah *buffer overflow*, penyerang dapat mengarahkan *instruction pointer* menuju alamat memori *buffer* tempat *shellcode* tersimpan, sehingga instruksi jahat tereksekusi.

Menurut Butt et al., (2022), teknik *shellcode injection* mendorong lahirnya mekanisme mitigasi seperti *No Execute* (NX) atau *Data Execution Prevention* (DEP), yang menjadikan area *stack* bersifat non-eksekusi guna mencegah eksekusi kode yang disisipkan pada *stack*. Namun, apabila mitigasi NX tidak diaktifkan atau dikonfigurasi secara lemah, teknik *shellcode injection* yang dikombinasikan dengan *return-to-shellcode* (*ret2shellcode*) tetap dapat dieksploitasi untuk memperoleh *shell*.

2.10.5 *Open-Read-Write (ORW)*

Open-Read-Write (ORW) merupakan istilah praktis dalam komunitas keamanan untuk teknik yang memanfaatkan *system call* "*open()*", "*read()*", dan "*write()*" secara langsung di dalam proses untuk mengakses, membaca, dan menampilkan isi suatu *file* ke *Terminal* atau *Command Line Interface* (CLI). Berbeda dengan pendekatan yang menjalankan program eksternal seperti *"/bin/cat"*, ORW membaca *file* sepenuhnya melalui *system call* tanpa menjalankan (*spawn*) *shell*, sehingga tetap efektif ketika pemanggilan *shell* dibatasi oleh mekanisme *sandbox* seperti "*seccomp*".

Secara akademik, skenario yang mendasari teknik ORW dibahas dalam literatur mengenai *syscall filtering*. Canella et al. (2021) menjelaskan bahwa penyerang pasca-eksploitasi yang telah memperoleh eksekusi kode arbitrer akan bergantung pada *system call* operasi *file* (*open*, *read*, *write*) maupun eksekusi (*execve*) untuk membaca atau mengedit *file-file* sensitif. Oleh karena itu pembatasan *system call* menjadi lapisan pertahanan penting. Seyedhamed et al. (2020) memperkuat argumen tersebut dengan menunjukkan bahwa pembatasan himpunan *system call*

pada suatu proses secara signifikan mengurangi permukaan serangan *(attack surface)*. Teknik ORW dapat membocorkan informasi sensitif, misalnya *file "/etc/passwd"* yang menyimpan daftar pengguna pada sistem operasi berbasis Linux, maupun *file* penting lainnya.

2.10.6 *Global Offset Table* (GOT) *Overwrite*

GOT Overwrite merupakan teknik eksploitasi yang menargetkan entri GOT pada *binary* yang tidak menerapkan proteksi *Full RELRO*, sehingga dapat dimodifikasi dengan alamat memori yang diinginkan (Gaidis et al., 2023). Dengan memanfaatkan celah baca-tulis memori seperti *format string*, penyerang dapat menimpa alamat fungsi yang tersimpan di GOT sehingga pemanggilan fungsi yang seharusnya sah dialihkan ke alamat lain yang dikendalikan penyerang, misalnya ke fungsi "*system()*" (Petrean dan Coleşa, 2024). Pada penelitian ini, teknik *GOT Overwrite* relevan dengan *binary* ke-2, yaitu "*darkmagic*", yang mana tidak menerapkan mitigasi *Full RELRO* dan PIE.

2.11 *Reverse Engineering*

Reverse engineering merupakan proses menganalisis suatu perangkat lunak untuk memahami struktur, fungsi, dan cara kerjanya tanpa bergantung pada kode sumber aslinya (Chen et al., 2019). Dalam konteks keamanan, *reverse engineering* dimanfaatkan untuk menemukan kerentanan (*vulnerability mining*) pada *file binary* melalui dua pendekatan utama, yaitu analisis statis (*static analysis*) yang mengkaji *binary* tanpa menjalankannya, dan analisis dinamis (*dynamic analysis*) yang mengamati perilaku *binary* saat dieksekusi (Chen et al., 2019). Untuk keperluan analisis statis, umumnya digunakan perangkat lunak *disassembler* dan *decompiler* seperti IDA dan Ghidra yang mampu menerjemahkan instruksi *binary* menjadi *pseudocode* yang lebih mudah dipahami (Rohleder, 2019).

2.12 Fail2Ban

Fail2Ban merupakan alat pencegahan intrusi yang bekerja dengan memantau *file log* sistem untuk mendeteksi pola aktivitas serangan berbasis jaringan. Jika ditemukan pelanggaran yang berulang, sistem akan secara otomatis memblokir IP penyerang melalui pembaruan aturan pada *firewall* (Farhannullah dan Hardjianto, 2022).

Peran Fail2Ban dalam konteks penelitian ini adalah untuk memblokir alamat IP penyerang dalam upaya eksploitasi *buffer overflow* dan *format string* pada ELF *binary*. Dengan begitu, penyerang yang berhasil diblokir tidak bisa mengakses *binary* tersebut melalui jaringan.

Logo alat Fail2Ban ditunjukkan pada Gambar 2.11.

Gambar 2. 11 Logo Fail2Ban

Sumber: (Peppas, n.d.)

2.13 AppArmor

AppArmor adalah modul keamanan kernel Linux yang mengimplementasikan *Mandatory Access Control* (MAC) melalui profil keamanan berbasis jalur (*path-based*). Sistem ini membatasi kapabilitas aplikasi dalam mengakses *file*, jaringan, atau sumber daya sistem lainnya guna meminimalisir dampak jika aplikasi tersebut berhasil dikompromi (Alviano dan Sestito, 2025).

Profil AppArmor dapat dijalankan dalam mode *Enforce* untuk memblokir akses ilegal secara langsung, atau mode *Complain* untuk sekadar mencatat pelanggaran dalam *log* guna analisis awal (Rahman et al., 2026). Dalam memitigasi *buffer overflow*, AppArmor berperan penting dalam membatasi akses suatu ELF *binary* yang berjalan di sisi server agar tidak bisa mengakses *resource* sensitif pada server tersebut, seperti eksekusi *shell* dan mengakses *file* sensitif.

Logo modul AppArmor ditunjukkan pada Gambar 2.12.

Gambar 2. 12 Logo AppArmor

Sumber: (Dickinson, 2019)

2.14 Perangkat Lunak Monitoring

Perangkat lunak *monitoring* merupakan sekumpulan alat yang berfungsi mengumpulkan, menyimpan, memvisualisasikan, dan menganalisis data telemetri dari suatu sistem guna memperoleh gambaran menyeluruh *(observability)* terhadap kondisi, kinerja, dan perilaku sistem tersebut (Pereira dan Sanches, 2025). Data telemetri secara umum terbagi menjadi tiga jenis, yaitu metrik *(metrics)* yang berupa pengukuran numerik dari waktu ke waktu, log *(logs)* yang berupa catatan peristiwa, dan jejak *(traces)*. Kemampuan *observability* memungkinkan pengelola sistem beralih dari pemantauan yang bersifat reaktif menuju pengelolaan yang bersifat proaktif, sehingga anomali dapat terdeteksi lebih dini (Pereira dan Sanches, 2025). Dalam penelitian ini, perangkat lunak *monitoring* digunakan untuk memantau beban sumber daya server serta memvisualisasikan aktivitas serangan dan mekanisme pertahanan yang tercatat pada *log*.

2.14.1 Grafana

Grafana merupakan platform visualisasi dan analitik data *time-series* bersifat *open-source* yang banyak digunakan untuk membangun dasbor *(dashboard)* pemantauan secara *real-time* (Pereira dan Sanches, 2025). Grafana mendukung berbagai sumber data *(data source)*, termasuk Prometheus untuk data metrik dan Loki untuk data log, sehingga memungkinkan korelasi antar-jenis data telemetri dalam satu antarmuka terpadu (Pereira dan Sanches, 2025). Melalui Grafana, pengguna dapat menyusun dasbor interaktif yang menampilkan grafik tren, tabel, maupun indikator status untuk mendukung proses pemantauan dan analisis.

Logo alat monitoring Grafana tertera pada Gambar 3.13.

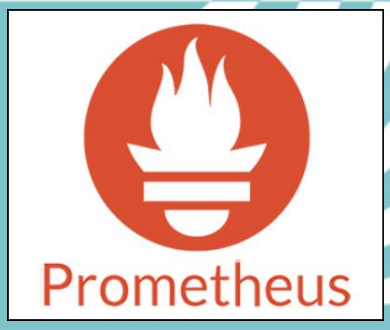
Gambar 2. 13 Logo Grafana

Sumber: (Ansar, 2025)

2.14.2 *Prometheus*

Prometheus merupakan perkakas *(toolkit)* pemantauan sistem dan *alerting* bersifat *open-source* yang menyimpan data dalam bentuk basis data *time-series* (Pereira & Sanches, 2025). Prometheus mengidentifikasi setiap *time series* melalui nama metrik dan sepasang *key-value* yang disebut label, sehingga membentuk model data dimensional. *Prometheus* bekerja dengan model *pull*, yaitu secara berkala menarik *(scrape)* data metrik dari target melalui protokol HTTP pada *endpoint* tertentu (Elradi, 2025). Bahasa kueri PromQL *(Prometheus Query Language)* memungkinkan pengguna melakukan kueri, korelasi, dan transformasi terhadap data *time-series* untuk keperluan visualisasi maupun *alerting*.

Logo Prometheus tertera pada Gambar 3.14.



Gambar 2. 14 Logo Prometheus

Sumber: (Degioanni, 2018)

2.14.3 *Node Exporter*

Node Exporter merupakan *exporter* resmi dari ekosistem Prometheus yang berfungsi mengekspos metrik tingkat perangkat keras dan sistem operasi dari sebuah *host* berbasis Linux (Elradi, 2025). Metrik yang diekspos mencakup penggunaan *Central Processing Unit* (CPU), memori, ruang penyimpanan *(disk)*, serta aktivitas jaringan. *Node Exporter* menyajikan metrik tersebut pada suatu *endpoint* HTTP agar dapat ditarik *(scrape)* oleh Prometheus secara berkala (Elradi, 2025). Dalam konteks pemantauan server, *Node Exporter* menjadi komponen penting untuk memperoleh gambaran beban sumber daya *host*, khususnya ketika server menerima beban serangan seperti *flooding*.

2.14.4 Grafana Loki

Grafana Loki merupakan sistem agregasi log *(log aggregation)* yang bersifat *horizontally scalable*, *highly available*, dan *multi-tenant*. Loki dikembangkan oleh Grafana Labs dengan mengambil inspirasi dari Prometheus (S. Xie et al., 2026). Prinsip desain utama Loki adalah tidak melakukan pengindeksan terhadap keseluruhan isi log, melainkan hanya mengindeks sekumpulan label untuk setiap aliran log *(log stream)*, sehingga menjadikannya hemat biaya penyimpanan dan mudah dioperasikan (S. Xie et al., 2026). Isi *log* disimpan dalam bentuk potongan terkompresi *(chunks)*, sementara metadata berupa label digunakan untuk menemukan aliran log secara efisien. Berbeda dengan Prometheus yang menarik data dengan metode *pull*, Grafana Loki menerima *log* melalui mekanisme *push* dari agen pengumpul. Kueri terhadap log dilakukan menggunakan bahasa LogQL *(Log Query Language)* yang terinspirasi dari PromQL (S. Xie et al., 2026).

Logo Grafana Loki tertera pada Gambar 3.15.

Gambar 2. 15 Logo Grafana Loki

Sumber: (VM, 2024)

2.14.5 Grafana Alloy

Grafana Alloy merupakan distribusi *collector* telemetri bersifat *open-source* yang kompatibel dengan *OpenTelemetry* dan dilengkapi dukungan *native* untuk pipeline Prometheus, serta mendukung pengumpulan metrik, log, jejak, dan *profiles* (Gherghe & Tudose, 2025). Alloy diumumkan pada GrafanaCON 2024 dan merupakan penerus dari Grafana Agent. Sejak Promtail memasuki status *end-of-life*, Grafana Labs merekomendasikan migrasi ke Alloy sebagai agen pengumpul *log*. Alloy menggunakan pendekatan berbasis komponen *(component)*, di mana

setiap komponen menjalankan fungsi tertentu, seperti menemukan *file*, membaca isi *file*, dan mengirimkan data ke *backend* yang saling disusun membentuk suatu pipeline telemetri (Gherghe & Tudose, 2025). Dalam penelitian ini, Alloy berperan sebagai agen yang membaca *file log* pada setiap *host* dan meneruskannya ke Loki untuk keperluan agregasi dan visualisasi.

Logo Grafana Alloy tertera pada Gambar 2.16.

Gambar 2. 16 Logo Grafana Alloy

Sumber: (Lam, 2024)

2.15 CTFd

CTFd merupakan platform *Capture The Flag* (CTF) bersifat *open-source* yang digunakan untuk menyelenggarakan kompetisi keamanan siber, khususnya dalam format *jeopardy*, yaitu format yang menyajikan sekumpulan tantangan *(challenge)* berdasarkan kategori, di mana setiap tantangan menyembunyikan sebuah *flag* berupa teks rahasia yang harus ditemukan dan dikirimkan peserta untuk memperoleh poin (Maulana et al., 2023). CTFd menyediakan fitur seperti papan skor *(scoreboard)*, manajemen tantangan, manajemen pengguna, serta pencatatan aktivitas peserta seperti waktu penyelesaian dan pengiriman *flag*.

Dalam ranah pendidikan dan penelitian keamanan siber, CTFd banyak dimanfaatkan sebagai wadah untuk mengekspos sistem atau *file* rentan kepada peserta guna dieksploitasi secara terkendali (Maulana et al., 2023). Pada penelitian ini, CTFd berperan menggantikan penggunaan *VirtualBox* sebagai media pengujian, dengan menyediakan sepuluh *file* ELF *binary* yang rentan kepada penyerang untuk dieksploitasi. Melalui platform ini, data pengujian dapat dikumpulkan dari aktivitas penyerang yang nyata, sehingga evaluasi efektivitas sistem keamanan menjadi lebih representatif.

Logo platform CTFd tertera pada Gambar 2.17.

Gambar 2. 17 Logo Platform CTFd

Sumber: (Maulana et al., 2023)

2.16 Nginx

Nginx merupakan perangkat lunak *web server* bersifat *open-source* dan berkinerja tinggi, yang mana juga umum difungsikan sebagai *reverse proxy* (Yanto, 2024). Sebagai *reverse proxy*, Nginx bertindak sebagai perantara *(intermediary)* yang menerima permintaan HTTP dari pengguna, lalu meneruskannya ke server *backend* yang sesuai (Yanto, 2024). Nginx pada penelitian berperan untuk meneruskan setiap permintaan HTTP dari pengguna ke platform CTFd dan mengembalikan respons yang sesuai. Dengan pola ini, *backend* tidak terekspos secara langsung ke internet, sehingga Nginx berfungsi sebagai lapisan pelindung sekaligus pengatur lalu lintas.

Selain sebagai *reverse proxy*, Nginx berperan melakukan terminasi TLS *(TLS termination)*, yaitu mengubah permintaan HTTP menjadi HTTPS dengan memanfaatkan sertifikat SSL/TLS yang sah dari pihak ketiga (Yanto, 2024). Melalui HTTPS, data yang dipertukarkan antara pengguna dan server dienkripsi, sehingga kerahasiaan *(confidentiality)* dan integritas *(integrity)* data selama transmisi *(in-transit)* tetap terjaga. Dengan demikian, penggunaan HTTPS memitigasi risiko penyadapan *(eavesdropping)* dan serangan *Man-in-the-Middle* (MitM) terhadap lalu lintas pengguna di *website*.

Logo Nginx *web server* tertera pada Gambar 2.18.

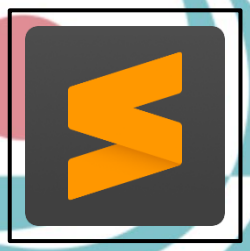
Gambar 2. 18 Logo Nginx

Sumber: (Ansar, 2025)

2.17 Perangkat Lunak Pendukung

Dalam proses perancangan dan pengujian sistem keamanan tidak lepas dari kebutuhan perangkat lunak pendukung. Beberapa perangkat lunak pendukung untuk pembuatan skrip pengujian mencakup Sublime Text, Python3 dengan *library pwntools*, *netcat*, *socat*, *GNU Debugger Enhanced Feature* (GEF), *Terminal Multiplexer* (*tmux*), dan *Integrated Disassembler* (IDA). Selain itu, untuk menjalankan *file ELF binary* bisa menggunakan alat bernama *SOCket CAT* (socat).

2.15.1 Sublime Text



Gambar 2. 19 Logo Sublime Text

Sumber: (Sublime Text, n.d.)

Sublime Text merupakan perangkat lunak untuk menulis, mengedit, dan mengelola kode pemrograman (Hadi et al., 2025). Teks editor Sublime Text sudah mendukung berbagai jenis bahasa pemrograman, seperti PHP, HTML, CSS, JavaScript, Python, dan sebagainya.

2.15.2 Python3

Gambar 2. 20 Logo Bahasa Python

Sumber: (Full Stack Python, n.d.)

Python3 merupakan bahasa pemrograman yang digunakan dalam penelitian ini untuk membangun skrip pengujian (*exploit script*). Dalam pengembangan eksploitasi, Python3 dipadukan dengan *library pwntools*, yaitu sebuah *framework* CTF sekaligus pustaka pengembangan eksploitasi yang dirancang untuk mempercepat proses prototipe dan penulisan eksploitasi (Hanaputra et al., 2023).

2.15.3 *Library Pwntools*

Gambar 2. 21 Logo *Library Pwntools*

Sumber: (OffSec Tools, n.d.)

Pwntools merupakan *library* Python yang dikembangkan untuk mendukung riset keamanan biner dan pembuatan *payload* eksploitasi secara cepat. *Library* ini menyediakan berbagai fungsionalitas otomasi seperti manipulasi data biner (*machine code*), interaksi dengan layanan jaringan, serta analisis struktur *file ELF* ataupun *EXE* pada Windows (Shao et al., 2024).

Integrasi *library pwntools* dengan Bahasa Python dalam pengujian keamanan memungkinkan penyusunan *payload* serangan yang presisi untuk mengevaluasi ketangguhan sistem keamanan. Alat ini digunakan secara luas dalam lingkungan akademik untuk mendalami mekanisme kerentanan memori dan mengembangkan metode pertahanan yang lebih efektif (Shao et al., 2024).

2.15.4 Netcat

Gambar 2. 22 Logo *netcat*

Sumber: (Kali Linux Documentation, n.d.-a)

Netcat (nc) merupakan perangkat lunak jaringan yang berfungsi sebagai pemindaian *port*, kirim pesan melalui jaringan, mengirimkan data atau *file* tanpa memerlukan FTP server, mengidentifikasi versi *web server*, dan yang terpenting mampu melakukan eksploitasi dari jarak jauh (*remote*) dengan *reverse shell* pada server target (Boyanov, 2023). Dalam penelitian ini, netcat digunakan untuk mengakses *binary* yang telah dijalankan di port tertentu pada Ubuntu Server melalui socat, sehingga interaksi dengan *binary* dari sisi penyerang dapat dilakukan.

2.15.5 Socat

Gambar 2. 23 Logo *SOcket CAT* (*socat*)

Sumber: (Kali Linux Documentation, n.d.-b)

SOcket CAT (*socat*) merupakan perangkat lunak yang berfungsi sebagai *relay* untuk transfer data dua arah antara dua kanal data independen (Rieger, n.d.). Dalam penelitian ini, *socat* digunakan untuk menjalankan *file ELF binary* pada Ubuntu Server dengan cara mengeksposnya sebagai layanan jaringan berbasis TCP, sehingga *binary* tersebut dapat diakses dari jarak jauh untuk keperluan pengujian.

2.15.6 GNU Debugger *Enhanced Feature* (GEF)

Gambar 2. 24 Logo GEF

Sumber: (GEF Documentation, n.d.)

GDB Enhanced Features (GEF) merupakan perangkat lunak *debugger* berbasis *GNU Debugger* (GDB) yang digunakan untuk menganalisis *binary* secara dinamis (*dynamic analysis*) (GEF Documentation, n.d.). Dalam penelitian ini, GEF dimanfaatkan untuk mengamati kondisi register, *stack*, dan memori saat *binary* dijalankan di lingkungan yang rentan, sehingga memudahkan proses penyusunan eksploitasi.

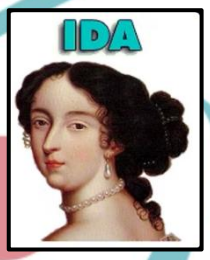
2.15.7 *Terminal Multiplexer* (tmux)

Gambar 2. 25 Logo *Terminal Multiplexer* (*tmux*)

Sumber: (Full Stack Python, n.d.-a)

Terminal Multiplexer (*tmux*) merupakan perangkat lunak yang memungkinkan pengelolaan beberapa sesi terminal dalam satu jendela secara bersamaan. Dalam penelitian ini, tmux digunakan untuk mempermudah pengelolaan beberapa proses sekaligus, misalnya menjalankan *debugger* dan skrip eksploitasi secara berdampingan.

2.15.8 *Interactive Disassembler Free Edition* (IDA)



Gambar 2. 26 Logo *Interactive Disassembler* (IDA)

Sumber: (HackerArise, 2023)

Interactive Disassembler (IDA) merupakan perangkat lunak *disassembler* dan *decompiler* yang digunakan untuk membaca serta menganalisis statis *file ELF binary* (*reverse engineering*) (Rohleder, 2019). Dalam penelitian ini, IDA dimanfaatkan untuk menerjemahkan *file ELF binary* kembali (dekompilasi) menjadi *pseudocode*, sehingga kerentanan *buffer overflow* dan *format string* dapat diidentifikasi.

2.18 Penelitian Terkait

Berikut adalah hasil studi literatur terhadap 5 penelitian terkait yang dijadikan sebagai rujukan utama untuk mendapatkan keterbaruan penelitian. Hasil studi literatur tertera pada Tabel 2.1.

Tabel 2. 1 Hasil Studi Literatur pada Penelitian Terkait

No.	Judul dan Penulis	Penelitian yang Dilakukan	Perbedaan Penelitian
1.	*A Buffer Overflow Detection and*	Mengembangkan metode deteksi dan pertahanan *buffer overflow* melalui	**Tujuan:** Proteksi *buffer overflow* di level arsitektur prosesor RISC-V, sedangkan

	Defense Method Based on RISC-V Instruction Set Extension (Liu et al., 2023)	ekstensi *instruction set* RISC-V, sehingga proteksi diterapkan pada level perangkat keras (arsitektur prosesor).	penelitian ini berfokus pada *hardening* server di lapisan sistem operasi. **Lingkungan Pengujian:** Arsitektur RISC-V, sedangkan penelitian ini pada Ubuntu Server 24.04. **Objek Penelitian**: Ekstensi *instruction set*, sedangkan penelitian ini menggunakan 10 *file ELF binary* yang dirancang memiliki celah *buffer overflow dan format string*. **Mekanisme Keamanan**: Modifikasi instruction set, sedangkan penelitian ini menggunakan Fail2Ban dan AppArmor.
2.	*Identifikasi Kerentanan Perangkat Lunak Pengolah Dokumen Berbasis Binary Menggunakan Metode Whitebox*	Menemukan kerentanan *buffer overflow* pada perangkat lunak PDFCook versi 0.4.5 menggunakan teknik *fuzzing* (AFL++) yang dikombinasikan dengan *Address Sanitizer* (ASan).	**Tujuan**: Menemukan atau mengidentifikasi kerentanan melalui *fuzzing*, sedangkan penelitian ini berfokus pada *hardening* server terhadap eksploitasi. **Lingkungan Pengujian**: Tidak spesifik pada *hardening* server, sedangkan penelitian ini pada Ubuntu Server 24.04.

	Fuzzing AFL++ (Rasyid dan Santoso, 2026)		**Objek Penelitian**: PDFCook v0.4.5, sedangkan penelitian ini menggunakan 10 *file ELF binary* yang dirancang memiliki celah *buffer overflow* dan *format string*. **Mekanisme Keamanan**: tidak menerapkan mekanisme pertahanan (hanya deteksi), sedangkan penelitian ini menggunakan Fail2Ban dan AppArmor.
3.	*PwnMaster: Automatic Buffer Overflow and Format String Vulnerability Detection and Exploitation* (Petrean dan Coleşa, 2024)	Merancang alat otomatis bernama "PwnMaster" untuk mendeteksi sekaligus menghasilkan skrip eksploitasi *buffer overflow* dan *format string* pada *file* ELF menggunakan *symbolic execution* dan *pwntools*.	**Tujuan:** deteksi dan pembuatan skrip eksploitasi otomatis, sedangkan penelitian ini berfokus pada *hardening* server terhadap eksploitasi kedua celah tersebut. **Lingkungan Pengujian:** tidak berfokus pada proteksi server, sedangkan penelitian ini pada Ubuntu Server 24.04. **Objek Penelitian**: *file ELF binary* yang dikustomisasi, sedangkan penelitian ini menggunakan 10 *file ELF binary* yang dirancang

			memiliki celah *buffer overflow* dan *format string*. **Mekanisme Keamanan**: tidak menerapkan mekanisme pertahanan, sedangkan penelitian ini menggunakan Fail2Ban dan AppArmor.
4.	*BofAEG: Automated Stack Buffer Overflow Vulnerability Detection and Exploit Generation Based on Symbolic Execution and Dynamic Analysis* (Xu dan Wang, 2022)	Membangun sistem *automated exploit generation* (BofAEG) untuk mendeteksi dan menghasilkan eksploitasi *stack buffer overflow* secara otomatis berbasis *symbolic execution* dan analisis dinamis.	**Tujuan**: Mendeteksi dan pembuatan eksploitasi *buffer overflow* otomatis, sedangkan penelitian ini berfokus pada *hardening* server terhadap eksploitasi *buffer overflow* dan juga *format string*. **Lingkungan Pengujian**: Tidak berfokus pada proteksi server, sedangkan penelitian ini pada Ubuntu Server 24.04. **Objek Penelitian**: *File ELF binary* dari kompetisi CTF dan CVE secara umum, sedangkan penelitian ini menggunakan 10 *file ELF binary* yang dirancang memiliki celah *buffer overflow* dan *format string*. **Mekanisme Keamanan**: tidak menerapkan

			mekanisme pertahanan, sedangkan penelitian ini menggunakan Fail2Ban dan AppArmor.
5.	*StackGuard⁺: Interoperable Alternative to Canary-Based Protection of Stack Smashing* (Kim et al., 2024)	Mengembangkan metode *StackGuard⁺* sebagai alternatif interoperabel terhadap proteksi berbasis *canary* konvensional untuk mencegah *stack smashing* pada aplikasi berbasis C.	**Tujuan:** Meningkatkan mekanisme proteksi *stack* pada level aplikasi, sedangkan penelitian ini berfokus pada *hardening* server di lapisan sistem operasi. **Lingkungan Pengujian**: tidak berfokus pada proteksi server, sedangkan penelitian ini pada Ubuntu Server 24.04. **Objek Penelitian**: aplikasi berbasis C secara umum, sedangkan penelitian ini menggunakan 10 *file ELF binary* yang dirancang memiliki celah *buffer overflow* dan *format string*. **Mekanisme Keamanan**: modifikasi mekanisme *stack canary* (*StackGuard⁺*), sedangkan penelitian ini menggunakan Fail2Ban dan AppArmor.

6.	*Implementasi IPS (Intrusion Prevention System) Fail2ban pada Server terhadap Serangan DDoS dan Brute Force* (Dawamsyach et al., 2023)	Penelitian ini bertujuan meningkatkan keamanan server dengan mengimplementasikan Fail2Ban sebagai *Intrusion Prevention System* (IPS) terhadap serangan *brute force* pada *port* 22 (SSH) dan DDoS pada *port* 80. Sistem dilengkapi antarmuka website serta notifikasi Telegram, dan menggunakan aturan *jail* untuk mendeteksi serta memblokir alamat IP. Pengujian mengukur waktu deteksi, notifikasi, dan dampak serangan terhadap kinerja server. Hasilnya, Fail2Ban terbukti mampu mendeteksi dan memblokir IP penyerang berdasarkan log, dengan serangan DDoS berdampak paling besar pada CPU dan *brute force* pada memori.	**Tujuan:** Penelitian terdahulu berfokus pada mitigasi *brute force* SSH dan DDoS pada layanan umum, sedangkan penelitian ini berfokus pada pembatasan dampak eksploitasi *buffer overflow* dan *format string*. **Lingkungan Pengujian**: Penelitian terdahulu tidak spesifik pada Ubuntu Server 24.04 LTS dengan skema CTF multi-VM sebagaimana penelitian ini. **Objek Penelitian**: Penelitian terdahulu menguji layanan SSH dan HTTP, sedangkan penelitian ini menguji 10 *file ELF binary* yang dirancang memiliki celah *buffer overflow* dan *format string*. **Mekanisme Keamanan**: Penelitian terdahulu hanya menggunakan Fail2Ban, sedangkan penelitian ini mengombinasikan Fail2Ban dengan AppArmor sebagai pertahanan berlapis.
7.	*Network Log-Based SSH*	Penelitian ini membangun model deteksi serangan	**Tujuan**: Penelitian terdahulu berfokus pada pemodelan

	Brute-Force Attack Detection Model (Park et al., 2021)	SSH *brute-force* berbasis log jaringan, dengan menekankan bahwa perangkat pada lapisan atas *firewall* sulit dilindungi sehingga rentan terhadap serangan. Metode yang digunakan adalah mengekstraksi dan memfragmentasi data spesifik dari paket yang dikumpulkan untuk mendeteksi upaya login SSH bertubi-tubi, kemudian mengumpulkan alamat IP penyerang dan melakukan kontrol akses (pemblokiran) terhadap IP tersebut. Pendekatan ini menegaskan prinsip deteksi berbasis *log* yang dilanjutkan dengan pemblokiran IP, yang menjadi dasar kerja Fail2Ban.	deteksi *brute force* SSH, penelitian ini berfokus pada pembatasan dampak eksploitasi memori (*buffer overflow* dan *format string*). **Lingkungan Pengujian**: Penelitian terdahulu berbasis analisis log atau paket jaringan pada lingkungan router, sedangkan penelitian ini pada Ubuntu Server 24.04 LTS dengan platform CTF. **Objek Penelitian**: Penelitian terdahulu menargetkan layanan SSH, sedangkan penelitian ini menguji 10 *file ELF binary* yang dirancang memiliki celah *buffer overflow* dan *format string*. **Mekanisme Keamanan**: Penelitian terdahulu membangun model deteksi berbasis log secara mandiri, sedangkan penelitian ini menggunakan Fail2Ban yang dikombinasikan dengan AppArmor (*Mandatory Access Control*).

BAB III
METODE PENELITIAN

3.1 Rancangan Penelitian

Metodologi penelitian ini menggunakan pendekatan kualitatif dengan jenis penelitian eksperimental (Creswell dan Creswell, 2022). Pendekatan kualitatif dipilih karena fokus utama penelitian adalah merancang dan mengimplementasikan sistem keamanan pada Ubuntu Server menggunakan AppArmor dan Fail2Ban terhadap eksploitasi *buffer overflow* dan *format string*. Selain itu, penelitian ini mengukur tingkat efektivitas kedua mekanisme keamanan tersebut dalam memproteksi Ubuntu Server dari upaya eksploitasi. Untuk menjawab rumusan masalah pertama, penelitian merancang dan mengimplementasikan mekanisme keamanan pada lingkungan pengujian terkendali, yaitu di mesin virtual. Adapun untuk menjawab rumusan masalah kedua, efektivitas mekanisme keamanan dievaluasi melalui serangkaian skenario pengujian yang hasilnya disajikan dalam bentuk data deskriptif dan dianalisis secara kualitatif. Jenis penelitian eksperimental digunakan karena penelitian melakukan penerapan tingkat proteksi yang berbeda pada lingkungan pengujian, kemudian mengamati dan mendeskripsikan respons sistem terhadap eksploitasi celah *buffer overflow* dan *format string*.

Pengujian sistem keamanan dilakukan dengan pendekatan *grey box testing*, di mana penyerang hanya akan memperoleh 10 *file ELF binary* tanpa mengetahui kode sumber asli programnya (Zhang et al., 2023). Dalam kasus ini, penyerang akan menggunakan teknik *reverse engineering* untuk membongkar *file binary* tersebut untuk memahami alur logika dan menemukan kerentanannya. Kesepuluh *file binary* tersebut diekspos pada platform CTFd agar bisa diunduh dan diakses oleh para penyerang. Berbagai pelanggaran di lapisan sistem operasi yang dilakukan oleh penyerang saat mengeksploitasi *binary* akan dideteksi oleh AppArmor dan upaya tersebut akan ditolak. Apabila penyerang berupaya menyerang dari sisi jaringan, seperti membanjiri koneksi dan spam karakter secara masif atau kesalahan pada skrip eksploitasi secara berulang hingga menyebabkan *crash* pada aplikasi, maka Fail2Ban akan memblokir alamat IP penyerang. Dengan demikian, akses penyerang ke *binary* dan server target melalui jaringan akan terputus.

3.2 Tahapan Penelitian

Tahapan penelitian ini disusun secara sistematis untuk menjawab rumusan masalah, mulai dari studi literatur hingga analisis dan evaluasi data. Adapun tahapan penelitian adalah sebagai berikut:

1. Studi Literatur

   Tahap awal penelitian adalah mengumpulkan studi literatur yang relevan terkait metode deteksi celah *buffer overflow* dan *format string*, teknik eksploitasi kerentanan memori, serta mekanisme keamanan yang telah diterapkan pada penelitian terdahulu. Studi literatur ini menjadi dasar dalam menentukan celah penelitian (*research gap*) sekaligus landasan teoretis bagi perancangan sistem keamanan.

2. Perancangan Arsitektur Mesin Virtual

   Pada tahap ini dirancang arsitektur lingkungan pengujian berupa empat mesin virtual *(virtual private server)* berbasis Ubuntu Server 24.04 LTS. Tiga mesin pertama berfungsi sebagai *host* tantangan dengan tingkat proteksi yang berbeda, yaitu VM-1 yang dikonfigurasi tanpa mekanisme proteksi tambahan *(no protection)*, VM-2 yang dikonfigurasi dengan AppArmor melalui dua jenis profil (longgar dan ketat), serta VM-3 yang dikonfigurasi dengan AppArmor dan Fail2Ban secara bersamaan. Adapun mesin keempat, yaitu VM-CTF, berfungsi sebagai *host* platform kompetisi CTFd sekaligus pusat pemantauan *(monitoring)* yang menjalankan Grafana, Prometheus, dan Loki. Perbedaan tingkat proteksi pada ketiga mesin tantangan bertujuan untuk membandingkan efektivitas masing-masing mekanisme keamanan.

3. Perancangan Mekanisme Keamanan

   Tahap ini merupakan inti dari penelitian, yaitu perancangan mekanisme *hardening* pada Ubuntu Server. Kegiatan pada tahap ini meliputi perancangan profil AppArmor untuk mengurung *(contain)* ruang lingkup *binary* yang diuji, serta konfigurasi Fail2Ban beserta filter, *jail*, dan *action* untuk memblokir alamat IP penyerang. Fail2Ban dirancang untuk

memblokir alamat IP berdasarkan aktivitas serangan pada sisi jaringan yang memuat alamat IP sumber, yaitu upaya pembanjiran koneksi *(TCP connection flooding)* dan upaya merusak *binary* hingga menyebabkan galat *(crash)*. Sebagai penghubung, dirancang sebuah *wrapper* pada layanan socat yang mencatat alamat IP sumber koneksi *(SOCAT_PEERADDR)* beserta sinyal keluaran *(exit status)* proses ke dalam *file log*. Dengan demikian, entri log yang dihasilkan telah memuat alamat IP penyerang sehingga dapat dibaca dan ditindaklanjuti oleh Fail2Ban. Perlu ditegaskan bahwa AppArmor dan Fail2Ban memiliki peran yang terpisah, di mana AppArmor menolak *(deny)* operasi tidak sah pada lapisan sistem operasi tanpa mengenali alamat IP, sedangkan Fail2Ban memblokir alamat IP pada lapisan jaringan berdasarkan log yang memuat alamat IP.

4. Pengujian Sistem Keamanan

Pengujian sistem keamanan dilakukan melalui platform kompetisi CTFd. Kesepuluh *file ELF binary* yang rentan diekspos pada platform tersebut agar dapat diunduh dan dieksploitasi oleh para penyerang. Skrip eksploitasi (*exploit script*) dikembangkan sepenuhnya oleh penyerang, yang terlebih dahulu menganalisis *binary* melalui teknik *reverse engineering* untuk memahami alur logika dan menemukan kerentanannya. Setiap penyerang melakukan upaya eksploitasi dari alamat IP masing-masing, sehingga data pengujian diperoleh dari aktivitas penyerang yang nyata pada ketiga mesin tantangan (VM-1, VM-2, dan VM-3).

Selain eksploitasi celah memori, dilakukan pula pengujian pada sisi jaringan berupa pembanjiran koneksi (*TCP connection flooding*), yaitu membuka koneksi TCP secara berulang dalam jumlah besar ke salah satu layanan *binary* untuk menguji apakah Fail2Ban mendeteksi dan memblokir alamat IP penyerang ketika ambang batas (*threshold*) tercapai. Kompetisi diselenggarakan selama enam jam, yaitu mulai dari pukul 17:00 WIB hingga 23:59 WIB, dengan papan skor (*scoreboard*) akan dibekukan (*freeze time*) satu jam sebelum kompetisi berakhir, yaitu pada pukul 22:00 WIB. Sebagaimana aturan kompetisi CTF pada umumnya, peserta dilarang

merusak platform CTFd, misalnya melalui serangan *Distributed Denial of Service* (DDoS), maupun melakukan pelanggaran berupa berbagi *flag* (*sharing flag*) antar-peserta.

5. Analisis Data dan Evaluasi

   Tahap akhir adalah menganalisis data hasil pengujian untuk mengevaluasi kinerja Fail2Ban dan AppArmor. Analisis dilakukan dengan membandingkan keberhasilan eksploitasi, kemampuan AppArmor dalam mengurung proses eksploitasi yang berupaya mengakses *shell* ataupun membaca *file* sensitif, serta kemampuan Fail2Ban dalam memblokir alamat IP penyerang pada keempat mesin virtual. Salah satu parameter penting yang dievaluasi adalah selisih antara waktu serangan (*attack time*) dan waktu pemblokiran (*blocking time*). Waktu terjadinya serangan adalah waktu ketika penyerang melakukan upaya eksploitasi atau pembanjiran koneksi. Sedangkan, waktu pemblokiran adalah waktu ketika Fail2Ban memblokir alamat IP penyerang. Selisih kedua waktu tersebut merepresentasikan responsivitas Fail2Ban dalam menindak serangan. Data hasil pengujian bersumber dari *file log* pada tiap mesin yang kemudian diagregasi dan divisualisasikan melalui dasbor Grafana untuk mempermudah analisis.

3.3 Objek Penelitian

Objek penelitian ini adalah celah keamanan *buffer overflow* dan *format string*. Terdapat 10 file ELF *binary* yang digunakan untuk penelitian, yang terbagi menjadi dua kelompok, yaitu lima *binary* dengan celah *buffer overflow* (BOF-1 hingga BOF-5) dan lima *binary* dengan celah *format string* (FMT-1 hingga FMT-5). Kesepuluh *binary* tersebut telah memiliki celah secara *by design* sebagai representasi dari lemahnya praktik *secure coding*, dan dikombinasikan dengan konfigurasi mitigasi bawaan yang bervariasi untuk menghadirkan beragam tingkat kesulitan pengujian.

Setiap *binary* juga memiliki alur logika program dan konfigurasi proteksi bawaan yang berbeda-beda, seperti *No Execute* (NX), *Relocation Read-Only* (RELRO), *Position Independent Executable* (PIE), dan *Stack Canary*. Perlu ditegaskan bahwa variasi mitigasi bawaan tersebut diperlakukan sebagai dimensi tingkat kesulitan tantangan yang disengaja dalam desain kompetisi, bukan sebagai variabel yang diukur. Adapun yang dievaluasi dalam penelitian ini adalah efektivitas lapisan proteksi sistem operasi (AppArmor dan Fail2Ban) yang ditambahkan di atas *binary*, dengan objektif tantangan yang sama pada setiap *binary*, yaitu memperoleh akses ke sistem dan membaca *file flag*. Dengan demikian, penelitian tidak membandingkan tingkat kesulitan antar-*binary*, melainkan mengamati respons lapisan proteksi terhadap upaya eksploitasi.

Detail dari masing-masing *file ELF binary* yang akan dijadikan sebagai objek pengujian tertera pada Tabel 3.1.

Tabel 3. 1 Daftar Objek Penelitian

No	Nama *ELF Binary*	Jenis Celah	Proteksi Bawaan	Proteksi Tambahan
1.	*BOF-1*	*Buffer Overflow*	• Full RELRO • *No Execute* (NX) aktif • PIE tidak aktif • *Stack Canary* aktif	-
2.	*FMT-1*	*Format String*	• *Partial* RELRO • *No Execute* (NX) aktif • PIE tidak aktif • *Stack Canary* aktif	-

3.	*BOF-2*	*Buffer Overflow*	• Full RELRO • *No Execute* (NX) aktif • PIE tidak aktif • *Stack Canary* aktif	AppArmor (Profil Longgar)
4.	*FMT-2*	*Format String*	• *Partial* RELRO • *No Execute* (NX) aktif • PIE aktif • *Stack Canary* aktif	AppArmor (Profil Longgar)
5.	*BOF-3*	*Buffer Overflow*	• Full RELRO • *No Execute* (NX) tidak aktif • PIE aktif • *Stack Canary* aktif	AppArmor (Profil Ketat)
6.	*FMT-3*	*Format String*	• *Partial* RELRO • *No Execute* (NX) aktif • PIE aktif • *Stack Canary* aktif	AppArmor (Profil Ketat)
7.	*BOF-4*	*Buffer Overflow*	• *Full* RELRO • *No Execute* (NX) aktif • PIE aktif	AppArmor (Profil Longgar) + Fail2Ban

			• *Stack Canary* aktif	
8.	*FMT-4*	*Format String*	• *Full* RELRO • *No Execute* (NX) aktif • PIE aktif • *Stack Canary* aktif	AppArmor (Profil Longgar) + Fail2Ban
9.	*BOF-5*	*Buffer Overflow*	• *Full* RELRO • *No Execute* (NX) aktif • PIE aktif • *Stack Canary* aktif	AppArmor (Profil Ketat) + Fail2Ban
10.	*FMT-5*	*Format String*	• *Full* RELRO • *No Execute* (NX) tidak aktif • PIE aktif • *Stack Canary* aktif	AppArmor (Profil Ketat) + Fail2Ban

BAB IV
HASIL DAN PEMBAHASAN

4.1 Analisis Kebutuhan

Tahap analisis kebutuhan dilakukan untuk mendefinisikan spesifikasi perangkat virtual dan perangkat lunak yang diperlukan dalam perancangan sistem keamanan terhadap eksploitasi *buffer overflow* dan *format string*.

4.1.1 Analisis Perangkat Virtual

Lingkungan pengujian membutuhkan perangkat virtual atau *Virtual Private Server* yang disewa dari 2 penyedia layanan komputasi awan (*cloud provider*), yaitu Tencent Cloud dan Alibaba Cloud. Perangkat virtual yang dibutuhkan tertera pada Tabel 4.1.

Tabel 4. 1 Kebutuhan Perangkat Virtual

No	Nama Komponen	Spesifikasi	Keterangan
1.	Platform CTFd (server pusat)	• 8 GB RAM • 2 vCPU cores • 80 GB SSD	Mesin virtual ini menjalankan platform CTFd yang mengekspos 10 *file ELF binary* dan akses ke masing-masing VM kepada penyerang. Selain itu, terdapat aplikasi Grafana, Prometheus, dan Grafana Loki yang bertujuan untuk memonitoring dan menampilkan visualisasi grafik saat terjadi upaya eksploitasi pada setiap VM.
2.	VM-1 (tanpa proteksi)	• 2 GB RAM • 2 vCPU cores • 50 GB SSD	Mesin virtual ini akan menjalankan *file ELF binary* "BOF-1" di *port* 10001 dan "FMT-1" di *port* 10002.

3.	VM-2 (proteksi sebagian)	• 2 GB RAM • 2 vCPU cores • 50 GB SSD	Mesin virtual ini akan menjalankan *file ELF binary* "BOF-2" di *port* 10003, "FMT-2" di *port* 10004, "BOF-3" di *port* 10005, dan "FMT-3" di *port* 10006. Untuk *file binary* "BOF-2" dan "FMT-2" dikonfigurasi AppArmor dengan profil longgar, sedangkan *file binary* "BOF-3" dan "FMT-3" memiliki profil AppArmor yang ketat.
4.	VM-3 (proteksi penuh)	• 2 GB RAM • 2 vCPU cores • 40 GB SSD	Mesin virtual ini akan menjalankan *file ELF binary* "BOF-4" di *port* 10007, "FMT-4" di *port* 10008, "BOF-5" di *port* 10009, dan "FMT-5" di *port* 10010. Untuk *file binary* "BOF-4" dan "FMT-4" dikonfigurasi AppArmor dengan profil longgar, sedangkan *file binary* "BOF-5" dan "FMT-5" memiliki profil AppArmor yang ketat. Selain itu, terdapat proteksi tambahan Fail2Ban untuk melindungi eksploitasi dari sisi jaringan.

Adapun keempat kebutuhan perangkat virtual di atas memiliki alamat IP publik yang dapat diakses dari internet. Detail alamat IP dari keempat *Virtual Private Server* yang digunakan pada penelitian ini tertera pada Tabel 4.2.

Tabel 4. 2 Informasi Alamat IP pada Perangkat Virtual yang Digunakan

No	Nama VM	Alamat IP	Keterangan
1.	VM-CTF (server pusat)	43.129.38.76	VPS ini dijalankan di provider Tencent Cloud. VM-CTF menjalankan platform CTF yang mengekspos 10 *file ELF binary* yang dapat diakses oleh penyerang.
2.	VM-1 (tanpa proteksi)	43.133.128.127	VPS ini dijalankan di penyedia layanan Tencent Cloud. VM-1 menjalankan 2 *file ELF binary* tanpa proteksi apapun.
3.	VM-2 (proteksi sebagian)	43.157.243.92	VPS ini dijalankan di penyedia layanan Tencent Cloud. VM-2 menjalankan 4 *file ELF binary* di *port* yang berbeda-beda dan dengan proteksi tambahan, yaitu AppArmor.
4.	VM-3 (proteksi penuh)	149.129.193.12	VPS ini dijalankan di penyedia layanan Alibaba Cloud. VM-3 menjalankan 4 *file ELF binary* di *port* yang berbeda-beda dengan proteksi penuh, di mana terdapat AppArmor dan Fail2Ban. AppArmor untuk menolak pelanggaran yang dilakukan oleh penyerang, sedangkan Fail2Ban untuk memblokir insiden serangan melalui jaringan.

4.1.2 Analisis Perangkat Lunak

Daftar perangkat lunak yang dibutuhkan pada penelitian ini tertera pada Tabel 4.3.

Tabel 4. 3 Kebutuhan Perangkat Lunak

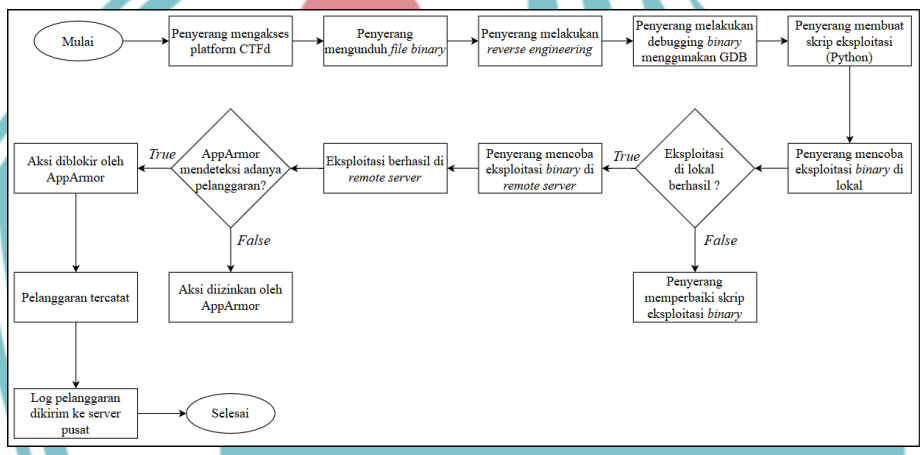
No	Nama Perangkat Lunak	Keterangan
1.	CTFd	Menjalankan kompetisi CTF selama durasi waktu tertentu.
2.	Sublime Text	Membuat kode program (berbasis Bahasa C) yang memiliki kerentanan *buffer overflow* dan *format string* sebagai objek pengujian.
3.	Fail2Ban	Alat untuk memblokir alamat IP penyerang dalam upaya eksploitasi *buffer overflow* dan *format string*.
4.	AppArmor	Modul keamanan untuk membatasi ruang lingkup *binary* yang berjalan di Ubuntu Server.
5.	Socat	Alat yang berfungsi untuk menjalankan 10 *file ELF binary* di *port* yang telah ditentukan.

4.2 Perancangan Sistem

Perancangan sistem pada penelitian pembuatan diagram alir (*flowchart*) dan diagram blok (*block diagram*). Diagram alir (*flowchart*) dibuat dalam bentuk grafik disertai dengan urutan terhadap prosedur suatu program (Malabay, 2016). Diagram blok lebih berfokus pada alur *input*, *process*, dan *output* hasil pengujian keamanan. Penelitian ini membagi diagram alir menjadi 2, yaitu diagram alir proses pembatasan dampak eksploitasi oleh AppArmor dan diagram alir proses pemblokiran alamat IP oleh Fail2Ban. *Flowchart* dipisah dikarenakan perbedaan peran dari kedua alat keamanan.

4.2.1 Diagram Alir Proses Pembatasan Dampak Eksploitasi

AppArmor bertugas untuk menolak setiap adanya pelanggaran yang dilakukan oleh penyerang selama proses eksploitasi *binary*. Dalam kasus ini, profil AppArmor dibuat menjadi 2 bagian, yaitu profil longgar dan profil ketat. Kedua profil tersebut sama-sama mengizinkan penyerang untuk mengakses shell dan membaca isi dari *file flag* untuk keperluan kompetisi CTF. Namun, profil ketat tidak mengizinkan penyerang untuk eksekusi perintah tertentu, seperti *"whoami"* dan membaca file-file sensitif, seperti file *"/etc/passwd"*. Diagram alir proses penolakan ditunjukkan pada Gambar 4.1.

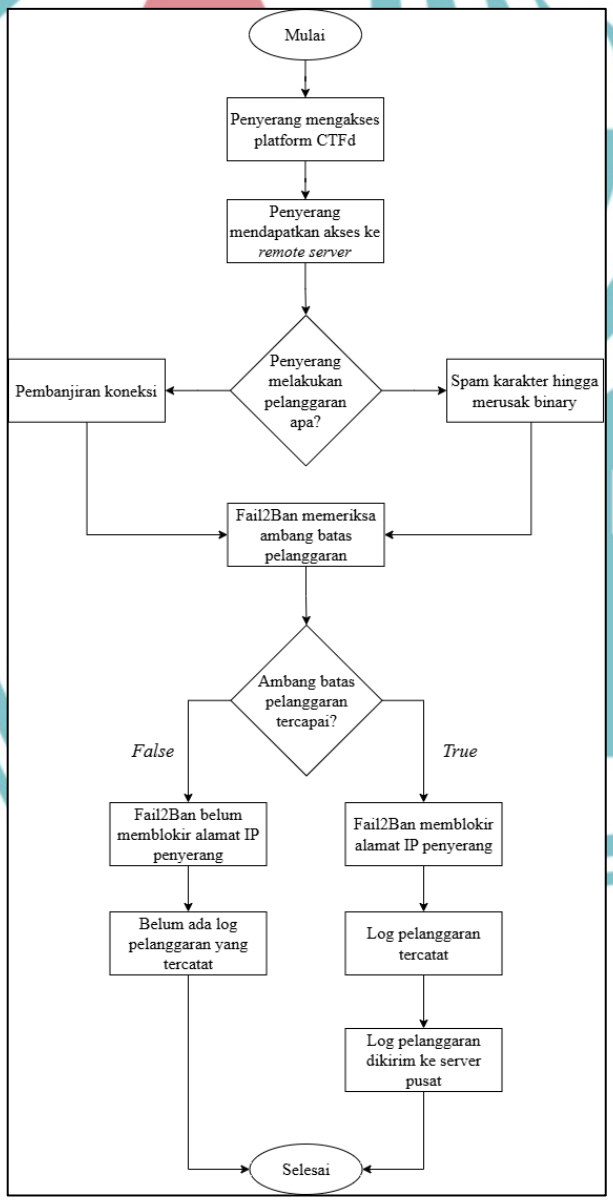


Gambar 4. 1 Diagram Alir Proses Pembatasan Dampak Eskploitasi oleh AppArmor

Proses diawali dengan penyerang yang mengakses platform CTFd untuk mendapatkan *file binary* yang ingin dieksploitasi. Sebelum eksploitasi di remote server menggunakan netcat, penyerang umumnya melakukan proses *reverse engineering* atau *static analysis* untuk memahami alur logika program. Kemudian, memahami *memory mapping* ketika program dijalankan di lokal melalui proses *debugging* menggunakan GDB. Ketika proses eksploitasi di lokal berhasil, penyerang langsung eksploitasi di *remote server*. Apabila proses eksploitasi di *remote server* berhasil, maka AppArmor mulai bekerja. Di mana, AppArmor akan mendeteksi adanya pelanggaran saat proses eksploitasi. Jika pelanggaran terdeteksi, maka upaya pelanggaran tersebut langsung ditolak (status *"denied"*), lalu *log* pelanggaran akan tercatat di *file* log sistem Linux, yaitu *"/var/log/syslog"*. Log yang tercatat akan dikirimkan ke server pusat melalui *Node Exporter* untuk divisualisasikan dalam bentuk grafik yang mudah dimengerti.

4.2.2 Diagram Alir Sistem Proses Pemblokiran Alamat IP

Pemblokiran alamat IP penyerang dilakukan oleh Fail2Ban. Di mana, Fail2Ban berperan sebagai penjaga di sisi jaringan, seperti pembanjiran koneksi (*TCP connection flooding*) dan spam karakter secara masif atau kesalahan skrip eksploitasi yang berpotensi merusak alur jalannya aplikasi atau *binary*. Dengan demikian, apabila terdapat upaya perusakan di VM-3 yang telah mencapai ambang batas tertentu, maka Fail2Ban akan langsung memblokir alamat IP penyerang. Detail dari diagram alir proses pemblokiran alamat IP oleh Fail2Ban dapat dilihat pada Gambar 4.2.



Gambar 4. 2 Diagram Alir Proses Pemblokiran Alamat IP oleh Fail2Ban

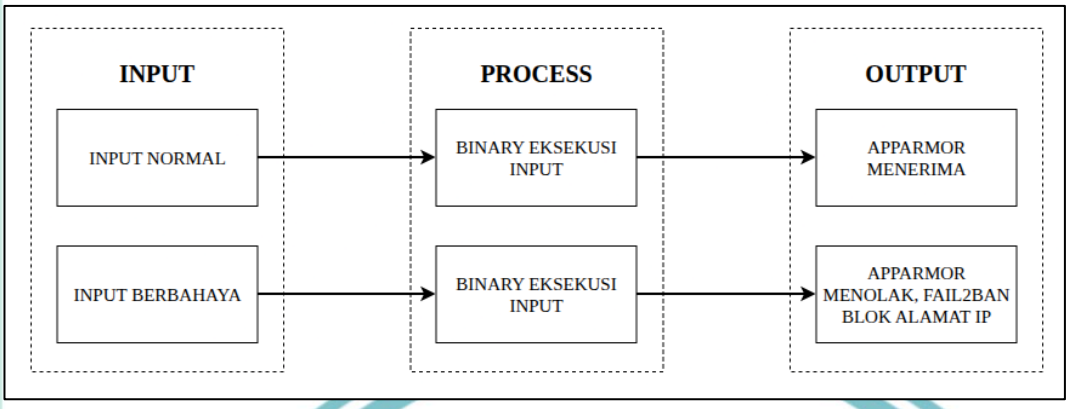
Alur awalnya mirip dengan *flowchart* proses pembatasan dampak eksploitasi oleh AppArmor, yaitu penyerang mengakses platform CTFd terlebih dahulu untuk mendapatkan akses ke *remote server*. Setelah akses diperoleh, penyerang melakukan 2 upaya pelanggaran pada VM-3, yaitu bisa dengan pembanjiran koneksi (*TCP connection flooding*) atau spam karakter atau kesalahan pada skrip eksploitasi hingga membuat aplikasi atau *binary* menjadi *crash* berulang kali. Berikut adalah konfigurasi ambang batas pada Fail2Ban terhadap 2 kondisi perusakan tersebut:

1) Ambang batas pelanggaran pembanjiran koneksi (*flood-detection*) adalah 50 koneksi pada 1 sumber alamat IP yang sama dalam kurun waktu 1 menit (60 detik). Konfigurasi tersebut menjadi indikasi kuat jika penyerang berupaya membuat *remote server* menjadi *down*, sehingga tidak dapat diakses.
2) Ambang batas peristiwa *crash* proses *binary* (*crash-detection*) adalah sebanyak 3 kali pada 1 sumber alamat IP yang sama dalam kurun waktu 10 menit (600 detik). Peristiwa *crash* dapat karena pembanjiran koneksi dan spam karakter secara masif atau upaya eksploitasi yang gagal. Konfigurasi tersebut dipilih karena tergolong ideal untuk memastikan jika 1 sumber alamat IP yang sama yang melakukan pelanggaran. Jika hanya 1 kali langsung diblokir (terlalu ketat), maka dapat terjadi *false-positive*, karena belum tentu itu merupakan alamat IP penyerang. Jika lebih dari 3 kali, maka ketika penyerang melakukan perusakan secara masif, dampaknya adalah aplikasi atau *binary* yang sering *crash*.

Jika ambang batas pelanggaran pada kedua kondisi di atas telah tercapai, maka alamat IP penyerang akan langsung diblokir oleh Fail2Ban. Kemudian, *log* pelanggaran akan dicatat oleh Fail2Ban dan *log* tersebut dikirim ke server pusat melalui Grafana Alloy untuk proses audit sistem keamanan lebih lanjut.

4.2.3 Diagram Blok

Diagram blok dibuat dengan 2 jenis, yaitu proteksi tambahan di lapisan sistem operasi oleh AppArmor dan proteksi tambahan di lapisan jaringan oleh Fail2Ban. Rancangan diagram blok dapat dilihat pada Gambar 4.3.



Gambar 4. 3 Diagram Blok

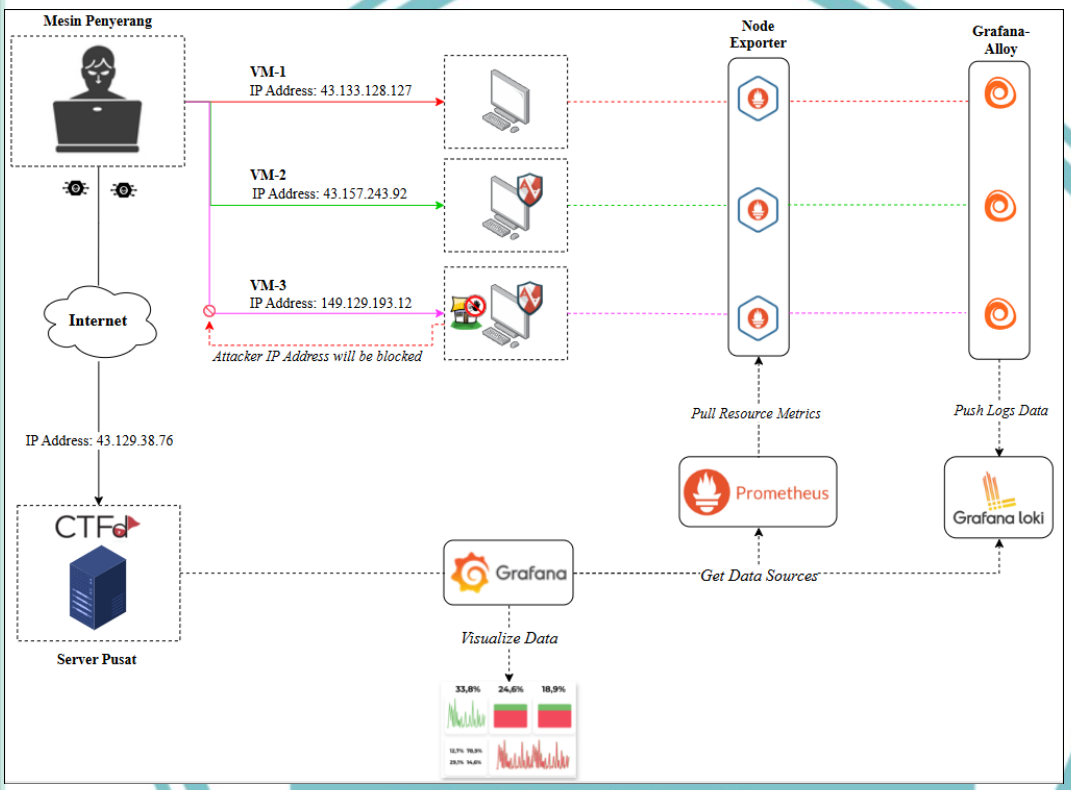
Perbedaan aksi terletak pada bagian *output*. Di mana, AppArmor akan mengizinkan input normal dan menolak input yang memiliki indikasi pelanggaran. Selain itu, input karakter yang dilakukan secara terus-menerus (dispam) hingga membuat *binary* menjadi *crash*, maka Fail2Ban akan mencatat pelanggaran sampai mencapai ambang batas tertentu. Ketika ambang batas pelanggaran sudah tercapai, maka Fail2Ban akan memblokir sumber asal alamat IP yang melakukan spam input secara masif.

4.2.4 Arsitektur Sistem

Arsitektur sistem keamanan dirancang untuk bisa memonitoring dan menahan, serta memblokir upaya eksploitasi yang dilakukan oleh penyerang. Untuk memonitoring sumber daya dan data log pada VM-1, VM-2, dan VM-3, diperlukan alat *Node Exporter* dan Grafana Alloy. *Node Exporter* dan Grafana Alloy bertindak sebagai agen yang dipasang langsung di setiap VM. *Node Exporter* menyediakan metrik sumber daya di setiap VM yang dapat ditarik datanya setiap 10 detik oleh Prometheus. Kemudian, Grafana Alloy akan mengirimkan data *log* (berupa teks) ke Grafana Loki untuk mengaudit adanya indikasi pelanggaran. Perbedaan antara *Node Exporter* dan Grafana Alloy terletak pada penerimaan data di server pusat. Di mana, Prometheus mengambil data metrik setiap VM menggunakan metode *pull* yang telah disediakan oleh agen *Node Exporter*, sedangkan Grafana Loki menggunakan metode *push* yang dikirimkan oleh agen Grafana Alloy. Data yang telah dikirimkan ke server pusat melalui Prometheus dan Grafana Loki selanjutnya akan divisualisasikan di dasbor Grafana.

Selain alat untuk memonitoring, terdapat AppArmor dan Fail2Ban yang berperan langsung dalam memproteksi VM-2 dan VM-3, baik pelanggaran di level jaringan dan di level sistem operasi. Dengan demikian, setiap pelanggaran yang dilakukan oleh penyerang dapat termonitoring secara akurat dan upaya tersebut akan ditolak oleh AppArmor, serta diblokir oleh Fail2Ban, sehingga penyerang sudah tidak bisa mengakses aplikasi atau *binary* melalui jaringan.

Detail arsitektur sistem pada penelitian ini tertera pada Gambar 4.4.



Gambar 4. 4 Arsitektur Sistem

4.3 Implementasi Sistem

Tahap implementasi adalah fase instalasi sistem secara nyata terhadap rancangan penelitian yang sudah dibuat. Implementasi mencakup instalasi platform CTFd, pemasangan dan konfigurasi sistem keamanan pada Ubuntu Server menggunakan Fail2Ban dan AppArmor. Setelah itu, dilakukan instalasi alat monitoring untuk observasi data metrik sumber daya dan *log* pada VM-1, VM-2, dan VM-3 secara *real-time*.

4.3.1 Instalasi Platform CTFd

Platform CTFd akan dipasang di server pusat. Langkah awal sebelum instalasi CTFd adalah dengan instalasi *web server* Nginx yang akan menjadi *reverse proxy*. Artinya, trafik pengguna dari internet akan diteruskan ke web CTFd yang berjalan di lokal pada *port* 8000. Selain itu, Nginx bisa digunakan untuk mengatur *rate limit* agar terhindar dari serangan DDoS. Dimulai dari instalasi *web server* Nginx dengan mengikuti perintah yang tertera pada Gambar 4.5.

```
ubuntu@VM-CTF:~$ sudo apt update && sudo apt install -y nginx
Hit:1 http://mirrors.tencentyun.com/ubuntu noble InRelease
Hit:2 http://mirrors.tencentyun.com/ubuntu noble-updates InRelease
Hit:3 http://mirrors.tencentyun.com/ubuntu noble-backports InRelease
Hit:4 http://mirrors.tencentyun.com/ubuntu noble-security InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
1 package can be upgraded. Run 'apt list --upgradable' to see it.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
nginx is already the newest version (1.24.0-2ubuntu7.13).
0 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
ubuntu@VM-CTF:~$
```

Gambar 4. 5 Instalasi Paket Nginx di Server Pusat

Setelah itu, hapus *file virtual host* bawaan milik Nginx, nama *file* tersebut adalah "*default*" yang berlokasi di direktori *"/etc/nginx/sites-enabled"*. Perintah untuk menonaktifkan *file virtual host* bawaan Nginx tertera pada Gambar 4.6.

```
(base) ubuntu@VM-CTF:/var/www/html/ctfd$ sudo rm -f /etc/nginx/sites-enabled/default
(base) ubuntu@VM-CTF:/var/www/html/ctfd$ ls -lah /etc/nginx/sites-enabled/
total 8.0K
drwxr-xr-x 2 root root 4.0K Jul 13 22:42 .
drwxr-xr-x 8 root root 4.0K Jul 13 20:16 ..
```

Gambar 4. 6 Hapus *File Virtual Host Default* Nginx

Selain itu, nonaktifkan *default web page* milik Nginx agar tidak tampil saat pengguna mengakses platform web CTFd. Gambar 4.7 menunjukkan *file default web page* Nginx yang telah dinonaktifkan.

```
ubuntu@VM-CTF:~$ sudo mv /var/www/html/index.nginx-debian.html /var/www/html/index.nginx-debian.html.bak
ubuntu@VM-CTF:~$ ls -lah /var/www/html/
total 12K
drwxr-xr-x 2 root root 4.0K Jul 13 20:13 .
drwxr-xr-x 3 root root 4.0K Jul 13 20:08 ..
-rw-r--r-- 1 root root  615 Jul 13 20:08 index.nginx-debian.html.bak
ubuntu@VM-CTF:~$
```

Gambar 4. 7 Nonaktifkan Nginx *Default Web Page*

Edit *file* konfigurasi Nginx (*"/etc/nginx/nginx.conf"*) untuk menyembunyikan informasi versi Nginx untuk mencegah enumerasi kerentanan dari versi Nginx. Selain itu, terapkan konfigurasi *rate limit* agar meminimalisir dari upaya DDoS. Konfigurasi untuk menyembunyikan versi Nginx dan *rate limit* tertera pada Gambar 4.8.

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
error_log /var/log/nginx/error.log;
include /etc/nginx/modules-enabled/*.conf;

events {
        worker_connections 768;
        # multi_accept on;
}

http {

        ##
        # Basic Settings
        ##

        sendfile on;
        tcp_nopush on;
        types_hash_max_size 2048;
        server_tokens off; # hilangkan versi Nginx dari header Server & halaman error

        # zona rate-limit: 100 request/menit per IP; memori 10MB simpan ~160k IP
        limit_req_zone $binary_remote_addr zone=ctfd_rl:10m rate=100r/m;

        # server_names_hash_bucket_size 64;
        # server_name_in_redirect off;

        include /etc/nginx/mime.types;
        default_type application/octet-stream;

        ##
        # SSL Settings
        ##

        ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3; # Dropping SSLv3, ref: POODLE
"/etc/nginx/nginx.conf" 86L, 1644B written
```

Gambar 4. 8 Edit File Konfigurasi Nginx

Buat file *virtual host* Nginx sebagai *reverse proxy*, yaitu untuk mengarahkan pengguna dari internet ke platform web CTFd yang berjalan di alamat IP lokal pada *port* 8000. Pada tahap ini, *file virtual host* tersebut dinamai dengan *"ctfd.conf"* dan simpan di dalam direktori *"/etc/nginx/sites-available/*". Gambar 4.9 menunjukkan *file virtual host* Nginx sebagai *reverse proxy* ke platform CTFd.

```
server {
        listen 80;
        server_name 43.129.38.76 pwngame.site;

        return 301 http://ctf.pwngame.site$request_uri;
}

server {
        listen 80;
        server_name ctf.pwngame.site;

        # Sembunyikan teknologi backend yang dibocorkan Gunicorn/Werkzeug/Flask
        proxy_hide_header Server;
        proxy_hide_header X-Powered-By;

        # Header keamanan dasar
        add_header X-Content-Type-Options "nosniff" always;
        add_header X-Frame-Options "SAMEORIGIN" always;

        client_max_body_size 10m; # CTFd upload file/attachment challenge

        location / {
                # Rate limit: 100r/m, burst 20 supaya lonjakan wajar peserta tidak kena;
                # nodelay agar burst dilayani segera, sisanya baru ditolak (503)
                limit_req zone=ctfd_rl burst=20 nodelay;

                proxy_pass http://127.0.0.1:8000;
                proxy_set_header Host              $host;
                proxy_set_header X-Real-IP         $remote_addr;
                proxy_set_header X-Forwarded-For   $proxy_add_x_forwarded_for;
                proxy_set_header X-Forwarded-Proto $scheme;
                proxy_read_timeout 60s;
        }
}
/etc/nginx/sites-available/ctfd.conf  NGINX
```

Gambar 4. 9 Konfigurasi Nginx Sebagai Reverse Proxy

Aktifkan konfigurasi *virtual host* Nginx yang sudah dibuat sebelumnya. Lalu, jalankan layanan *web server* Nginx secara permanen. Perintah untuk mengaktivasi layanan Nginx secara permanen tertera pada Gambar 4.10.

```
ubuntu@VM-CTF:~$ sudo ln -s /etc/nginx/sites-available/ctfd.conf /etc/nginx/sites-enabled/
ubuntu@VM-CTF:~$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
ubuntu@VM-CTF:~$ sudo systemctl reload nginx.service
ubuntu@VM-CTF:~$ sudo systemctl restart nginx.service
ubuntu@VM-CTF:~$ sudo systemctl enable --now nginx.service
Synchronizing state of nginx.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable nginx
ubuntu@VM-CTF:~$
```

Gambar 4. 10 Jalankan Layanan *Web Server* Nginx Secara Permanen

Tahap konfigurasi Nginx sudah selesai, tetapi perlu penyesuaian kembali jika ingin ditambahkan dengan SSL/TLS supaya trafik pengguna aman dan terhindar dari *Man-in-the-Middle* (MitM).

Lanjut ke tahap instalasi platform CTFd dengan cara mengunduh versi aplikasi CTFd terbaru, lalu mengekstraknya dengan alat *"tar"*. Perintah untuk mengunduh aplikasi CTFd versi terbaru dan ekstraksi *file* unduhannya tertera pada Gambar 4.11.

```
(base) ubuntu@VM-CTF:/var/www/html$ sudo curl https://codeload.github.com/CTFd/CTFd/tar.gz/refs/tags/3.8.6 -o /var/www/html/ctfd.tar.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 13.0M    0 13.0M    0     0  10.1M      0 --:--:--  0:00:01 --:--:-- 10.1M
(base) ubuntu@VM-CTF:/var/www/html$ sudo tar -xzf ctfd.tar.gz
(base) ubuntu@VM-CTF:/var/www/html$ sudo mv CTFd-3.8.6/ ctfd/
(base) ubuntu@VM-CTF:/var/www/html$ sudo rm -f ctfd.tar.gz
(base) ubuntu@VM-CTF:/var/www/html$ sudo chgrp -R $USER ctfd/
(base) ubuntu@VM-CTF:/var/www/html$ ls -lah
total 16K
drwxr-xr-x 3 root root   4.0K Jul 13 21:38 .
drwxr-xr-x 3 root root   4.0K Jul 13 20:08 ..
drwxrwxr-x 8 root ubuntu 4.0K Jun 16 23:22 ctfd
-rw-r--r-- 1 root root    615 Jul 13 20:08 index.nginx-debian.html.bak
(base) ubuntu@VM-CTF:/var/www/html$
```

Gambar 4. 11 Kloning Repositori GitHub CTFd

Buat *file service Systemd* dengan nama "*ctfd.service*" agar platform CTFd dapat berjalan di belakang layar dan secara permanen. Buat *file service Systemd* di dalam direktori *"/etc/systemd/system"*. Dikarenakan platform CTFd dibuat menggunakan bahasa Python, maka untuk menjalankannya bisa dengan bantuan "*gunicorn*". Isi *file service Systemd* untuk platform CTFd ditunjukkan pada Gambar 4.12.

```
0  [Unit]
1  Description=CTFd (Gunicorn)
2  After=network.target
3
4  [Service]
5  User=ubuntu
6  Group=ubuntu
7  WorkingDirectory=/var/www/html/ctfd
8  ExecStart=/home/ubuntu/base/bin/gunicorn \
9      --workers 5 --bind 127.0.0.1:8000 "CTFd:create_app()"
10 Restart=always
11 RestartSec=3
12
13 [Install]
14 WantedBy=multi-user.target
```

Gambar 4. 12 Buat *File Service* CTFd

Jalankan layanan CTFd secara permanen dan pastikan jika layanannya sudah berjalan di belakang layar. Perintah untuk menjalankan layanan CTFd secara permanen tertera pada Gambar 4.13.

```
(base) ubuntu@VM-CTF:/var/www/html/ctfd$ sudo vim /etc/systemd/system/ctfd.service
(base) ubuntu@VM-CTF:/var/www/html/ctfd$ sudo systemctl daemon-reload
(base) ubuntu@VM-CTF:/var/www/html/ctfd$ sudo systemctl enable --now ctfd.service
Created symlink /etc/systemd/system/multi-user.target.wants/ctfd.service → /etc/systemd/system/ctfd.service.
(base) ubuntu@VM-CTF:/var/www/html/ctfd$ sudo systemctl status ctfd.service
● ctfd.service - CTFd (Gunicorn)
     Loaded: loaded (/etc/systemd/system/ctfd.service; enabled; preset: enabled)
     Active: active (running) since Tue 2026-07-14 04:33:39 WIB; 11s ago
   Main PID: 185724 (gunicorn)
      Tasks: 6 (limit: 9050)
     Memory: 408.1M (peak: 408.5M)
        CPU: 7.295s
     CGroup: /system.slice/ctfd.service
             ├─185724 /home/ubuntu/base/bin/python3 /home/ubuntu/base/bin/gunicorn --workers 5 --bind 127.0.0.1:8000 "CTFd:create_app()"
             ├─185730 /home/ubuntu/base/bin/python3 /home/ubuntu/base/bin/gunicorn --workers 5 --bind 127.0.0.1:8000 "CTFd:create_app()"
             ├─185731 /home/ubuntu/base/bin/python3 /home/ubuntu/base/bin/gunicorn --workers 5 --bind 127.0.0.1:8000 "CTFd:create_app()"
             ├─185732 /home/ubuntu/base/bin/python3 /home/ubuntu/base/bin/gunicorn --workers 5 --bind 127.0.0.1:8000 "CTFd:create_app()"
             ├─185733 /home/ubuntu/base/bin/python3 /home/ubuntu/base/bin/gunicorn --workers 5 --bind 127.0.0.1:8000 "CTFd:create_app()"
             └─185734 /home/ubuntu/base/bin/python3 /home/ubuntu/base/bin/gunicorn --workers 5 --bind 127.0.0.1:8000 "CTFd:create_app()"

Jul 14 04:33:41 VM-CTF gunicorn[185730]: INFO  [alembic.runtime.migration] Context impl SQLiteImpl.
Jul 14 04:33:41 VM-CTF gunicorn[185730]: INFO  [alembic.runtime.migration] Will assume non-transactional DDL.
Jul 14 04:33:41 VM-CTF gunicorn[185731]: INFO  [alembic.runtime.migration] Context impl SQLiteImpl.
Jul 14 04:33:41 VM-CTF gunicorn[185731]: INFO  [alembic.runtime.migration] Will assume non-transactional DDL.
Jul 14 04:33:41 VM-CTF gunicorn[185732]: INFO  [alembic.runtime.migration] Context impl SQLiteImpl.
Jul 14 04:33:41 VM-CTF gunicorn[185732]: INFO  [alembic.runtime.migration] Will assume non-transactional DDL.
Jul 14 04:33:41 VM-CTF gunicorn[185734]: INFO  [alembic.runtime.migration] Context impl SQLiteImpl.
Jul 14 04:33:41 VM-CTF gunicorn[185734]: INFO  [alembic.runtime.migration] Will assume non-transactional DDL.
Jul 14 04:33:41 VM-CTF gunicorn[185733]: INFO  [alembic.runtime.migration] Context impl SQLiteImpl.
Jul 14 04:33:41 VM-CTF gunicorn[185733]: INFO  [alembic.runtime.migration] Will assume non-transactional DDL.
(base) ubuntu@VM-CTF:/var/www/html/ctfd$
```

Gambar 4. 13 Jalankan Layanan CTFd Secara Permanen

Pemasangan platform CTFd pada server pusat sudah selesai. Selanjutnya, diperlukan instalasi sertifikat SSL/TLS pada *web server* Nginx agar trafik pengiriman data aman dari upaya intersepsi pihak ke-3 (MitM). Diawali dengan instalasi paket "*certbot*" dan *"python3-certbot-nginx"*. Perintah instalasinya tertera pada Gambar 4.14.

```
(base) ubuntu@VM-CTF:~$ sudo apt install -y certbot python3-certbot-nginx
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
certbot is already the newest version (2.9.0-1).
python3-certbot-nginx is already the newest version (2.9.0-1).
0 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
(base) ubuntu@VM-CTF:~$
```

Gambar 4. 14 Instalasi Paket Certbot untuk *Generate* Sertifikat SSL

Kemudian, *generate* sertifikat SSL yang akan digunakan oleh *web server* Nginx. Tahap ini perlu disesuaikan dengan nama domain atau alamat IP pada server pusat. Perintah *"certbot"* untuk *generate* sertifikat SSL pada nama domain tertentu ditunjukkan pada Gambar 4.15.

```
(base) ubuntu@VM-CTF:~$ sudo certbot --nginx -d ctf.pwngame.site -d pwngame.site
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Enter email address (used for urgent renewal and security notices)
 (Enter 'c' to cancel): wahyupriambodo.cloud@gmail.com

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Please read the Terms of Service at
https://letsencrypt.org/documents/LE-SA-v1.8-July-06-2026.pdf. You must agree in
order to register with the ACME server. Do you agree?
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(Y)es/(N)o: Y

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Would you be willing, once your first certificate is successfully issued, to
share your email address with the Electronic Frontier Foundation, a founding
partner of the Let's Encrypt project and the non-profit organization that
develops Certbot? We'd like to send you email about our work encrypting the web,
EFF news, campaigns, and ways to support digital freedom.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(Y)es/(N)o: Y
Account registered.
Requesting a certificate for ctf.pwngame.site and pwngame.site

Successfully received certificate.
Certificate is saved at: /etc/letsencrypt/live/ctf.pwngame.site/fullchain.pem
Key is saved at:         /etc/letsencrypt/live/ctf.pwngame.site/privkey.pem
This certificate expires on 2026-10-11.
These files will be updated when the certificate renews.
Certbot has set up a scheduled task to automatically renew this certificate in the background.

Deploying certificate
Successfully deployed certificate for ctf.pwngame.site to /etc/nginx/sites-enabled/ctfd.conf
Successfully deployed certificate for pwngame.site to /etc/nginx/sites-enabled/ctfd.conf
Congratulations! You have successfully enabled HTTPS on https://ctf.pwngame.site and https://pwngame.site

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
If you like Certbot, please consider supporting our work by:
 * Donating to ISRG / Let's Encrypt:   https://letsencrypt.org/donate
 * Donating to EFF:                    https://eff.org/donate-le
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Gambar 4. 15 Generate Sertifikat SSL untuk *Web Server* Nginx

Pastikan sertifikat SSL yang berhasil di-*generate* dapat diperbarui kembali (*renew*) secara otomatis. Perintah untuk memeriksa status renew sertifikat SSL Nginx tertera pada Gambar 4.16.

```
(base) ubuntu@VM-CTF:~$ sudo systemctl status certbot.timer
● certbot.timer - Run certbot twice daily
     Loaded: loaded (/usr/lib/systemd/system/certbot.timer; enabled; preset: enabled)
     Active: active (waiting) since Tue 2026-07-14 04:36:21 CST; 11min ago
    Trigger: Tue 2026-07-14 18:13:08 CST; 13h left
   Triggers: ● certbot.service

Jul 14 04:36:21 VM-CTF systemd[1]: Started certbot.timer - Run certbot twice daily.
(base) ubuntu@VM-CTF:~$ sudo certbot renew --dry-run
Saving debug log to /var/log/letsencrypt/letsencrypt.log

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Processing /etc/letsencrypt/renewal/ctf.pwngame.site.conf
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Account registered.
Simulating renewal of an existing certificate for ctf.pwngame.site and pwngame.site

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Congratulations, all simulated renewals succeeded:
  /etc/letsencrypt/live/ctf.pwngame.site/fullchain.pem (success)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Gambar 4. 16 Testing *Renew* Sertifikat SSL Menggunakan Certbot

Edit *file virtual host* Nginx awal yang tadinya hanya menerima koneksi HTTP menjadi HTTPS dengan menggunakan sertifikat SSL yang berhasil di-*generate*. Gambar 4.17 menunjukkan *file virtual host* Nginx yang sudah diedit dengan menambahkan konfigurasi HTTPS.

```
# ── Block 1: semua HTTP (port 80) naik ke HTTPS ──
server {
    listen 80;
    server_name ctf.pwngame.site pwngame.site 43.129.38.76;
    return 301 https://ctf.pwngame.site$request_uri;
}

# ── Block 2: HTTPS apex/IP → redirect ke host kanonik ──
server {
    listen 443 ssl;
    server_name pwngame.site 43.129.38.76;

    ssl_certificate     /etc/letsencrypt/live/ctf.pwngame.site/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/ctf.pwngame.site/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    return 301 https://ctf.pwngame.site$request_uri;
}

# ── Block 3: HTTPS host kanonik → proxy ke CTFd ──
server {
    listen 443 ssl;
    server_name ctf.pwngame.site;

    ssl_certificate     /etc/letsencrypt/live/ctf.pwngame.site/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/ctf.pwngame.site/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    proxy_hide_header Server;
    proxy_hide_header X-Powered-By;

    add_header X-Content-Type-Options "nosniff" always;
    add_header X-Frame-Options "SAMEORIGIN" always;

    client_max_body_size 10m;

    location / {
        limit_req zone=ctfd_rl burst=20 nodelay;

        proxy_pass http://127.0.0.1:8000;
        proxy_set_header Host              $host;
        proxy_set_header X-Real-IP         $remote_addr;
        proxy_set_header X-Forwarded-For   $remote_addr;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_read_timeout 60s;
    }
}
```

Gambar 4. 17 Edit *File Virtual Host* Nginx

Pastikan *file virtual host* Nginx sudah tidak terdapat kesalahan sintaks. Jika sudah tidak ada sintaks yang bermasalah, maka bisa *restart* layanan Nginx menggunakan perintah yang tertera pada Gambar 4.18.

```
(base) ubuntu@VM-CTF: $ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
(base) ubuntu@VM-CTF: $ sudo ln -s /etc/nginx/sites-available/ctfd.conf /etc/nginx/sites-enabled/
(base) ubuntu@VM-CTF: $ sudo systemctl reload nginx.service
(base) ubuntu@VM-CTF: $
```

Gambar 4. 18 *Reload* Layanan Nginx

4.3.2 Kompilasi Kode Program C Menjadi *File ELF Binary*

Jumlah *file ELF binary* yang digunakan sebagai objek penelitian adalah 10 dengan variasi logika program dan proteksi bawaan. Hal tersebut bertujuan untuk membuat penyerang mendapatkan tantangan di setiap objek pengujian. Perancangan *file binary* tersebut diskenariokan memiliki kerentanan *buffer overflow* dan *format string*. Kesepuluh *file binary* tersebut dikompilasi dengan berbasis bahasa C yang memiliki banyak fungsi-fungsi berbahaya. Salah satunya adalah fungsi *"gets()"* yang tidak menerapkan *boundary check*, sehingga penyerang dapat menginputkan karakter lebih dari batas maksimal yang telah ditentukan (*buffer overflow*). Selain itu, penggunaan fungsi *"printf()"* yang tidak diimplementasikan dengan benar juga dapat berdampak terhadap kerentanan *format string*.

Potongan fungsi pada "BOF-1" yang memiliki celah *buffer overflow* terletak di baris ke-40 dan ke-46. Potongan fungsi tersebut tertera pada Gambar 4.19.

```
33  // ---- Vulnerable Function ----
34  static void vuln() {
35      char buf[MAX_SIZE];
36
37      // ---- Leak Canary ----
38      puts("Leak something first!");
39      printf(">> ");
40      read(0, buf, MAX_SIZE+1); // <---- Off-by-one overwrite
41      printf("Hmmm... %s\n", buf);
42
43      // ---- Classic Buffer Overflow ----
44      puts("Then?");
45      printf(">> ");
46      gets(&buf); // <---- Buffer overflow
47      printf("\n"); // only for newline
48  }
```

Gambar 4. 19 *Vulnerable Function* pada Kode Program BOF-1

Potongan fungsi pada "FMT-1" yang memiliki kerentanan *format string* terletak di baris ke-36. Potongan fungsi tersebut tertera pada Gambar 4.20.

```
24  // ---- Vulnerable Function ----
25  static void vuln() {
26      char buf[MAX_SIZE];
27      memset(buf, 0, MAX_SIZE);
28
29      puts("What do you want to do?");
30      printf(">> ");
31
32      read(0, buf, MAX_SIZE-1);
33      buf[strcspn(buf, "\n")] = '\0';
34
35      puts("You said : ");
36      printf(buf); // <---- format string
37
38      puts("\nBye");
39      exit(EXIT_FAILURE);
40  }
```

Gambar 4. 20 *Vulnerable Function* pada Kode Program FMT-1

Potongan fungsi pada "BOF-2" yang memiliki celah *buffer overflow* terletak di baris ke-103 dan ke-114. Potongan fungsi tersebut tertera pada Gambar 4.21.

```
94  // ---- Vulnerable Function ----
95  static void vuln() {
96      char buf[BUF_MAX_SIZE];
97
98      puts("Note as a Service (NaaS)");
99      puts("Input your name first!");
100     printf(">> ");
101
102     // ---- STAGE 1 : LEAK CANARY ----
103     read(0, buf, 0x20); // <---- Buffer overflow
104     printf("Welcome, %s\n", buf);
105
106     // ---- STAGE 2 : ORW ----
107     puts("Input your note");
108     printf(">> ");
109     read(0, note, NOTE_MAX_SIZE);
110
111     // ---- STAGE 3 : STACK PIVOTING TO BSS ----
112     puts("Input note again?");
113     printf(">> ");
114     read(0, buf, 0x100); // <---- Buffer overflow
115
116     puts("Noted.");
117 }
```

Gambar 4. 21 *Vulnerable Function* pada Kode Program BOF-2

Potongan fungsi pada "FMT-2" yang memiliki kerentanan *format string* terletak di baris ke-33 dan ke-39. Potongan fungsi tersebut tertera pada Gambar 4.22.

```
24  // ---- Vulnerable Function ----
25  static void vuln() {
26      char buf[MAX_SIZE];
27      memset(buf, 0, MAX_SIZE);
28
29      puts("I'm just out of idea, this actually same as before lolz");
30      printf(">> ");
31      read(0, buf, MAX_SIZE-1);
32      buf[strcspn(buf, "\n")] = '\0';
33      printf(buf); // <---- leak pie
34
35      puts("\nContinuing...");
36      printf(">> ");
37      read(0, buf, MAX_SIZE-1);
38      buf[strcspn(buf, "\n")] = '\0';
39      printf(buf); // <---- GOT overwrite
40      puts("\nOkay, thank's for playing!");
41  }
```

Gambar 4. 22 *Vulnerable Function* pada Kode Program FMT-2

Potongan fungsi pada "BOF-3" yang memiliki celah *buffer overflow* terletak di baris ke-84. Potongan fungsi tersebut tertera pada Gambar 4.23.

```
57  // ---- Vulnerable Function ----
58  static void vuln() {
59      struct shop s;
60      s.cart = s.buy_game;
61
62      unsigned int option = 0;
63
64      while(option != 3) {
65          puts("Kamu hanya punya 3 kesempatan untuk bertransaksi!");
66          puts("Selamat datang di Rusdi Gameshop");
67          puts("1. Request Game");
68          puts("2. Beli Game");
69          puts("3. Exit");
70
71          printf("Opsi >> ");
72          scanf("%d", &option);
73
74          switch(option) {
75              case 1:
76                  puts("Game apa yang kamu minta adakan?");
77                  printf(">> ");
78                  read(0, s.request_game, sizeof(s.request_game));
79                  printf("Game %s akan ditambahkan ke dalam list oleh admin Rusdi\n", s.request_game);
80                  break;
81              case 2:
82                  puts("Game apa yang kamu beli?");
83                  printf(">> ");
84                  read(0, s.buy_game, sizeof(s.buy_game)*2); // <---- Buffer overflow
85                  printf("Game %s tidak ada di dalam list\n", s.buy_game);
86                  break;
87              case 3:
88                  break;
89              default:
90                  printf("Opsi yang dipilih tidak valid!\n");
91                  exit(EXIT_FAILURE);
92          }
93      }
94
95      return;
96  }
```

Gambar 4. 23 *Vulnerable Function* pada Kode Program BOF-3

Potongan fungsi pada "FMT-3" yang memiliki kerentanan *format string* terletak di baris ke-115 dan ke-121. Potongan fungsi tersebut tertera pada Gambar 4.24.

```
104  // ---- Vulnerable Function ----
105  static void vuln() {
106      char buf[BUF_MAX_SIZE];
107      memset(buf, 0, BUF_MAX_SIZE);
108
109      puts("Welcome, king!");
110      puts("You made it this far");
111      puts("Before that, you've to prove that you deserve the crown");
112      printf(">> ");
113      read(0, buf, BUF_MAX_SIZE-1);
114      buf[strcspn(buf, "\n")] = '\0';
115      printf(buf); // <---- Format string
116
117      puts("\nThis is your 2nd chance king!");
118      printf(">> ");
119      read(0, buf, BUF_MAX_SIZE-1);
120      buf[strcspn(buf, "\n")] = '\0';
121      printf(buf); // <---- Format string
122      puts("\nNope.");
123  }
```

Gambar 4. 24 *Vulnerable Function* pada Kode Program FMT-3

Potongan fungsi pada "BOF-4" yang memiliki celah *buffer overflow* terletak di baris ke-32 dan ke-37. Potongan fungsi tersebut tertera pada Gambar 4.25.

```
26  // ---- Vulnerable Function ----
27  static void vuln() {
28      char buf[0x100];
29
30      puts("Leak something first!");
31      printf(">> ");
32      read(0, buf, sizeof(buf)+25); // <---- Buffer overflow
33      printf("Okay, here's for you: %s\n", buf);
34
35      puts("Are you sure?");
36      printf(">> ");
37      read(0, buf, sizeof(buf)*2); // <---- Buffer overflow
38      puts("Done, thank's!");
39  }
```

Gambar 4. 25 *Vulnerable Function* pada Kode Program BOF-4

Potongan fungsi pada "FMT-4" yang memiliki kerentanan *format string* terletak di baris ke-47 dan ke-53. Potongan fungsi tersebut tertera pada Gambar 4.26.

```
38  // ---- Vulnerable Function ----
39  static void vuln() {
40      char buf[MAX_SIZE];
41      memset(buf, 0, MAX_SIZE);
42
43      puts("Ez kok banh :v");
44      printf(">> ");
45      read(0, buf, MAX_SIZE-1);
46      buf[strcspn(buf, "\n")] = '\0';
47      printf(buf); // <---- Format string
48
49      puts("\nCoba lagi banh");
50      printf(">> ");
51      read(0, buf, MAX_SIZE-1);
52      buf[strcspn(buf, "\n")] = '\0';
53      printf(buf); // <---- Format string
54
55      // calling callback function
56      callback_t fn;
57      fn = (callback_t)(callback ^ guard);
58      fn();
59  }
```

Gambar 4. 26 *Vulnerable Function* pada Kode Program FMT-4

Potongan fungsi pada "BOF-5" yang memiliki celah *buffer overflow* terletak di baris ke-54 dan ke-59. Potongan fungsi tersebut tertera pada Gambar 4.27.

```
48  // ---- Vulnerable Function ----
49  static void vuln() {
50      char buffer[0x200];
51
52      puts("I give you 2 chances!");
53      printf(">> ");
54      read(0, buffer, sizeof(buffer)*2); // <---- Buffer overflow
55      printf("Okay, this is the first one: %s\n", buffer);
56
57      puts("This is your last chance, but not much!");
58      printf(">> ");
59      read(0, buffer, sizeof(buffer)*2); // <---- Buffer overflow
60      puts("Thank you for participating!");
61  }
```

Gambar 4. 27 *Vulnerable Function* pada Kode Program BOF-5

Potongan fungsi pada "FMT-5" yang memiliki kerentanan *format string* terletak di baris ke-32 dan ke-43. Potongan fungsi tersebut tertera pada Gambar 4.28.

```
20  // ---- Vulnerable Function ----
21  static void vuln() {
22      char sc[SC_MAX_SIZE];
23      char buf[BUF_MAX_SIZE];
24
25      memset(buf, 0, BUF_MAX_SIZE);
26      memset(sc, 0, SC_MAX_SIZE);
27
28      puts("Kalo ada yg gabisa solve parah sih :v");
29      printf(">> ");
30      read(0, buf, BUF_MAX_SIZE-1);
31      buf[strcspn(buf, "\n")] = '\0';
32      printf(buf); // <---- Format string
33
34      puts("\nGampang bat ini aseli, coba masukin dulu anunya wkwk");
35      printf(">> ");
36      read(0, sc, SC_MAX_SIZE-1);
37      sc[strcspn(sc, "\n")] = '\0';
38
39      puts("\nSaved. Sekarang waktunya dianuin :v");
40      printf(">> ");
41      read(0, buf, BUF_MAX_SIZE-1);
42      buf[strcspn(buf, "\n")] = '\0';
43      printf(buf); // <---- Format string
44  }
```

Gambar 4. 28 *Vulnerable Function* pada Kode Program FMT-5

Setelah kesepuluh *file ELF binary* telah dirancang memiliki kerentanan *buffer overflow* dan *format string*, maka tahap selanjutnya adalah kompilasi kode program menggunakan alat *"gcc"*. Namun, kompilasi kode program yang dilakukan satu per satu sangat tidak efisien, apalagi dengan proteksi bawaan yang berbeda-beda. Oleh karena itu, penggunaan *"Makefile"* akan mengotomatisasikan proses kompilasi hingga menghasilkan kesepuluh *file ELF binary*. *Makefile* untuk mengkompilasi 10 kode program C menjadi *file ELF binary* ditunjukkan pada Gambar 4.29.

```
79          -O0
80
81  fmt3:
82      $(CC) -o ../dist/fmt3 fmt3.c \
83          -Wno-format-security \
84          -Wno-implicit-function-declaration \
85          -fstack-protector-all \
86          -pie \
87          -Wl,-z,relro,-z,lazy \
88          -U_FORTIFY_SOURCE \
89          -O0 \
90          -lcrypto
91
92  fmt4:
93      $(CC) -o ../dist/fmt4 fmt4.c \
94          -Wno-format-security \
95          -Wno-implicit-function-declaration \
96          -fstack-protector-all \
97          -pie \
98          -Wl,-z,relro,-z,now \
99          -U_FORTIFY_SOURCE \
100         -O0
101
102 fmt5:
103     $(CC) -o ../dist/fmt5 fmt5.c \
104         -Wno-format-security \
105         -Wno-implicit-function-declaration \
106         -fstack-protector-all \
107         -pie \
108         -z execstack \
109         -Wl,-z,relro,-z,now \
110         -U_FORTIFY_SOURCE \
111         -O0
112
113 all: bof1 bof2 bof3 bof4 bof5 fmt1 fmt2 fmt3 fmt4 fmt5
114
115 clean:
116     rm -f ../dist/bof* ../dist/fmt*
```

Gambar 4. 29 *Makefile* untuk Otomatisasi Proses Kompilasi 10 *File ELF Binary*

Setelah itu, cukup ketikan perintah *"make all"*, maka semua kode program akan dikompilasi menjadi *file ELF binary*. Berikan izin eksekusi terlebih dahulu jika ingin menjalankan *file binary* tersebut. Dengan demikian, kesepuluh *file ELF binary* dapat didistribusikan ke penyerang melalui platform CTFd untuk percobaan eksploitasi selama durasi yang telah ditentukan. Proses kompilasi menggunakan *Makefile* tertera pada Gambar 4.30.

```
(labs) kuchiy0se@w1nd0wz:/mnt/d/Kuliah/Semester8/Skripsi/pwn/attachments/src$ make all
gcc -o ../dist/bof1 bof1.c \
        -Wno-stringop-overflow \
        -Wno-implicit-function-declaration \
        -Wformat-security \
        -fstack-protector \
        -no-pie \
        -Wl,-z,relro,-z,now \
        -U_FORTIFY_SOURCE \
        -O0
/usr/bin/ld: /tmp/ccRHOjdV.o: in function `vuln':
bof1.c:(.text+0x131): warning: the `gets' function is dangerous and should not be used.
gcc -o ../dist/bof2 bof2.c \
        -Wno-stringop-overflow \
        -Wno-implicit-function-declaration \
        -Wformat-security \
        -fstack-protector \
        -no-pie \
        -Wl,-z,relro,-z,now \
        -U_FORTIFY_SOURCE \
        -O0 \
        -lseccomp
```

Gambar 4. 30 Proses Kompilasi 10 Kode Program Menggunakan *Makefile*

4.3.3 Menjalankan *File ELF Binary* Menjadi Layanan Berbasis Jaringan

Setelah kesepuluh kode program berbasis C dikompilasi menjadi *file ELF binary*, maka langkah selanjutnya adalah menjalankan kesepuluh *file ELF binary* tersebut di masing-masing VM. Berikut adalah skenario *file ELF binary* yang akan dijalankan di masing-masing VM.

Tabel 4. 4 Detail Skenario *File ELF Binary* yang akan Dijalankan

No	Nama VM	Alamat IP Publik	Nama *Binary*	*Port* yang Digunakan
1.	VM-1	43.133.128.127	• BOF-1 • FMT-1	• 10001 • 10002
2.	VM-2	43.157.243.92	• BOF-2 • FMT-2 • BOF-3 • FMT-3	• 10003 • 10004 • 10005 • 10006

3.	VM-3	149.129.193.12	• BOF-4 • FMT-4 • BOF-5 • FMT-5	• 10007 • 10008 • 10009 • 10010

Untuk menjalankan *file ELF binary* di Linux agar bisa diakses melalui jaringan, diperlukan alat bernama *"socat"*. Alat tersebut perlu dipasang di ketiga VM. Tahap instalasi alat *"socat"* di Ubuntu Server dapat menggunakan perintah yang tertera pada Gambar 4.31.

```
ubuntu@VM-1:~$ sudo apt install -y socat
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
socat is already the newest version (1.8.0.0-4ubuntu0.1).
0 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
ubuntu@VM-1:~$
```

Gambar 4. 31 Instalasi *Socat* di Ubuntu Server

4.3.3.1 Menjalankan *File ELF Binary* di VM-1

Alat *"socat"* yang telah terpasang di VM-1, selanjutnya bisa langsung menjalankan *file binary* "BOF-1" dan "FMT-1" sesuai dengan *port* yang tertera pada Tabel 4.4. Sebelum menjalankan kedua *binary* tersebut, ada tahap tambahan yang diperlukan untuk menunjang ajang CTF ini, yaitu membuat *file flag*. Pembuatan file flag tertera pada Gambar 4.32.

```
ctf@VM-1:~/research-lab$ cd /opt/labs/
ctf@VM-1:/opt/labs$ cd bof/bof-1/
ctf@VM-1:/opt/labs/bof/bof-1$ touch flag-$(cat /proc/sys/kernel/random/uuid).txt
ctf@VM-1:/opt/labs/bof/bof-1$ vim flag-2976189a-a716-42f6-89e1-179517f33229.txt
ctf@VM-1:/opt/labs/bof/bof-1$ cd ../../fmt/fmt-1/
ctf@VM-1:/opt/labs/fmt/fmt-1$ touch flag-$(cat /proc/sys/kernel/random/uuid).txt
ctf@VM-1:/opt/labs/fmt/fmt-1$ vim flag-0a7ecab2-1616-48fa-a9e7-6207cac094e4.txt
ctf@VM-1:/opt/labs/fmt/fmt-1$ cd ../../
ctf@VM-1:/opt/labs$
```

Gambar 4. 32 Buat *File Flag* di VM-1

Jika penyerang berhasil eksploitasi hingga mendapatkan akses *shell* ke dalam sistem, maka penyerang sudah pasti bisa membaca isi *file flag*. Di tahap ini, kedua *file binary* sudah bisa dijalankan menggunakan *"socat"*, tetapi perlu penghubung antara *log socat* yang mengandung alamat IP penyerang dengan log sistem (*"/var/log/syslog"*) yang menyimpan riwayat pelanggaran selama eksploitasi berlangsung. Dengan begitu, proses audit *log* pelanggaran dapat mudah dipahami karena sudah disatukan.

Buat *file* baru bernama *"wrapper.sh"* (*custom Bash script*) untuk membantu menyatukan kedua file log tersebut. Isi dari *file "wrapper.sh"* tertera pada Gambar 4.33.

```
#!/bin/bash
# Usage: /opt/labs/wrapper.sh <path-binary>
# Run: EXEC:/opt/labs/wrapper.sh /opt/labs/bof/bof-1/bof1
# Used for: menjembatani SOCAT_PEERADDR (IP penyerang) dengan event koneksi & crash,
#         lalu menuliskannya ke /var/log/labs/ agar dapat dibaca Promtail → Loki.

BIN="$1"
CONN_LOG=/var/log/labs/conn.log
CRASH_LOG=/var/log/labs/crash.log
IP="${SOCAT_PEERADDR:-unknown}"
PORT="${SOCAT_SOCKPORT:-unknown}"

# Catat setiap koneksi masuk (baseline VM-1 & analisis flooding)
printf '%s CONNECT dari %s ke port %s (%s)\n' \
    "$(date '+%Y-%m-%d %H:%M:%S')" "$IP" "$PORT" "$BIN" >> "$CONN_LOG"

# Jalankan binary; stdin/stdout sudah terhubung ke socket oleh socat
stdbuf -o0 "$BIN"
status=$?

# 128+N = proses diterminasi oleh sinyal N. 139=SIGSEGV, 134=SIGABRT
if [ "$status" -ge 128 ]; then
    sig=$((status - 128))
    printf '%s CRASH terdeteksi dari %s port %s (%s exit=%d signal=%d)\n' \
        "$(date '+%Y-%m-%d %H:%M:%S')" "$IP" "$PORT" "$BIN" "$status" "$sig" >> "$CRASH_LOG"
fi

exit "$status"
```

Gambar 4. 33 Buat *File "wrapper.sh"* di VM-1 Untuk Menghubungkan *Log Socat* Dengan *Log* Sistem

Berikan hak eksekusi pada *file "wrapper.sh"* tersebut dengan perintah yang ditunjukkan pada Gambar 4.34.

```
ctf@VM-1:/opt/labs$ vim wrapper.sh
ctf@VM-1:/opt/labs$ chmod +x wrapper.sh
ctf@VM-1:/opt/labs$ ls -lah wrapper.sh
-rwxrwxr-x 1 ctf ctf 1.1K Jul 14 09:18 wrapper.sh
ctf@VM-1:/opt/labs$
```

Gambar 4. 34 Memberikan Hak Izin Eksekusi pada File *"wrapper.sh"*

Tahap berikutnya adalah membuat *file log "/var/log/conn.log"* untuk menyimpan koneksi dari alamat IP penyerang atau alamat IP pihak eksternal yang mengakses *binary*. Selain itu, buat *file log "/var/log/crash.log"* untuk menyimpan pelanggaran yang dilakukan oleh penyerang selama proses eksploitasi. Perintah untuk membuat kedua *file log* tersebut tertera pada Gambar 4.35.

```
ctf@VM-2:/etc/apparmor.d$ sudo mkdir -pm777 /var/log/labs/
ctf@VM-2:/etc/apparmor.d$ sudo chown -R $USER:$USER /var/log/labs/
ctf@VM-2:/etc/apparmor.d$ ls -lah /var/log/labs/
total 8.0K
drwxrwxrwx  2 ctf  ctf    4.0K Jul 14 11:09 .
drwxrwxr-x 13 root syslog 4.0K Jul 14 11:09 ..
ctf@VM-2:/etc/apparmor.d$
```

Gambar 4. 35 Membuat Direktori *Log* Untuk Monitoring Koneksi dan Pelanggaran Selama Proses Eksploitasi Berlangsung

Kedua *file binary* selanjutnya dapat dijalankan secara permanen menggunakan *"socat"* dengan membuat *file Systemd service*. *File* tersebut dapat disimpan ke dalam direktori *"/etc/systemd/system/"* dengan nama *"ctf-bof1.service"*. Isi dari *file service "/etc/systemd/system/ctf-bof1.service"* untuk menjalankan *binary* "BOF-1" secara permanen tertera pada Gambar 4.36.

```
[Unit]
Description=CTF Challenge BOF-1 (port 10001, tanpa proteksi)
After=network.target

[Service]
Type=simple
User=ctf
WorkingDirectory=/opt/labs/bof/bof-1
ExecStart=/usr/bin/socat -4 TCP4-LISTEN:10001,reuseaddr,fork EXEC:"/opt/labs/wrapper.sh /opt/labs/bof/bof-1/bof1",stderr
Restart=always
RestartSec=2

[Install]
WantedBy=multi-user.target
```

Gambar 4. 36 *File Service* Untuk Menjalankan *Binary* "BOF-1"

Buat *file Systemd service* yang serupa untuk *file binary* "FMT-1" untuk dijalankan secara permanen. Isi *file service "/etc/systemd/system/ctf-fmt1.service"* untuk menjalankan *binary* "FMT-1" secara permanen tertera pada Gambar 4.37.

```
[Unit]
Description=CTF Challenge FMT-1 (port 10002, tanpa proteksi)
After=network.target

[Service]
Type=simple
User=ctf
WorkingDirectory=/opt/labs/fmt/fmt-1
ExecStart=/usr/bin/socat -4 TCP4-LISTEN:10002,reuseaddr,fork EXEC:"/opt/labs/wrapper.sh /opt/labs/fmt/fmt-1/fmt1",stderr
Restart=always
RestartSec=2

[Install]
WantedBy=multi-user.target
```

Gambar 4. 37 *File Service* Untuk Menjalankan *Binary* "FMT-1"

Tahap selanjutnya adalah mengaktifkan kedua *file service* tersebut agar dapat berjalan di belakang layar secara permanen. Perintah untuk mengaktivasi kedua *file service* untuk "BOF-1" dan "FMT-1" secara permanen tertera pada Gambar 4.38.

```
ctf@VM-1:/opt/labs$ cd /etc/systemd/system
ctf@VM-1:/etc/systemd/system$ sudo vim ctf-bof1.service
ctf@VM-1:/etc/systemd/system$ sudo vim ctf-fmt1.service
ctf@VM-1:/etc/systemd/system$ sudo systemctl enable --now ctf-bof1.service
Created symlink /etc/systemd/system/multi-user.target.wants/ctf-bof1.service → /etc/systemd/system/ctf-bof1.service.
ctf@VM-1:/etc/systemd/system$ sudo systemctl enable --now ctf-fmt1.service
Created symlink /etc/systemd/system/multi-user.target.wants/ctf-fmt1.service → /etc/systemd/system/ctf-fmt1.service.
ctf@VM-1:/etc/systemd/system$
```

Gambar 4. 38 Aktivasi *Service Binary* Secara Permanen di VM-1

Setelah *binary* tersebut dijalankan sebagai *service*, umumnya penyerang mengakses *binary* tersebut melalui jaringan menggunakan alat bernama *"netcat"* atau *library pwntools* untuk tahap eksploitasi lebih lanjut.

4.3.3.2 Menjalankan *File ELF Binary* di VM-2

Berbeda halnya dengan VM-1, VM-2 melakukan pembatasan akses *(hardening)* pada *binary* menggunakan modul AppArmor. AppArmor akan menolak pelanggaran yang dilakukan oleh penyerang selama proses eksploitasi berlangsung. Untuk bisa menggunakan AppArmor di Ubuntu server, perlu instalasi utilitasnya terlebih dahulu. Selain itu, pastikan jika alat *"socat"* sudah terpasang di VM-2 agar *binary* dapat dijalankan melalui jaringan di *port* yang telah ditentukan pada Tabel 4.4. Perintah untuk instalasi utilitas AppArmor di VM-2 tertera pada Gambar 4.39.

```
ubuntu@VM-2:~$ sudo apt install -y socat apparmor-utils
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
socat is already the newest version (1.8.0.0-4ubuntu0.1).
apparmor-utils is already the newest version (4.0.1really4.0.1-0ubuntu0.24.04.7).
The following packages were automatically installed and are no longer required:
  libfwupd2 libgusb2
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
ubuntu@VM-2:~$
```

Gambar 4. 39 Instalasi *Socat* dan Utilitas AppArmor di VM-2

Sama seperti di VM-1, tahap selanjutnya adalah membuat *file flag* yang dapat dibaca oleh penyerang dan dikirim ke platform web CTFd untuk mendapatkan poin. Proses pembuatan file flag ditunjukkan pada Gambar 4.40.

```
ctf@VM-2:~/research-lab$ cd /opt/labs/
ctf@VM-2:/opt/labs$ cd bof/bof-2/
ctf@VM-2:/opt/labs/bof/bof-2$ touch flag.txt
ctf@VM-2:/opt/labs/bof/bof-2$ vim flag.txt
ctf@VM-2:/opt/labs/bof/bof-2$ cd ../bof-3/
ctf@VM-2:/opt/labs/bof/bof-3$ touch flag.tct
ctf@VM-2:/opt/labs/bof/bof-3$ vim flag.txt
ctf@VM-2:/opt/labs/bof/bof-3$ cd ../../fmt/fmt-2/
ctf@VM-2:/opt/labs/fmt/fmt-2$ touch flag-$(cat /proc/sys/kernel/random/uuid).txt
ctf@VM-2:/opt/labs/fmt/fmt-2$ vim flag-ba3007b6-9afe-4071-a6fa-4516eaaa27a3.txt
ctf@VM-2:/opt/labs/fmt/fmt-2$ cd ../fmt-3/
ctf@VM-2:/opt/labs/fmt/fmt-3$ touch flag-$(cat /proc/sys/kernel/random/uuid).txt
ctf@VM-2:/opt/labs/fmt/fmt-3$ vim flag-72f1aa12-2881-4432-ac3e-74991a5a2188.txt
ctf@VM-2:/opt/labs/fmt/fmt-3$ cd ../..
ctf@VM-2:/opt/labs$
```

Gambar 4. 40 Membuat *File Flag* di VM-2

Langkah berikutnya adalah membuat profil AppArmor, yang dibagi menjadi 2 jenis, yaitu profil longgar dan profil ketat. Tujuannya adalah untuk mengidentifikasi dampak lanjutan eksploitasi yang penyerang berhasil lakukan. Di mana, harapannya adalah penyerang tidak berhasil mengeksekusi perintah-perintah dan *file-file* sensitif tertentu pada VM-2 setelah berhasil mendapatkan akses *shell*.

Sesuai dengan Tabel 4.4, terdapat 4 *binary* yang akan dijalankan di VM-2, yaitu "BOF-2", "FMT-2", "BOF-3", dan "FMT-3". Profil AppArmor longgar dipasangkan ke *file binary* "BOF-2" dan "FMT-2", sedangkan profil ketat dipasangkan ke *file binary* "BOF-3" dan "FMT-3". *File profil* AppArmor dapat dibuat dan didefinisikan di dalam direktori *"/etc/apparmor.d/"*.

Profil AppArmor longgar *"/etc/apparmor.d/apparmor-bof2"* yang dipasangkan pada *file binary* "BOF-2" ditunjukkan pada Gambar 4.41.

```
#include <tunables/global>

/opt/labs/fmt/fmt-2/fmt2 {
    #include <abstractions/base>

    /opt/labs/fmt/fmt-2/fmt2 mr,
    /opt/labs/fmt/fmt-2/ r, # izinkan listing current working dir
    /opt/labs/fmt/fmt-2/flag-*.txt r, # file flag boleh dibaca

    # Izinkan stdbuf
    /usr/libexec/coreutils/libstdbuf.so mr,

    # Longgar: nyaris tidak membatasi dampak lanjutan
    /etc/passwd r,
    /etc/hostname r,
    /etc/hosts r,

    # Izinkan eksekusi shell tanpa mewarisi profil binary (unconfined execute)
    /bin/sh ux,
    /bin/bash ux,
    /usr/bin/dash ux,

    network inet stream,
}
```

Gambar 4. 41 Profil AppArmor Longgar yang Dipasangkan pada *File Binary* "BOF-2"

Profil AppArmor longgar *"/etc/apparmor.d/apparmor-fmt2"* yang dipasangkan pada *file binary* "FMT-2" ditunjukkan pada Gambar 4.42.

```
#include <tunables/global>

/opt/labs/fmt/fmt-2/fmt2 {
    #include <abstractions/base>

    /opt/labs/fmt/fmt-2/fmt2 mr,
    /opt/labs/fmt/fmt-2/ r, # izinkan listing current working dir
    /opt/labs/fmt/fmt-2/flag-*.txt r, # file flag boleh dibaca

    # Izinkan stdbuf
    /usr/libexec/coreutils/libstdbuf.so mr,

    # Longgar: nyaris tidak membatasi dampak lanjutan
    /etc/passwd r,
    /etc/hostname r,
    /etc/hosts r,

    # Izinkan eksekusi shell tanpa mewarisi profil binary (unconfined execute)
    /bin/sh ux,
    /bin/bash ux,
    /usr/bin/dash ux,

    network inet stream,
}
```

Gambar 4. 42 Profil AppArmor Longgar yang Dipasangkan pada *File Binary* "FMT-2"

Dapat dilihat pada profil longgar yang tertera pada Gambar 4.41 dan Gambar 4.42, di mana eksekusi *shell* diizinkan sepenuhnya tanpa pembatasan apapun. Selain itu, akses baca *file* sensitif juga diizinkan, yang mencakup *file "/etc/passwd"*, *"/etc/hostname"*, dan *"/etc/hosts"*. Perbedannya sangat kontras dengan profil AppArmor ketat. Profil AppArmor ketat *"/etc/apparmor.d/apparmor-bof3"* yang dipasangkan pada *file binary* "BOF-3" ditunjukkan pada Gambar 4.43.

```
#include <tunables/global>

/opt/labs/bof/bof-3/bof3 {
    #include <abstractions/base>

    /opt/labs/bof/bof-3/bof3 mr,
    /opt/labs/bof/bof-3/flag.txt r, # file flag boleh dibaca

    # Izinkan stdbuf
    /usr/libexec/coreutils/libstdbuf.so mr,

    # Ketat: membatasi dampak lanjutan (least privilege)
    deny /etc/passwd r,
    deny /etc/hostname r,
    deny /etc/hosts r,
    deny /root/** r,

    network inet stream,
}
```

Gambar 4. 43 Profil AppArmor Ketat yang Dipasangkan pada *File Binary* "BOF-3"

Profil AppArmor ketat *"/etc/apparmor.d/apparmor-fmt3"* yang dipasangkan pada *file binary* "FMT-3" ditunjukkan pada Gambar 4.44.

```
#include <tunables/global>

/opt/labs/fmt/fmt-3/fmt3 {
    #include <abstractions/base>

    /opt/labs/fmt/fmt-3/fmt3 mr,
    /opt/labs/fmt/fmt-3/ r, # izinkan listing current working dir
    /opt/labs/fmt/fmt-3/flag-*.txt r, # file flag boleh dibaca

    # Izinkan stdbuf
    /usr/libexec/coreutils/libstdbuf.so mr,

    # Ketat: membatasi dampak lanjutan (least privilege)
    deny /etc/passwd r,
    deny /etc/hostname r,
    deny /etc/hosts r,
    deny /root/** r,

    # Berikan akses shell, tetapi hanya perintah yang diizinkan
    /bin/sh ix,
    /bin/bash ix,
    /usr/bin/dash ix,
    /bin/cat ix,
    /usr/bin/cat ix,
    /bin/ls ix,
    /usr/bin/ls ix,
    /bin/id ix,
    /usr/bin/id ix,

    network inet stream,
}
```

Gambar 4. 44 Profil AppArmor Ketat yang Dipasangkan pada *File Binary* "FMT-3"

Profil AppArmor yang telah didefinsikan untuk masing-masing *file binary*, kemudian dikompilasi *(apparmor_parser)* ke *format binary* yang dapat dimengerti oleh Kernel Linux. Setelah itu, profil tersebut diubah ke mode *Enforce (aa-enforce)* supaya AppArmor berperan aktif dalam menolak dan mencatat pelanggaran. Semua pelanggaran akan tercatat ke *file log* sistem, yaitu *"/var/log/syslog"*. Perintah untuk *parsing* profil AppArmor dan *load* ke mode *Enforce* ditunjukkan pada Gambar 4.45.

```
ctf@VM-2:/etc/systemd/system$ cd /etc/apparmor.d/
ctf@VM-2:/etc/apparmor.d$ sudo vim apparmor-bof2
ctf@VM-2:/etc/apparmor.d$ sudo vim apparmor-bof3
ctf@VM-2:/etc/apparmor.d$ sudo vim apparmor-fmt2
ctf@VM-2:/etc/apparmor.d$ sudo vim apparmor-fmt3
ctf@VM-2:/etc/apparmor.d$ sudo apparmor_parser -r /etc/apparmor.d/apparmor-bof2
ctf@VM-2:/etc/apparmor.d$ sudo apparmor_parser -r /etc/apparmor.d/apparmor-fmt2
ctf@VM-2:/etc/apparmor.d$ sudo apparmor_parser -r /etc/apparmor.d/apparmor-bof3
ctf@VM-2:/etc/apparmor.d$ sudo apparmor_parser -r /etc/apparmor.d/apparmor-fmt3
ctf@VM-2:/etc/apparmor.d$ sudo aa-enforce /etc/apparmor.d/apparmor-bof2
Setting /etc/apparmor.d/apparmor-bof2 to enforce mode.
ctf@VM-2:/etc/apparmor.d$ sudo aa-enforce /etc/apparmor.d/apparmor-fmt2
Setting /etc/apparmor.d/apparmor-fmt2 to enforce mode.
ctf@VM-2:/etc/apparmor.d$ sudo aa-enforce /etc/apparmor.d/apparmor-bof3
Setting /etc/apparmor.d/apparmor-bof3 to enforce mode.
ctf@VM-2:/etc/apparmor.d$ sudo aa-enforce /etc/apparmor.d/apparmor-fmt3
Setting /etc/apparmor.d/apparmor-fmt3 to enforce mode.
ctf@VM-2:/etc/apparmor.d$ aa-status | grep -E "bof-*|fmt-*"
You do not have enough privilege to read the profile set.
ctf@VM-2:/etc/apparmor.d$ sudo aa-status | grep -E "bof-*|fmt-*"
   /opt/labs/bof/bof-2/bof2
   /opt/labs/bof/bof-3/bof3
   /opt/labs/fmt/fmt-2/fmt2
   /opt/labs/fmt/fmt-3/fmt3
ctf@VM-2:/etc/apparmor.d$
```

Gambar 4. 45 *Load* Keempat Profil *Binary* ke Mode *Enforce*

Pastikan jika *service* AppArmor telah berjalan di belakang layar dan telah diaktifkan secara permanen. Perintah untuk mengaktivasi *service* AppArmor secara permanen tertera pada Gambar 4.46.

```
ctf@VM-2:/etc/apparmor.d$ sudo systemctl enable --now apparmor.service
Synchronizing state of apparmor.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable apparmor
ctf@VM-2:/etc/apparmor.d$ sudo systemctl is-enabled apparmor.service
enabled
ctf@VM-2:/etc/apparmor.d$
```

Gambar 4. 46 Aktivasi *Service* AppArmor Secara Permanen

Terdapat tahap tambahan seperti di VM-1, yaitu membuat *file "wrapper.sh"*. *File* tersebut digunakan untuk menghubungkan informasi IP penyerang yang tersimpan di *log "socat"* dengan riwayat pelanggaran pada *log* sistem *"/var/log/syslog"*. *File "wrapper.sh"* di VM-2 tertera pada Gambar 4.47.

```
#!/bin/bash
# Usage: /opt/labs/wrapper.sh <path-binary>
# Run: EXEC:/opt/labs/wrapper.sh /opt/labs/bof/bof-1/bof1
# Used for: menjembatani SOCAT_PEERADDR (IP penyerang) dengan event koneksi & crash,
#         lalu menuliskannya ke /var/log/labs/ agar dapat dibaca Promtail → Loki.

BIN="$1"
CONN_LOG=/var/log/labs/conn.log
CRASH_LOG=/var/log/labs/crash.log
IP="${SOCAT_PEERADDR:-unknown}"
PORT="${SOCAT_SOCKPORT:-unknown}"

# Catat setiap koneksi masuk (baseline VM-1 & analisis flooding)
printf '%s CONNECT dari %s ke port %s (%s)\n' \
    "$(date '+%Y-%m-%d %H:%M:%S')" "$IP" "$PORT" "$BIN" >> "$CONN_LOG"

# Jalankan binary; stdin/stdout sudah terhubung ke socket oleh socat
stdbuf -o0 "$BIN"
status=$?

# 128+N = proses diterminasi oleh sinyal N. 139=SIGSEGV, 134=SIGABRT
if [ "$status" -ge 128 ]; then
    sig=$((status - 128))
    printf '%s CRASH terdeteksi dari %s port %s (%s exit=%d signal=%d)\n' \
        "$(date '+%Y-%m-%d %H:%M:%S')" "$IP" "$PORT" "$BIN" "$status" "$sig" >> "$CRASH_LOG"
fi

exit "$status"
```

Gambar 4. 47 Buat *File "wrapper.sh"* di VM-2 Untuk Menghubungkan *Log Socat* Dengan *Log* Sistem

Berikan hak izin eksekusi pada *file "wrapper.sh"*. Perintah untuk menambahkan hak izin eksekusi pada *file "wrapper.sh"* tertera pada Gambar 4.48.

```
ctf@VM-2:/opt/labs$ chmod +x wrapper.sh
ctf@VM-2:/opt/labs$ ls -lah wrapper.sh
-rwxrwxr-x 1 ctf ctf 1023 Jul 14 10:30 wrapper.sh
ctf@VM-2:/opt/labs$
```

Gambar 4. 48 Berikan Hak Izin Eksekusi pada *File "wrapper.sh"* di VM-2

Kedua *file log* yang telah disatukan oleh *file "wrapper.sh"* perlu disimpan ke dalam *file log* khusus. *Log* tersebut terdiri dari 2 *file*, yaitu *"/var/log/labs/conn.log"* yang memuat informasi alamat IP penyerang atau pihak eksternal yang mengakses *binary*, serta *"/var/log/labs/crash.log"* untuk menyimpan pelanggaran yang

dilakukan oleh penyerang selama proses eksploitasi. Untuk itu, buat terlebih dahulu direktori untuk menyimpan kedua *file log* tersebut. Pembuatan direktori log tertera pada Gambar 4.49.

```
ctf@VM-2:/etc/apparmor.d$ sudo mkdir -pm777 /var/log/labs/
ctf@VM-2:/etc/apparmor.d$ sudo chown -R $USER:$USER /var/log/labs/
ctf@VM-2:/etc/apparmor.d$ ls -lah /var/log/labs/
total 8.0K
drwxrwxrwx  2 ctf  ctf    4.0K Jul 14 11:09 .
drwxrwxr-x 13 root syslog 4.0K Jul 14 11:09 ..
ctf@VM-2:/etc/apparmor.d$
```

Gambar 4. 49 Membuat Direktori *Log* Untuk Monitoring Koneksi dan Pelanggaran Selama Proses Eksploitasi Berlangsung

Setelah direktori untuk menyimpan *file log* khusus telah dibuat, langkah berikutnya adalah membuat *file Systemd service* untuk menjalankan keempat *file binary* secara permanen. Dimulai dengan membuat *file service* terhadap *file* "BOF-2" terlebih dahulu. Isi *file service "/etc/systemd/system/ctf-bof2.service"* untuk menjalankan *binary* "BOF-2" secara permanen tertera pada Gambar 4.50.

```
[Unit]
Description=CTF Challenge BOF-2 (port 10003, AppArmor longgar)
After=network.target

[Service]
Type=simple
User=ctf
WorkingDirectory=/opt/labs/bof/bof-2
ExecStart=/usr/bin/socat -4 TCP4-LISTEN:10003,reuseaddr,fork EXEC:"/opt/labs/wrapper.sh /opt/labs/bof/bof-2/bof2",stderr
Restart=always
RestartSec=2

[Install]
WantedBy=multi-user.target
```

Gambar 4. 50 *File Service* Untuk Menjalankan *Binary* "BOF-2"

Isi dari *file service "/etc/systemd/system/ctf-fmt2.service"* untuk menjalankan *binary* "FMT-2" secara permanen tertera pada Gambar 4.51.

```
[Unit]
Description=CTF Challenge FMT-2 (port 10004, AppArmor longgar)
After=network.target

[Service]
Type=simple
User=ctf
WorkingDirectory=/opt/labs/fmt/fmt-2
ExecStart=/usr/bin/socat -4 TCP4-LISTEN:10004,reuseaddr,fork EXEC:"/opt/labs/wrapper.sh /opt/labs/fmt/fmt-2/fmt2",stderr
Restart=always
RestartSec=2

[Install]
WantedBy=multi-user.target
```

Gambar 4. 51 *File Service* Untuk Menjalankan *Binary* "FMT-2"

Isi dari *file service "/etc/systemd/system/ctf-bof3.service"* untuk menjalankan *binary* "BOF-3" secara permanen tertera pada Gambar 4.52.

```
[Unit]
Description=CTF Challenge BOF-3 (port 10005, AppArmor ketat)
After=network.target

[Service]
Type=simple
User=ctf
WorkingDirectory=/opt/labs/bof/bof-3
ExecStart=/usr/bin/socat -4 TCP4-LISTEN:10005,reuseaddr,fork EXEC:"/opt/labs/wrapper.sh /opt/labs/bof/bof-3/bof3",stderr
Restart=always
RestartSec=2

[Install]
WantedBy=multi-user.target
```

Gambar 4. 52 *File Service* Untuk Menjalankan *Binary* "BOF-3"

Isi dari *file service "/etc/systemd/system/ctf-fmt3.service"* untuk menjalankan *binary* "FMT-3" secara permanen tertera pada Gambar 4.53.

```
[Unit]
Description=CTF Challenge FMT-3 (port 10006, AppArmor ketat)
After=network.target

[Service]
Type=simple
User=ctf
WorkingDirectory=/opt/labs/fmt/fmt-3
ExecStart=/usr/bin/socat -4 TCP4-LISTEN:10006,reuseaddr,fork EXEC:"/opt/labs/wrapper.sh /opt/labs/fmt/fmt-3/fmt3",stderr
Restart=always
RestartSec=2

[Install]
WantedBy=multi-user.target
```

Gambar 4. 53 *File Service* Untuk Menjalankan *Binary* "FMT-3"

Aktivasi *file service* yang telah dibuat untuk menjalankan keempat *binary* di VM-2 secara permanen. Perintah untuk mengaktivasi keempat *binary* di VM-2 secara permanen ditunjukkan pada Gambar 4.54.

```
ctf@VM-2:/opt/labs$ cd /etc/systemd/system
ctf@VM-2:/etc/systemd/system$ sudo vim ctf-bof2.service
ctf@VM-2:/etc/systemd/system$ sudo vim ctf-fmt2.service
ctf@VM-2:/etc/systemd/system$ sudo vim ctf-bof3.service
ctf@VM-2:/etc/systemd/system$ sudo vim ctf-fmt3.service
ctf@VM-2:/etc/systemd/system$ sudo systemctl enable --now ctf-bof2.service
Created symlink /etc/systemd/system/multi-user.target.wants/ctf-bof2.service → /etc/systemd/system/ctf-bof2.service.
ctf@VM-2:/etc/systemd/system$ sudo systemctl enable --now ctf-fmt2.service
Created symlink /etc/systemd/system/multi-user.target.wants/ctf-fmt2.service → /etc/systemd/system/ctf-fmt2.service.
ctf@VM-2:/etc/systemd/system$ sudo systemctl enable --now ctf-bof3.service
Created symlink /etc/systemd/system/multi-user.target.wants/ctf-bof3.service → /etc/systemd/system/ctf-bof3.service.
ctf@VM-2:/etc/systemd/system$ sudo systemctl enable --now ctf-fmt3.service
Created symlink /etc/systemd/system/multi-user.target.wants/ctf-fmt3.service → /etc/systemd/system/ctf-fmt3.service.
ctf@VM-2:/etc/systemd/system$
```

Gambar 4. 54 Aktivasi *Service Binary* Secara Permanen di VM-2

4.3.3.3 Menjalankan *File ELF Binary* di VM-3

Konfigurasi Fail2Ban dilakukan di VM-3 untuk memblokir alamat IP penyerang terhadap model serangan melalui jaringan. Dalam penelitian ini, penyerang yang berupaya melakukan pembanjiran koneksi *(TCP connection flooding)* lebih dari 50 kali dalam kurun waktu 1 menit atau spam karakter hingga menyebabkan aplikasi atau *binary* menjadi *crash* sebanyak 3 kali, maka Fail2Ban secara otomatis akan memblokir alamat IP penyerang.

Proteksi penuh yang akan diimplementasikan di VM-3 membutuhkan utilitas AppArmor dan Fail2Ban itu sendiri. Pastikan juga jika alat *"socat"* telah terpasang di VM-3 untuk menjalankan *binary* melalui jaringan. Perintah untuk instalasi utilitas AppArmor dan Fail2Ban tertera pada Gambar 4.55.

```
ctf@VM-3: $ sudo apt install -y socat apparmor-utils fail2ban
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
socat is already the newest version (1.8.0.0-4ubuntu0.1).
apparmor-utils is already the newest version (4.0.1really4.0.1-0ubuntu0.24.04.7).
fail2ban is already the newest version (1.0.2-3ubuntu0.1).
The following packages were automatically installed and are no longer required:
  libfwupd2 libgusb2
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
ctf@VM-3: $
```

Gambar 4. 55 Instalasi Utilitas AppArmor dan Fail2Ban di VM-3

Sama seperti di VM-1 dan VM-2, di mana *file flag* perlu dibuat agar dapat diakses oleh penyerang dan dikirim di platform web CTFd untuk mendapatkan poin. Buat file flag sesuai dengan jumlah *binary* yang dijalankan di VM-3, yaitu ada 4 sesuai dengan yang tertera di Table 4.4.

```
ctf@VM-3:~/research-lab$ cd /opt/labs/
ctf@VM-3:/opt/labs$ cd bof/bof-4/
ctf@VM-3:/opt/labs/bof/bof-4$ touch flag-$(cat /proc/sys/kernel/random/uuid).txt
ctf@VM-3:/opt/labs/bof/bof-4$ vim flag-ce795bbe-4cb2-4a7b-8791-3c2023889777.txt
ctf@VM-3:/opt/labs/bof/bof-4$ cd ../bof-5/
ctf@VM-3:/opt/labs/bof/bof-5$ touch flag-$(cat /proc/sys/kernel/random/uuid).txt
ctf@VM-3:/opt/labs/bof/bof-5$ vim flag-c4158db3-ede5-44af-8fa2-4cd6f05f06ee.txt
ctf@VM-3:/opt/labs/bof/bof-5$ cd ../../fmt/
ctf@VM-3:/opt/labs/fmt$ cd fmt-4/
ctf@VM-3:/opt/labs/fmt/fmt-4$ touch flag-$(cat /proc/sys/kernel/random/uuid).txt
ctf@VM-3:/opt/labs/fmt/fmt-4$ vim flag-0fbf60a8-7f82-4f2b-ac71-52ea8c0918d1.txt
ctf@VM-3:/opt/labs/fmt/fmt-4$ cd ../fmt-5/
ctf@VM-3:/opt/labs/fmt/fmt-5$ touch flag-$(cat /proc/sys/kernel/random/uuid).txt
ctf@VM-3:/opt/labs/fmt/fmt-5$ vim flag-97e43fe2-69c5-4d01-8daa-547b955295b5.txt
ctf@VM-3:/opt/labs/fmt/fmt-5$ cd ../../
ctf@VM-3:/opt/labs$
```

Gambar 4. 56 Membuat *File Flag* di VM-3

Tahap berikutnya adalah membuat profil AppArmor sesuai dengan skenario yang tertera pada Tabel 4.4. Di mana, terdapat 2 jenis profil AppArmor, yaitu profil longgar dan profil ketat. Profil longgar akan dipasangkan ke *binary* "BOF-4" dan "FMT-4", sedangkan profil ketat akan dipasangkan ke *binary* "BOF-5" dan "FMT-5". Detail kedua profil tersebut mirip seperti profil yang ada di VM-2. Di mana, pada profil ketat, penyerang yang berhasil mendapatkan akses *shell*, tidak diizinkan mengeksekusi perintah seperti *"whoami"*, *"id"*, serta membuka file-file sensitif yang ada di server seperti *"/etc/passwd"*. Dengan demikian, dampak eksploitasi lanjutan dapat dicegah, karena penyerang hanya dapat membaca *file flag* saja, tetapi tidak bisa melakukan pengumpulan informasi lebih dalam untuk mendapatkan akses secara penuh *(persistent access)*.

Dimulai dengan membuat profil AppArmor longgar untuk dipasangkan pada *file binary* "BOF-4". *File* profil AppArmor dapat disimpan ke dalam direktori *"/etc/apparmor.d/"*. Isi profil *"/etc/apparmor.d/apparmor-bof4"* tertera pada Gambar 4.57.

```
#include <tunables/global>

/opt/labs/bof/bof-4/bof4 {
        #include <abstractions/base>

        /opt/labs/bof/bof-4/bof4 mr,
        /opt/labs/bof/bof-4/ r, # izinkan listing current working dir
        /opt/labs/bof/bof-4/flag-*.txt r, # file flag boleh dibaca

        # Izinkan stdbuf
        /usr/libexec/coreutils/libstdbuf.so mr,

        # Longgar: nyaris tidak membatasi dampak lanjutan
        /etc/passwd r,
        /etc/hostname r,
        /etc/hosts r,

        # Izinkan eksekusi shell tanpa mewarisi profil binary (unconfined execute)
        /bin/sh ux,
        /bin/bash ux,
        /usr/bin/dash ux,

        network inet stream,
}
```

Gambar 4. 57 Profil AppArmor Longgar yang Dipasangkan pada *Binary* "BOF-4"

Profil AppArmor longgar *"/etc/apparmor.d/apparmor-fmt4"* yang dipasangkan pada *file binary* "FMT-4" ditunjukkan pada Gambar 4.58.

```
#include <tunables/global>

/opt/labs/fmt/fmt-4/fmt4 {
        #include <abstractions/base>

        /opt/labs/fmt/fmt-4/fmt4 mr,
        /opt/labs/fmt/fmt-4/ r, # izinkan listing current working dir
        /opt/labs/fmt/fmt-4/flag-*.txt r, # file flag boleh dibaca

        # Izinkan stdbuf
        /usr/libexec/coreutils/libstdbuf.so mr,

        # Longgar: nyaris tidak membatasi dampak lanjutan
        /etc/passwd r,
        /etc/hostname r,
        /etc/hosts r,

        # Izinkan eksekusi shell tanpa mewarisi profil binary (unconfined execute)
        /bin/sh ux,
        /bin/bash ux,
        /usr/bin/dash ux,

        network inet stream,
}
```

Gambar 4. 58 Profil AppArmor Longgar yang Dipasangkan pada *Binary* "FMT-4"

Dilanjutkan dengan membuat profil AppArmor ketat *"/etc/apparmor.d/apparmor-bof5"* yang dipasangkan pada *file binary* "BOF-5" ditunjukkan pada Gambar 4.59.

```
#include <tunables/global>

/opt/labs/bof/bof-5/bof5 {
        #include <abstractions/base>

        /opt/labs/bof/bof-5/bof5 mr,
        /opt/labs/bof/bof-5/ r, # izinkan listing current working dir
        /opt/labs/bof/bof-5/flag-*.txt r, # file flag boleh dibaca

        # Izinkan stdbuf
        /usr/libexec/coreutils/libstdbuf.so mr,

        # Ketat: membatasi dampak lanjutan (least privilege)
        deny /etc/passwd r,
        deny /etc/hostname r,
        deny /etc/hosts r,
        deny /root/** r,

        # Ketat: Berikan akses shell, tetapi hanya perintah yang diizinkan
        /bin/sh ix,
        /bin/bash ix,
        /usr/bin/dash ix,
        /bin/cat ix,
        /usr/bin/cat ix,
        /bin/ls ix,
        /usr/bin/ls ix,
        /bin/id ix,
        /usr/bin/id ix,

        network inet stream,
}
```

Gambar 4. 59 Profil AppArmor Ketat yang Dipasangkan pada *Binary* "BOF-5"

Profil AppArmor ketat *"/etc/apparmor.d/apparmor-fmt5"* yang dipasangkan pada *file binary* "FMT-5" ditunjukkan pada Gambar 4.60.

```
#include <tunables/global>

/opt/labs/fmt/fmt-5/fmt5 {
        #include <abstractions/base>

        /opt/labs/fmt/fmt-5/fmt5 mr,
        /opt/labs/fmt/fmt-5/ r, # izinkan listing current working dir
        /opt/labs/fmt/fmt-5/flag-*.txt r, # file flag boleh dibaca

        # Izinkan stdbuf
        /usr/libexec/coreutils/libstdbuf.so mr,

        # Ketat: membatasi dampak lanjutan (least privilege)
        deny /etc/passwd r,
        deny /etc/hostname r,
        deny /etc/hosts r,
        deny /root/** r,

        # Ketat: Berikan akses shell, tetapi hanya perintah yang diizinkan
        /bin/sh ix,
        /bin/bash ix,
        /usr/bin/dash ix,
        /bin/cat ix,
        /usr/bin/cat ix,
        /bin/ls ix,
        /usr/bin/ls ix,
        /bin/id ix,
        /usr/bin/id ix,

        network inet stream,
}
```

Gambar 4. 60 Profil AppArmor Ketat yang Dipasangkan pada *Binary* "FMT-5"

Profil AppArmor yang telah didefinsikan untuk masing-masing *file binary*, kemudian dikompilasi *(apparmor_parser)* ke *format binary* yang dapat dimengerti oleh Kernel Linux. Setelah itu, profil tersebut diubah ke mode *Enforce (aa-enforce)* supaya AppArmor berperan aktif dalam menolak dan mencatat pelanggaran. Semua pelanggaran akan tercatat ke *file log* sistem, yaitu *"/var/log/syslog"*. Perintah untuk *parsing* profil AppArmor dan *load* ke mode *Enforce* ditunjukkan pada Gambar 4.61.

```
ctf@VM-3:/opt/labs$ cd /etc/apparmor.d/
ctf@VM-3:/etc/apparmor.d$ sudo vim apparmor-bof4
ctf@VM-3:/etc/apparmor.d$ sudo vim apparmor-fmt4
ctf@VM-3:/etc/apparmor.d$ sudo vim apparmor-bof5
ctf@VM-3:/etc/apparmor.d$ sudo vim apparmor-fmt5
ctf@VM-3:/etc/apparmor.d$ sudo apparmor_parser -r /etc/apparmor.d/apparmor-bof4
ctf@VM-3:/etc/apparmor.d$ sudo apparmor_parser -r /etc/apparmor.d/apparmor-fmt4
ctf@VM-3:/etc/apparmor.d$ sudo apparmor_parser -r /etc/apparmor.d/apparmor-bof5
ctf@VM-3:/etc/apparmor.d$ sudo apparmor_parser -r /etc/apparmor.d/apparmor-fmt5
ctf@VM-3:/etc/apparmor.d$ sudo aa-enforce /etc/apparmor.d/apparmor-bof4
Setting /etc/apparmor.d/apparmor-bof4 to enforce mode.
ctf@VM-3:/etc/apparmor.d$ sudo aa-enforce /etc/apparmor.d/apparmor-fmt4
Setting /etc/apparmor.d/apparmor-fmt4 to enforce mode.
ctf@VM-3:/etc/apparmor.d$ sudo aa-enforce /etc/apparmor.d/apparmor-bof5
Setting /etc/apparmor.d/apparmor-bof5 to enforce mode.
ctf@VM-3:/etc/apparmor.d$ sudo aa-enforce /etc/apparmor.d/apparmor-fmt5
Setting /etc/apparmor.d/apparmor-fmt5 to enforce mode.
ctf@VM-3:/etc/apparmor.d$ sudo aa-status | grep -E "bof-*|fmt-*"
   /opt/labs/bof/bof-4/bof4
   /opt/labs/bof/bof-5/bof5
   /opt/labs/fmt/fmt-4/fmt4
   /opt/labs/fmt/fmt-5/fmt5
ctf@VM-3:/etc/apparmor.d$
```

Gambar 4. 61 *Load* Keempat Profil *Binary* ke Mode *Enforce*

Pastikan *service* AppArmor telah diaktifkan secara permanen dan telah berjalan di belakang layar. Perintah untuk mengaktivasi *service* AppArmor secara permanen tertera pada Gambar 4.62.

```
ctf@VM-3:/etc/apparmor.d$ sudo systemctl enable --now apparmor.service
Synchronizing state of apparmor.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable apparmor
ctf@VM-3:/etc/apparmor.d$ sudo systemctl is-enabled apparmor.service
enabled
ctf@VM-3:/etc/apparmor.d$
```

Gambar 4. 62 Aktivasi *Service* AppArmor Secara Permanen

Fail2Ban yang telah terpasang di VM-3 selanjutnya dikonfigurasi untuk memblokir alamat IP penyerang berdasarkan upaya pelanggaran melalui jaringan. Konfigurasi pertama yang dilakukan adalah mendeteksi adanya upaya pembanjiran koneksi (*TCP connection flooding*). Mirip seperti upaya *Denial Distributed of Service* (DDoS), di mana pembanjiran koneksi dapat berakibat pada akses ke *binary* melalui jaringan yang melambat atau bahkan *lost connection*, jika tidak ada pembatasan jumlah koneksi pada 1 sumber alamat IP yang sama.

Buat *file* konfigurasi Fail2Ban dengan nama *"flood-detection.conf"* di dalam direktori *"/etc/fail2ban/filter.d/"* untuk mendeteksi pembanjiran koneksi yang berasal dari 1 sumber alamat IP yang sama. Gambar 4.63 menunjukkan konfigurasi Fail2Ban untuk mendeteksi upaya pembanjiran koneksi.

```
[Definition]
failregex = ^.*CONNECT dari <HOST>.*$
ignoreregex =
```

Gambar 4. 63 File Konfigurasi Fail2Ban Untuk Mendeteksi Pola Pembanjiran Koneksi

Selain konfigurasi Fail2Ban untuk mendeteksi pola pembanjiran koneksi, Fail2Ban juga bisa dikonfigurasi untuk mendeteksi pola *crash* yang berulang pada suatu *binary*. Terminasi pada *binary* diakibatkan oleh spam karakter atau upaya eksploitasi yang gagal, sehingga aplikasi tidak dapat berjalan dengan normal sesuai alur logikanya. Pola *crash* pada *binary* akan menghasilkan sinyal berupa nomor, seperti 11 atau *"Segmentation Fault"* (*SIGSEGV*) dan 6 *"Stack Smashing Detected"* (*SIGABRT*). Sinyal *crash* nomor 11 (*SIGSEGV*), berarti program yang tidak dapat *exit* dari program dengan benar, karena program mencoba akses alamat memori yang tidak dapat dijangkau *(invalid)*. Sedangkan, sinyal *crash* nomor 6 (*SIGABRT*), berarti *binary* yang memiliki proteksi *Stack Canary* mendeteksi adanya upaya *buffer overflow*, karena penyerang menginputkan karakter melebihi batas alokasi memori yang telah ditetapkan.

Buat *file* konfigurasi Fail2Ban dengan nama *"crash-detection.conf"* di dalam direktori *"/etc/fail2ban/filter.d/"* untuk mendeteksi pola *crash* berulang pada *binary*.

```
[Definition]
failregex = ^.*CRASH terdeteksi dari <HOST>.*$
ignoreregex =
```

Gambar 4. 64 File Konfigurasi Fail2Ban Untuk Mendeteksi Pola *Crash* pada *Binary*

Tahap selanjutnya masih konfigurasi Fail2Ban, tetapi untuk menentukan ambang batas pelanggaran yang dapat diterima oleh Fail2Ban. Jika pelanggaran yang dilakukan oleh penyerang melebihi ambang batas yang telah ditentukan, maka

Fail2Ban akan memblokir alamat IP penyerang. Adapun ambang batas upaya pembanjiran koneksi yang dapat diterima adalah 50 koneksi dalam durasi 1 menit pada 1 sumber alamat IP yang sama. Sedangkan, ambang batas upaya spam karakter hingga membuat *binary crash* dapat Fail2Ban terima sebanyak 3 kali selama durasi 10 menit. Lebih dari itu, maka Fail2Ban secara otomatis memblokir alamat IP penyerang supaya tidak bisa mengakses *binary* melalui jaringan.

File konfigurasi ambang batas Fail2Ban dapat dibuat dengan nama *"ctf.conf"* di dalam direktori *"/etc/fail2ban/jail.d/"*. Isi dari *file* konfigurasi *"/etc/fail2ban/jail.d/ctf.conf"* tertera pada Gambar 4.65.

```
[crash-detection]
enabled  = true
filter   = crash-detection
backend  = polling
logpath  = /var/log/labs/crash.log
maxretry = 3
findtime = 600
bantime  = -1

[flood-detection]
enabled  = true
filter   = flood-detection
backend  = polling
logpath  = /var/log/labs/conn.log
maxretry = 50
findtime = 60
bantime  = -1
```

Gambar 4. 65 File Konfigurasi Fail2Ban Untuk Menentukan Ambang Batas Pelanggaran

Cara Fail2Ban dalam mendeteksi pola pelanggaran, seperti pembanjiran koneksi dan spam karakter hingga menyebabkan *binary* menjadi *crash* bersumber pada *file log "/var/log/labs/conn.log"* dan *"/var/log/labs/crash.log"*. Di mana, kedua *file log* tersebut telah disatukan dari *file* log *socat* dengan log sistem *"/var/log/syslog"*.

Tahap selanjutnya adalah menjalankan *service* Fail2Ban secara permanen agar tidak perlu dinyalakan ulang setelah sistem operasi *booting*. Perintah untuk menjalankan *service* Fail2Ban secara permanen ditunjukkan pada Gambar 4.66.

```
ctf@VM-3:/etc/apparmor.d$ cd /etc/fail2ban/filter.d/
ctf@VM-3:/etc/fail2ban/filter.d$ sudo vim crash-detection.conf
ctf@VM-3:/etc/fail2ban/filter.d$ sudo vim flood-detection.conf
ctf@VM-3:/etc/fail2ban/filter.d$ cd ../jail.d/
ctf@VM-3:/etc/fail2ban/jail.d$ sudo vim ctf.conf
ctf@VM-3:/etc/fail2ban/jail.d$ sudo systemctl enable --now fail2ban.service
Synchronizing state of fail2ban.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable fail2ban
ctf@VM-3:/etc/fail2ban/jail.d$ sudo systemctl is-enabled fail2ban.service
enabled
ctf@VM-3:/etc/fail2ban/jail.d$
```

Gambar 4. 66 Jalankan Fail2Ban Secara Permanen di VM-3

Selain itu, pastikan konfigurasi ambang batas (*jail*) Fail2Ban telah aktif dan siap untuk mendeteksi adanya pelanggaran. Perintah untuk memeriksa konfigurasi ambang batas Fail2Ban tertera pada Gambar 4.67.

```
ctf@VM-3:/etc/fail2ban/jail.d$ sudo fail2ban-client status crash-detection
Status for the jail: crash-detection
|- Filter
|  |- Currently failed: 0
|  |- Total failed:     0
|  `- Journal matches:
`- Actions
   |- Currently banned: 0
   |- Total banned:     0
   `- Banned IP list:
ctf@VM-3:/etc/fail2ban/jail.d$ sudo fail2ban-client status flood-detection
Status for the jail: flood-detection
|- Filter
|  |- Currently failed: 0
|  |- Total failed:     0
|  `- Journal matches:
`- Actions
   |- Currently banned: 0
   |- Total banned:     0
   `- Banned IP list:
```

Gambar 4. 67 Konfigurasi Ambang Batas (*Jail*) Fail2Ban Telah Aktif di VM-3

Untuk menyatukan *log socat* dan *log* sistem *"/var/log/syslog"*, perlu membuat *file "wrapper.sh"*. *File "wrapper.sh"* di VM-3 sama seperti di VM-2, isinya ditunjukkan pada Gambar 4.68.

```
#!/bin/bash
# Usage: /opt/labs/wrapper.sh <path-binary>
# Run: EXEC:/opt/labs/wrapper.sh /opt/labs/bof/bof-1/bof1
# Used for: menjembatani SOCAT_PEERADDR (IP penyerang) dengan event koneksi & crash,
#         lalu menuliskannya ke /var/log/labs/ agar dapat dibaca Promtail → Loki.

BIN="$1"
CONN_LOG=/var/log/labs/conn.log
CRASH_LOG=/var/log/labs/crash.log
IP="${SOCAT_PEERADDR:-unknown}"
PORT="${SOCAT_SOCKPORT:-unknown}"

# Catat setiap koneksi masuk (baseline VM-1 & analisis flooding)
printf '%s CONNECT dari %s ke port %s (%s)\n' \
    "$(date '+%Y-%m-%d %H:%M:%S')" "$IP" "$PORT" "$BIN" >> "$CONN_LOG"

# Jalankan binary; stdin/stdout sudah terhubung ke socket oleh socat
stdbuf -o0 "$BIN"
status=$?

# 128+N = proses diterminasi oleh sinyal N. 139=SIGSEGV, 134=SIGABRT
if [ "$status" -ge 128 ]; then
    sig=$((status - 128))
    printf '%s CRASH terdeteksi dari %s port %s (%s exit=%d signal=%d)\n' \
        "$(date '+%Y-%m-%d %H:%M:%S')" "$IP" "$PORT" "$BIN" "$status" "$sig" >> "$CRASH_LOG"
fi

exit "$status"
```

Gambar 4. 68 Buat *File "wrapper.sh"* di VM-3 Untuk Menghubungkan *Log Socat* Dengan *Log* Sistem

Berikan hak izin eksekusi *file "wrapper.sh"* dengan perintah yang ditunjukkan pada Gambar 4.69.

```
ctf@VM-3:/opt/labs$ vim wrapper.sh
ctf@VM-3:/opt/labs$ chmod +x wrapper.sh
ctf@VM-3:/opt/labs$ ls -lah wrapper.sh
-rwxrwxr-x 1 ctf ctf 1023 Jul 14 19:42 wrapper.sh
ctf@VM-3:/opt/labs$
```

Gambar 4. 69 Berikan Hak Izin Eksekusi pada *File "wrapper.sh"* di VM-3

Kedua *file log* yang telah disatukan oleh *file "wrapper.sh"* perlu disimpan ke dalam *file log* khusus. *Log* tersebut terdiri dari 2 *file*, yaitu *"/var/log/labs/conn.log"* yang memuat informasi alamat IP penyerang atau pihak eksternal yang mengakses *binary*, serta *"/var/log/labs/crash.log"* untuk menyimpan pelanggaran yang dilakukan oleh penyerang selama proses eksploitasi. Untuk itu, buat terlebih dahulu direktori untuk menyimpan kedua *file log* tersebut.

Pembuatan direktori untuk kedua *file log* tersebut tertera pada Gambar 4.70.

```
ctf@VM-3:/opt/labs$ sudo mkdir -pm777 /var/log/labs/
ctf@VM-3:/opt/labs$ sudo chown -R $USER:$USER /var/log/labs/
ctf@VM-3:/opt/labs$ ls -lah /var/log/labs/
total 8.0K
drwxrwxrwx  2 ctf  ctf    4.0K Jul 14 19:42 .
drwxrwxr-x 14 root syslog 4.0K Jul 14 19:42 ..
ctf@VM-3:/opt/labs$
```

Gambar 4. 70 Membuat Direktori *Log* Untuk Monitoring Koneksi dan Pelanggaran Selama Proses Eksploitasi Berlangsung

Setelah direktori untuk menyimpan log koneksi alamat IP penyerang dan log crash telah dibuat, langkah berikutnya adalah membuat *file Systemd service* untuk menjalankan keempat *file binary* yang tertera pada Tabel 4.4 secara permanen. Dimulai dengan membuat *file service* terhadap *file* "BOF-4" terlebih dahulu. Isi *file service "/etc/systemd/system/ctf-bof4.service"* untuk menjalankan *binary* "BOF-4" secara permanen tertera pada Gambar 4.71.

```
[Unit]
Description=CTF Challenge BOF-4 (port 10007, AppArmor longgar + Fail2Ban)
After=network.target

[Service]
Type=simple
User=ctf
WorkingDirectory=/opt/labs/bof/bof-4
ExecStart=/usr/bin/socat -4 TCP4-LISTEN:10007,reuseaddr,fork EXEC:"/opt/labs/wrapper.sh /opt/labs/bof/bof-4/bof4",stderr
Restart=always
RestartSec=2

[Install]
WantedBy=multi-user.target
```

Gambar 4. 71 *File Service* Untuk Menjalankan *Binary* "BOF-4"

Isi dari *file service "/etc/systemd/system/ctf-fmt4.service"* untuk menjalankan *binary* "FMT-4" secara permanen tertera pada Gambar 4.72.

```
[Unit]
Description=CTF Challenge FMT-4 (port 10008, AppArmor longgar + Fail2Ban)
After=network.target

[Service]
Type=simple
User=ctf
WorkingDirectory=/opt/labs/fmt/fmt-4
ExecStart=/usr/bin/socat -4 TCP4-LISTEN:10008,reuseaddr,fork EXEC:"/opt/labs/wrapper.sh /opt/labs/fmt/fmt-4/fmt4",stderr
Restart=always
RestartSec=2

[Install]
WantedBy=multi-user.target
```

Gambar 4. 72 *File Service* Untuk Menjalankan *Binary* "FMT-4"

Isi dari *file service "/etc/systemd/system/ctf-bof5.service"* untuk menjalankan *binary* "BOF-5" secara permanen tertera pada Gambar 4.73.

```
[Unit]
Description=CTF Challenge BOF-5 (port 10009, AppArmor ketat + Fail2Ban)
After=network.target

[Service]
Type=simple
User=ctf
WorkingDirectory=/opt/labs/bof/bof-5
ExecStart=/usr/bin/socat -4 TCP4-LISTEN:10009,reuseaddr,fork EXEC:"/opt/labs/wrapper.sh /opt/labs/bof/bof-5/bof5",stderr
Restart=always
RestartSec=2

[Install]
WantedBy=multi-user.target
```

Gambar 4. 73 *File Service* Untuk Menjalankan *Binary* "BOF-5"

Isi dari *file service "/etc/systemd/system/ctf-fmt5.service"* untuk menjalankan *binary* "FMT-5" secara permanen tertera pada Gambar 4.74.

```
[Unit]
Description=CTF Challenge FMT-5 (port 10010, AppArmor ketat + Fail2Ban)
After=network.target

[Service]
Type=simple
User=ctf
WorkingDirectory=/opt/labs/fmt/fmt-5
ExecStart=/usr/bin/socat -4 TCP4-LISTEN:10010,reuseaddr,fork EXEC:"/opt/labs/wrapper.sh /opt/labs/fmt/fmt-5/fmt5",stderr
Restart=always
RestartSec=2

[Install]
WantedBy=multi-user.target
```

Gambar 4. 74 *File Service* Untuk Menjalankan *Binary* "FMT-5"

Setelah *file service* untuk keempat *binary* telah dibuat, selanjutnya adalah mengaktivasi keempat *binary* tersebut secara permanen supaya bisa diakses melalui jaringan. Perintah untuk mengaktivasi keempat *binary* secara permanen tertera pada Gambar 4.75.

```
ctf@VM-3:/etc/apparmor.d$ cd /etc/systemd/system/
ctf@VM-3:/etc/systemd/system$ sudo vim ctf-bof4.service
ctf@VM-3:/etc/systemd/system$ sudo vim ctf-fmt4.service
ctf@VM-3:/etc/systemd/system$ sudo vim ctf-bof5.service
ctf@VM-3:/etc/systemd/system$ sudo vim ctf-fmt5.service
ctf@VM-3:/etc/systemd/system$ sudo systemctl enable --now ctf-bof4.service
Created symlink /etc/systemd/system/multi-user.target.wants/ctf-bof4.service → /etc/systemd/system/ctf-bof4.service.
ctf@VM-3:/etc/systemd/system$ sudo systemctl enable --now ctf-fmt4.service
Created symlink /etc/systemd/system/multi-user.target.wants/ctf-fmt4.service → /etc/systemd/system/ctf-fmt4.service.
ctf@VM-3:/etc/systemd/system$ sudo systemctl enable --now ctf-bof5.service
Created symlink /etc/systemd/system/multi-user.target.wants/ctf-bof5.service → /etc/systemd/system/ctf-bof5.service.
ctf@VM-3:/etc/systemd/system$ sudo systemctl enable --now ctf-fmt5.service
Created symlink /etc/systemd/system/multi-user.target.wants/ctf-fmt5.service → /etc/systemd/system/ctf-fmt5.service.
ctf@VM-3:/etc/systemd/system$
```

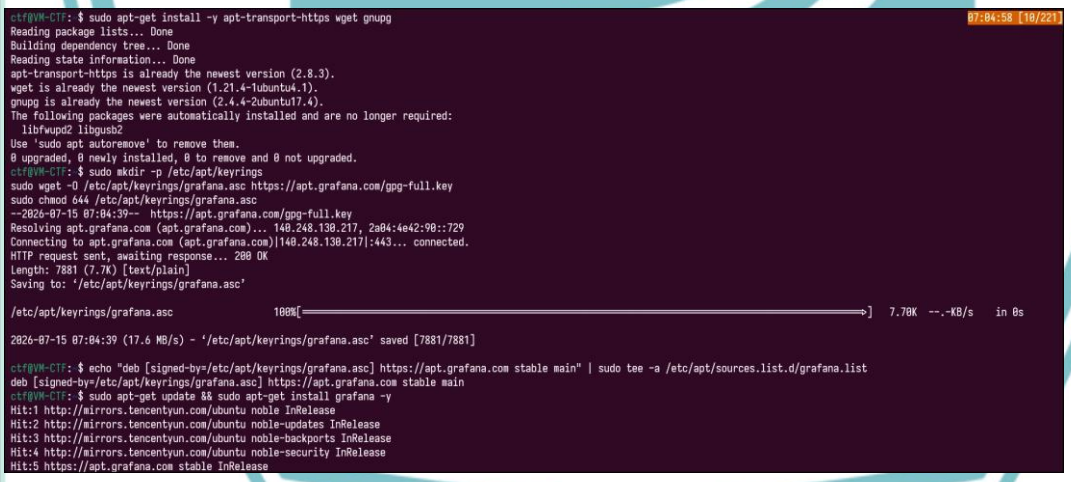
Gambar 4. 75 Aktivasi *Service Binary* Secara Permanen di VM-3

4.3.4 Instalasi Alat Monitoring

Alat monitoring tersentralisasi di server pusat, di mana Grafana dapat diakses pada *port* 3000 (*default*) dengan Prometheus pada *port* 9099 sebagai *data source* untuk mendapatkan metrik sumber daya secara *real-time* di ketiga VM. Selain itu, ada Grafana Loki yang berjalan di *port* 3100 (*default*) sebagai *data source* untuk memantau log aktivitas pelanggaran yang dilakukan saat proses eksploitasi *binary* di ketiga VM.

4.3.4.1 Instalasi Grafana

Grafana merupakan alat yang digunakan untuk memvisualisasikan data mentah (*data source*) dari Prometheus yang berupa angka metrik sumber daya pada setiap VM menjadi grafik *real-time* yang mudah dipahami. Instalasi Grafana dapat dilakukan dengan menambahkan repositori *apt* resmi Grafana terlebih dahulu ke server pusat. Kemudian, instalasi Grafana menggunakan perintah *"apt install grafana -y"*. Perintah untuk menambahkan repositori *apt* dan pemasangan Grafana di Ubuntu Server tertera pada Gambar 4.76.

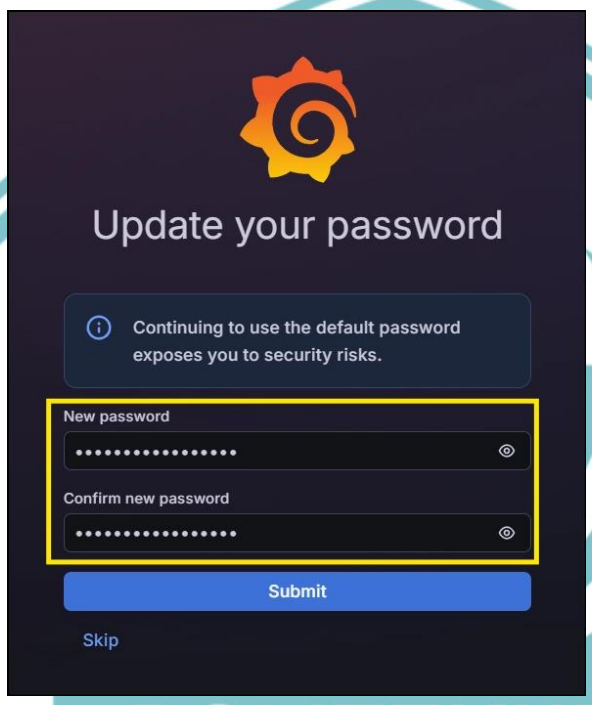


Gambar 4. 76 Instalasi Grafana di Server Pusat

Setelah Grafana berhasil terpasang di server pusat, jalankan layanan Grafana secara permanen di belakang layar menggunakan perintah yang tertera pada Gambar 4.77.

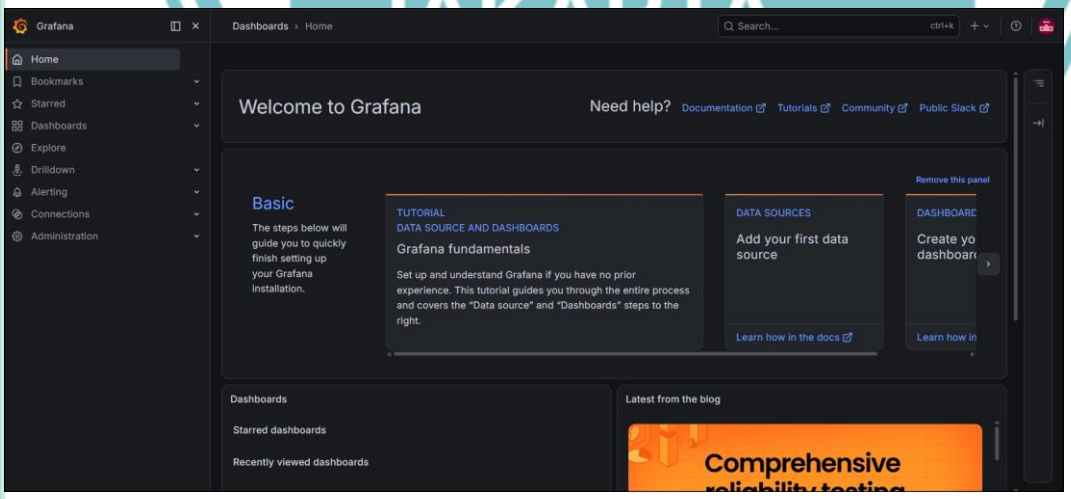
Gambar 4. 77 Jalankan Grafana Secara Permanen

Setelah itu, pastikan Grafana dapat diakses dari internet atau publik di *port* default, yaitu *port* 3000. Jika berhasil terpasang, akan tampil halaman *login* Grafana. Untuk autentikasi bawaan Grafana bisa menggunakan *username "admin"* dan *password "admin"*. Oleh karena itu, penting untuk mengubah *password* bawaan yang masih sangat lemah. Halaman ubah *password* lama Grafana tertera pada Gambar 4.78.



Gambar 4. 78 Halaman Ubah *Password Default* Grafana

Setelah *password* lama sudah diubah, maka pengguna bisa melihat halaman dasbor Grafana. Gambar 4.79 merupakan tampilan halaman dasbor Grafana.



Gambar 4. 79 Halaman Dasbor Grafana

4.3.4.2 Instalasi *Node Exporter*

Node Exporter adalah ekstensi tambahan dari Prometheus yang bertugas sebagai *agent* di setiap VM. Fungsi dari agen tersebut adalah membaca metrik *resource* pada sistem operasi dan mengubah ke dalam format yang dapat dibaca oleh Prometheus. Instalasi *Node Exporter* dilakukan di semua VM dan dapat dilakukan dengan mengunduh *file binary Node Exporter* terlebih dahulu. Perintah untuk mengunduh *file binary Node Exporter* tertera pada Gambar 4.80.

```
ctf@VM-CTF:/tmp$ wget -q https://github.com/prometheus/node_exporter/releases/download/v1.12.1/node_exporter-1.12.1.linux-amd64.tar.gz
ctf@VM-CTF:/tmp$ ls -lah node_exporter-1.12.1.linux-amd64.tar.gz
-rw-rw-r-- 1 ctf ctf 12M Jul 14 19:10 node_exporter-1.12.1.linux-amd64.tar.gz
ctf@VM-CTF:/tmp$
```

Gambar 4. 80 Unduh *File Binary Node Exporter* ke Server Pusat (VM-CTF)

Ekstrak *file* unduhan *Node Exporter* dalam format *"tar.gz"* menggunakan alat *"tar"*. Perintah ekstraksi file unduhan *Node Exporter* tertera pada Gambar 4.81.

```
ctf@VM-CTF:/tmp$ tar -xzf node_exporter-1.12.1.linux-amd64.tar.gz
ctf@VM-CTF:/tmp$ cd node_exporter-1.12.1.linux-amd64/
ctf@VM-CTF:/tmp/node_exporter-1.12.1.linux-amd64$ ls -la
total 23368
drwxr-xr-x  2 ctf  ctf      4096 Jul 14 19:10 .
drwxrwxrwt 18 root root     4096 Jul 15 07:20 ..
-rw-r--r--  1 ctf  ctf     11357 Jul 14 19:10 LICENSE
-rwxr-xr-x  1 ctf  ctf  23900840 Jul 14 19:10 node_exporter
-rw-r--r--  1 ctf  ctf       463 Jul 14 19:10 NOTICE
ctf@VM-CTF:/tmp/node_exporter-1.12.1.linux-amd64$
```

Gambar 4. 81 Ekstrak File Unduhan *Node Exporter*

Setelah ekstraksi selesai, muncul satu *file binary Node Exporter*. Pindahkan *file binary* tersebut ke direktori *"/usr/local/bin/"*. Perintah memindahkan *file Node Exporter* ke direktori Linux binaries tertera pada Gambar 4.82.

```
ctf@VM-CTF:/tmp/node_exporter-1.12.1.linux-amd64$ sudo mv node_exporter /usr/local/bin/
ctf@VM-CTF:/tmp/node_exporter-1.12.1.linux-amd64$ ls -lah /usr/local/bin/ | grep "node_exporter"
-rwxr-xr-x  1 ctf  ctf   23M Jul 14 19:10 node_exporter
ctf@VM-CTF:/tmp/node_exporter-1.12.1.linux-amd64$
```

Gambar 4. 82 Pindahkan File *Binary Node Exporter* ke Direktori Linux *Binaries*

Untuk membuat *Node Exporter* berjalan secara permanen di belakang layar, maka diperlukan *file service Node Exporter*. Buat *file service Systemd* dengan nama *"node-exporter.service"* di dalam direktori *"/etc/systemd/system"*.

Isi file *"node-exporter.service"* ditunjukkan pada Gambar 4.83.

```
[Unit]
Description=Prometheus exporter for machine metrics

[Service]
Restart=always
ExecStart=/usr/local/bin/node_exporter
ExecReload=/bin/kill -HUP SMAINPID
TimeoutStopSec=20s
SendSIGKILL=no

[Install]
WantedBy=multi-user.target
```

Gambar 4. 83 Buat *File Service Node Exporter*

Jalankan *Node Exporter* sebagai Systemd *service* secara permanen menggunakan perintah yang tertera pada Gambar 4.84.

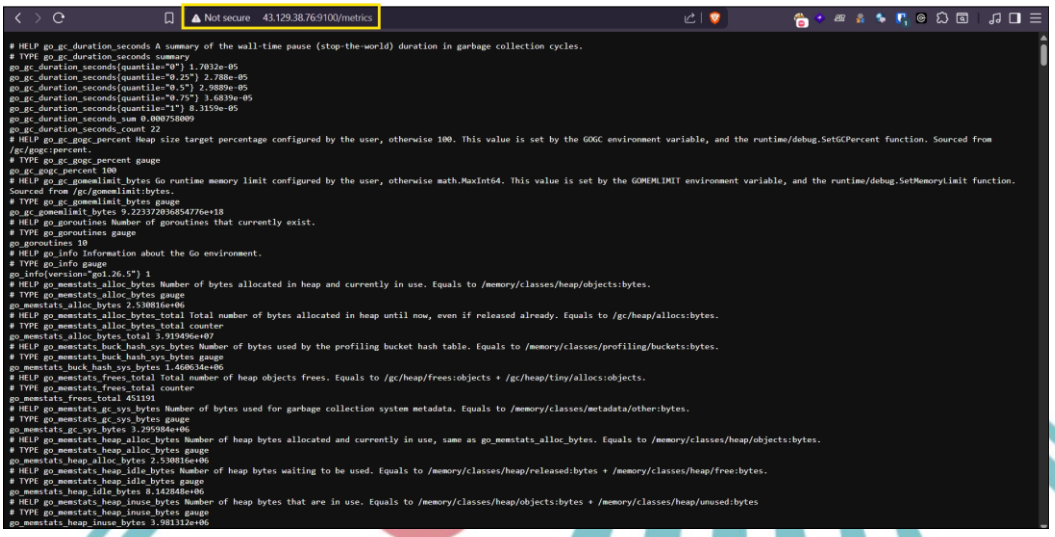
```
ctf@VM-CTF:/etc/systemd/system$ sudo vim node-exporter.service
ctf@VM-CTF:/etc/systemd/system$ sudo systemctl enable --now node-exporter.service
Created symlink /etc/systemd/system/multi-user.target.wants/node-exporter.service → /etc/systemd/system/node-exporter.service.
ctf@VM-CTF:/etc/systemd/system$
```

Gambar 4. 84 Jalankan *Node Exporter* Secara Permanen di Masing-masing VM

Instalasi *Node Exporter* juga perlu dilakukan pada VM-1, VM-2, dan VM-3. Langkahnya sama seperti sebelumnya, dari mulai Gambar 4.35 hingga Gambar 4.39. Dengan demikian, *Node Exporter* siap mengekspos metrik yang dibutuhkan oleh Prometheus untuk keperluan monitoring *resource* atau beban kerja pada setiap VM, seperti CPU, RAM, penyimpanan, *bandwidth* jaringan, dan lain sebagainya.

Pastikan *port Node Exporter* dapat diakses dari internet (melalui HTTP). Sebagai contoh, berikut adalah gambar metrik yang diekspos oleh *Node Exporter* pada server pusat (VM-CTF). *Endpoint* metrik yang diekspos oleh Node Exporter ada di *"/metrics"*.

Node Exporter yang berhasil mengekspos data metrik tertera pada Gambar 4.85.



Gambar 4. 85 *Node Exporter* Berhasil Mengekspos Data Metrik Melalui *Port* 9100

4.3.4.3 Instalasi Prometheus

Prometheus merupakan alat gratis yang dapat digunakan untuk monitoring *resource* pada *node server*. Agen *Node Exporter* yang telah terpasang di setiap VM akan mengekspos metrik sumber daya pada sistem operasi, seperti CPU, RAM, bandwidth jaringan, dan lain sebagainya. Prometheus secara berkala akan menarik metrik yang diekspos oleh *Node Exporter* selama 15 detik sekali untuk kebutuhan monitoring. Prometheus akan menjadi *data source* bagi Grafana agar dapat divisualisasikan.

Sebelum instalasi Prometheus, diperlukan pembuatan user dan grup baru bernama *"prometheus"*. Perintah untuk membuat user dan grup baru untuk Prometheus tertera pada Gambar 4.86.

```
ctf@VM-CTF:~$ sudo groupadd --system prometheus
ctf@VM-CTF:~$ sudo useradd --system --no-create-home --shell /sbin/nologin -g prometheus prometheus
ctf@VM-CTF:~$ sudo cat /etc/passwd | grep "prometheus"
prometheus:x:996:988::/home/prometheus:/sbin/nologin
ctf@VM-CTF:~$
```

Gambar 4. 86 Buat *User* dan *Group* Baru Untuk Operasional Prometheus

Kemudian, Unduh *file binary* Prometheus terbaru menggunakan perintah *"wget"* yang tertera pada Gambar 4.87.

```
ctf@VM-CTF:~$ cd /tmp/
ctf@VM-CTF:/tmp$ wget -q https://github.com/prometheus/prometheus/releases/download/v3.13.1/prometheus-3.13.1.linux-amd64.tar.gz
ctf@VM-CTF:/tmp$ ls -lah prometheus-3.13.1.linux-amd64.tar.gz
-rw-rw-r-- 1 ctf ctf 100M Jul 10 15:47 prometheus-3.13.1.linux-amd64.tar.gz
ctf@VM-CTF:/tmp$
```

Gambar 4. 87 Unduh *File Binary* Prometheus

Ekstrak *file* unduhan Prometheus dalam format *".tar.gz"* menggunakan alat *"tar"*. Perintah untuk ekstrak file unduhan Prometheus menggunakan perintah yang ditunjukkan pada Gambar 4.88.

```
ctf@VM-CTF:/tmp$ tar -xzf prometheus-3.13.1.linux-amd64.tar.gz
ctf@VM-CTF:/tmp$ cd prometheus-3.13.1.linux-amd64/
ctf@VM-CTF:/tmp/prometheus-3.13.1.linux-amd64$ ls -lah
total 237M
drwxr-xr-x  2 ctf  ctf  4.0K Jul 10 15:42 .
drwxrwxrwt 14 root root 4.0K Jul 15 06:01 ..
-rw-r--r--  1 ctf  ctf   12K Jul 10 15:42 LICENSE
-rw-r--r--  1 ctf  ctf  4.3K Jul 10 15:42 NOTICE
-rwxr-xr-x  1 ctf  ctf  130M Jul 10 15:42 prometheus
-rw-r--r--  1 ctf  ctf  1.1K Jul 10 15:42 prometheus.yml
-rwxr-xr-x  1 ctf  ctf  108M Jul 10 15:42 promtool
ctf@VM-CTF:/tmp/prometheus-3.13.1.linux-amd64$
```

Gambar 4. 88 Ekstrak *File* Unduhan Prometheus

Pindahkan *file binary "prometheus"* dan *"promtool"* hasil dari ektraksi sebelumnya ke direktori *"/usr/local/bin/"*. Perintah untuk memindahkannya tertera pada Gambar 4.89.

```
ctf@VM-CTF:/tmp/prometheus-3.13.1.linux-amd64$ sudo mv prometheus promtool /usr/local/bin/
ctf@VM-CTF:/tmp/prometheus-3.13.1.linux-amd64$ ls -lah /usr/local/bin/ | grep -E "prometheus|promtool"
-rwxr-xr-x  1 ctf  ctf  130M Jul 10 15:42 prometheus
-rwxr-xr-x  1 ctf  ctf  108M Jul 10 15:42 promtool
ctf@VM-CTF:/tmp/prometheus-3.13.1.linux-amd64$
```

Gambar 4. 89 Pindahkan *File Binary Prometheus* ke Direktori Linux *Binaries*

Buat direktori baru bernama *"prometheus"* di dalam direktori *"/etc/"* dan *"/var/lib"*. Direktori *"/etc/prometheus"* nantinya akan menyimpan *file* konfigurasi Prometheus. Sedangkan, direktori *"/var/lib/prometheus"* difungsikan untuk menyimpan metrik atau data hasil pengumpulan *resource* pada setiap VM.

Perintah untuk membuat direktori baru tertera pada Gambar 4.90.

```
ctf@VM-CTF:/tmp/prometheus-3.13.1.linux-amd64$ sudo mkdir -pm755 /etc/prometheus /var/lib/prometheus
ctf@VM-CTF:/tmp/prometheus-3.13.1.linux-amd64$ ls -lah /etc/prometheus/ /var/lib/prometheus/
/etc/prometheus/:
total 16K
drwxr-xr-x   2 root root 4.0K Jul 15 06:09 .
drwxr-xr-x 127 root root  12K Jul 15 06:09 ..

/var/lib/prometheus/:
total 8.0K
drwxr-xr-x  2 root root 4.0K Jul 15 06:09 .
drwxr-xr-x 53 root root 4.0K Jul 15 06:09 ..
```

Gambar 4. 90 Buat Direktori Baru Untuk Prometheus

Kepemilikan folder *"/var/lib/prometheus"* yang dibuat sebelumnya perlu diubah menjadi user dan grup "prometheus" yang sudah dibuat. Perintah untuk mengubah kepemilikan direktori *"/var/lib/prometheus"* tertera pada Gambar 4.91.

```
ctf@VM-CTF:/tmp/prometheus-3.13.1.linux-amd64$ sudo chown -R prometheus:prometheus /var/lib/prometheus/
ctf@VM-CTF:/tmp/prometheus-3.13.1.linux-amd64$ ls -lah /var/lib/prometheus/
total 8.0K
drwxr-xr-x  2 prometheus prometheus 4.0K Jul 15 06:09 .
drwxr-xr-x 54 root       root       4.0K Jul 15 07:05 ..
ctf@VM-CTF:/tmp/prometheus-3.13.1.linux-amd64$
```

Gambar 4. 91 Ubah Kepemilikan Direktori Penyimpanan Data Metrik Prometheus

Pindahkan *file* konfigurasi *"prometheus.yml"* ke direktori *"/etc/prometheus/"* yang telah dibuat. Kemudian, edit *file* tersebut sesuai dengan kebutuhan. Pada kasus ini, durasi *scrape* atau penarikan data metrik diubah dari 15 detik menjadi 10 detik. Tujuannya adalah untuk mempercepat penarikan data metrik oleh Prometheus. Konfigurasi Prometheus yang telah dimodifikasi tertera pada Gambar 4.92.

```
global:
  scrape_interval: 10s
  evaluation_interval: 10s

scrape_configs:
  # Job 1: Prometheus memantau dirinya sendiri
  - job_name: "prometheus"
    static_configs:
      - targets: ["localhost:9099"]
        labels:
          app: "CTF Platform"

  # Job 2: Node Exporter di semua VM (sumber daya sistem)
  - job_name: "node"
    static_configs:
      - targets: ["localhost:9100"]
        labels:
          vm: "VM-CTF"
          role: "platform"
      - targets: ["43.133.128.127:9100"]
        labels:
          vm: "VM-1"
          role: "no-protection"
      - targets: ["43.157.243.92:9100"]
        labels:
          vm: "VM-2"
          role: "apparmor"
      - targets: ["149.129.193.12:9100"]
        labels:
          vm: "VM-3"
          role: "apparmor-fail2ban"
```

Gambar 4. 92 Edit *File* Konfigurasi Prometheus

Selanjutnya, buat file *"prometheus.service"* di dalam direktori *"/etc/systemd/system/"* untuk menjalankan layanan Prometheus di belakang layar dan mengaktifkan Prometheus secara permanen. *File "prometheus.service"* ditunjukkan pada Gambar 4.93.

```
[Unit]
Description=Prometheus
Documentation=https://prometheus.io/docs/introduction/overview/
Wants=network-online.target
After=network-online.target

[Service]
User=prometheus
Group=prometheus
Type=simple
ExecStart=/usr/local/bin/prometheus \
        --config.file=/etc/prometheus/prometheus.yml \
        --storage.tsdb.path=/var/lib/prometheus/ \
        --storage.tsdb.retention.time=30d \
        --web.listen-address=:9099

[Install]
WantedBy=multi-user.target
```

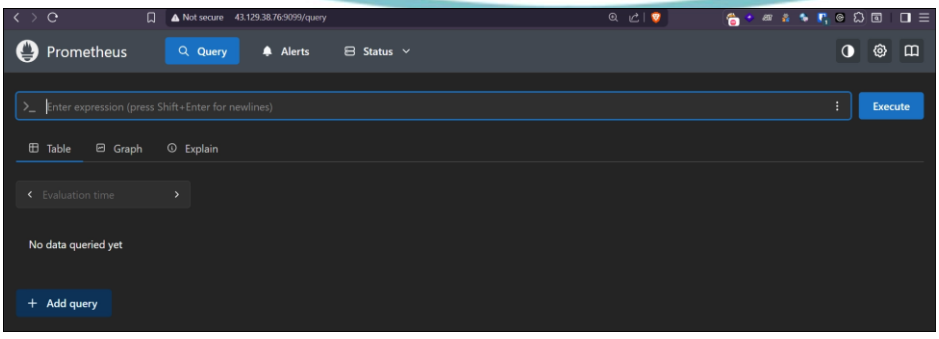
Gambar 4. 93 Buat *File Service* Prometheus

Jalankan *service* Prometheus secara permanen menggunakan perintah yang tertera pada Gambar 4.94.

```
ctf@VM-CTF:/etc/systemd/system$ sudo vim prometheus.service
ctf@VM-CTF:/etc/systemd/system$ sudo systemctl daemon-reload
ctf@VM-CTF:/etc/systemd/system$ sudo systemctl enable --now prometheus.service
Created symlink /etc/systemd/system/multi-user.target.wants/prometheus.service → /etc/systemd/system/prometheus.service.
ctf@VM-CTF:/etc/systemd/system$ sudo systemctl is-enabled prometheus.service
enabled
ctf@VM-CTF:/etc/systemd/system$
```

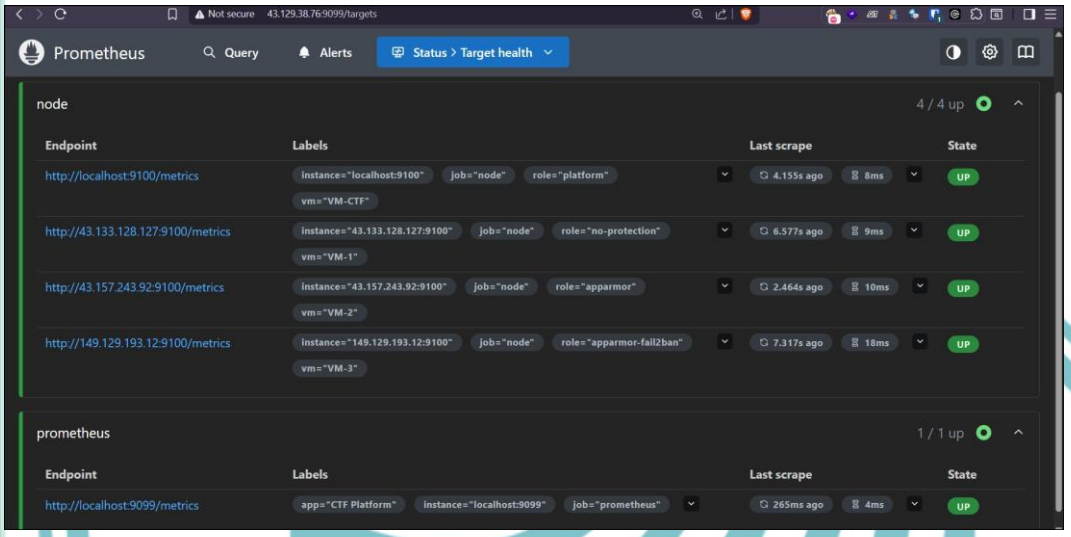
Gambar 4. 94 Jalankan Prometheus Secara Permanen

Pastikan jika halaman dasbor Prometheus dapat diakses melalui protokol HTTP pada *port* 9099 yang sudah didefinisikan pada *file* konfigurasi *"prometheus.yml"*. Gambar 4.95 menunjukkan jika dasbor Prometheus dapat diakses pada *port* yang telah ditentukan, yaitu *port* 9099.



Gambar 4. 95 Pastikan Prometheus Berjalan di *Port* 9099

Selain itu, pastikan status *Node Exporter* di setiap VM dan status Prometheus berstatus *"up"* yang ditandai dengan warna hijau. Status *Node Exporter* di setiap VM dan status Prometheus yang berhasil berjalan tertera pada Gambar 4.51.



Gambar 4. 96 Status *Node Exporter* dan Prometheus yang Berhasil Berjalan

4.3.4.4 Instalasi Grafana Alloy

Identik dengan *Node Exporter*, Grafana Alloy bertugas sebagai agen yang mengirimkan data *log* yang berupa teks ke *endpoint* API milik Grafana Loki. Instalasi Grafana Alloy perlu dilakukan di semua VM. Tahap instalasinya tidak serumit *Node Exporter*, di mana Grafana Alloy sudah tersimpan di repositori *apt* Grafana, sehingga instalasinya sangat mudah. Di awal, tambahkan repositori *apt* Grafana terlebih dahulu menggunakan perintah yang tertera pada Gambar 4.52.

```
ctf@VM-CTF: $ sudo apt install -y gpg wget
sudo mkdir -p /etc/apt/keyrings/
wget -q -O - https://apt.grafana.com/gpg.key | gpg --dearmor | sudo tee /etc/apt/keyrings/grafana.gpg > /dev/null
echo "deb [signed-by=/etc/apt/keyrings/grafana.gpg] https://apt.grafana.com stable main" | sudo tee /etc/apt/sources.list.d/grafana.list
sudo apt-get update
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
gpg is already the newest version (2.4.4-2ubuntu17.4).
gpg set to manually installed.
wget is already the newest version (1.21.4-1ubuntu4.1).
The following packages were automatically installed and are no longer required:
  libfwupd2 libgusb2
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
deb [signed-by=/etc/apt/keyrings/grafana.gpg] https://apt.grafana.com stable main
Hit:1 http://mirrors.tencentyun.com/ubuntu noble InRelease
Hit:2 http://mirrors.tencentyun.com/ubuntu noble-updates InRelease
Hit:3 http://mirrors.tencentyun.com/ubuntu noble-backports InRelease
Hit:4 http://mirrors.tencentyun.com/ubuntu noble-security InRelease
Get:5 https://apt.grafana.com stable InRelease [7,661 B]
Fetched 7,661 B in 0s (15.6 kB/s)
Reading package lists... Done
```

Gambar 4. 97 Menambahkan Repositori *apt* Grafana

Setelah repositori *apt* Grafana ditambahkan, selanjutnya bisa memasang Grafana Alloy menggunakan perintah yang tertera pada Gambar 4.53.

```
ctf@VM-CTF: $ sudo apt-get update && sudo apt-get install -y alloy
Hit:1 http://mirrors.tencentyun.com/ubuntu noble InRelease
Hit:2 http://mirrors.tencentyun.com/ubuntu noble-updates InRelease
Hit:3 http://mirrors.tencentyun.com/ubuntu noble-backports InRelease
Hit:4 http://mirrors.tencentyun.com/ubuntu noble-security InRelease
Hit:5 https://apt.grafana.com stable InRelease
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

Gambar 4. 98 Instalasi Grafana Alloy

Tambahkan *secondary* atau *supplementary group* pada *user "alloy"*, yang menjalankan fungsionalitas Grafana Alloy dengan grup *"adm"* dan *"systemd-journal"*. Tujuannya adalah supaya *user "alloy"* mendapatkan akses untuk membaca *file-file log* lain, seperti *"/var/log/syslog"*. Perintah untuk menambahkan *secondary group* pada *user "alloy"* tertera pada Gambar 4.99.

```
ctf@VM-CTF:~$ sudo usermod -aG adm alloy
ctf@VM-CTF:~$ sudo usermod -aG systemd-journal alloy
ctf@VM-CTF:~$
```

Gambar 4. 99 Menambahkan *Secondary Group* pada *User alloy*

Edit *file* konfigurasi Grafana Alloy yang berlokasi di *"/etc/default/alloy"* untuk mengizinkan akses halaman dasbor Grafana Alloy dari internet/eksternal pada *port* bawaannya, yaitu *port* 12345. Bagian yang diedit pada *file* tersebut tertera pada Gambar 4.100.

```
12 ## Path:
11 ## Description: Grafana Alloy settings
10 ## Type:        string
 9 ## Default:     ""
 8 ## ServiceRestart: alloy
 7 #
 6 # Command line options for Alloy.
 5 #
 4 # The configuration file holding the Alloy config.
 3 CONFIG_FILE="/etc/alloy/config.alloy"
 2
 1 # User-defined arguments to pass to the run command.
 0 CUSTOM_ARGS="--server.http.listen-addr=0.0.0.0:12345"
 1
 2 # Restart on system upgrade. Defaults to true.
 3 RESTART_ON_UPGRADE=true
```

Gambar 4. 100 Izinkan Halaman Dasbor Grafana Alloy Dapat Diakses Dari Internet

Terdapat tahap konfigurasi tambahan di masing-masing VM, yaitu mengedit file konfigurasi Grafana Alloy yang mendefinisikan daftar *file log* yang akan dikirim ke Grafana Loki. *File* konfigurasi tersebut berlokasi di *"/etc/alloy/config.alloy"*.

Di VM-1, konfigurasi *"/etc/alloy/config.alloy"* seperti yang tertera pada Gambar 4.101. Di mana, *api endpoint* Grafana Loki mengarah ke alamat IP server pusat.

```
// ═══ VM-1 (tanpa proteksi) ═══
// Writer: kirim ke Loki di VM-CTF
loki.write "vmctf" {
  endpoint {
    url = "http://43.129.38.76:3100/loki/api/v1/push"   // IP VM-CTF
  }
}

// Sumber: log challenge (crash & connection)
local.file_match "labs" {
  path_targets = [{ __path__ = "/var/log/labs/*.log", vm = "VM-1", role = "no-protection" }]
  sync_period  = "10s"
}
loki.source.file "labs" {
  targets    = local.file_match.labs.targets
  forward_to = [loki.write.vmctf.receiver]
}
```

Gambar 4. 101 Daftar *File Log* di VM-1 yang Dikirimkan ke Grafana Loki (Server Pusat)

Mirip seperti di VM-1, di VM-2 ada tambahan *file log "/var/log/syslog"* yang menyimpan riwayat pelanggaran yang berhasil ditolak oleh AppArmor. *File log* tersebut juga dikirim ke Grafana Loki. Detail *file* konfigurasi Grafana Alloy di VM-2 tertera pada Gambar 4.102.

```
// ═══ VM-2 (AppArmor longgar & ketat) ═══
loki.write "vmctf" {
  endpoint {
    url = "http://43.129.38.76:3100/loki/api/v1/push"
  }
}

// Sumber 1: log challenge
local.file_match "labs" {
  path_targets = [{ __path__ = "/var/log/labs/*.log", vm = "VM-2", role = "apparmor" }]
  sync_period  = "10s"
}
loki.source.file "labs" {
  targets    = local.file_match.labs.targets
  forward_to = [loki.write.vmctf.receiver]
}

// Sumber 2: syslog (deny-event AppArmor)
local.file_match "syslog" {
  path_targets = [{ __path__ = "/var/log/syslog", vm = "VM-2", job = "syslog" }]
  sync_period  = "10s"
}
loki.source.file "syslog" {
  targets    = local.file_match.syslog.targets
  forward_to = [loki.write.vmctf.receiver]
}
```

Gambar 4. 102 Daftar *File Log* di VM-2 yang Dikirim ke Grafana Loki (Server Pusat)

Di VM-3, selain ada tambahan *file log "/var/log/syslog"*, terdapat *file log* Fail2Ban, yang berlokasi di *"/var/log/fail2ban.log"*. Detail dari konfigurasi file konfigurasi Grafana Alloy di VM-3 tertera pada Gambar 4.103.

```
loki.write "vmctf" {
  endpoint {
    url = "http://43.129.38.76:3100/loki/api/v1/push"
  }
}

// Sumber 1: log challenge (crash & connection)
local.file_match "labs" {
  path_targets = [{ __path__ = "/var/log/labs/*.log", vm = "VM-3", role = "apparmor-fail2ban" }]
  sync_period  = "10s"
}
loki.source.file "labs" {
  targets    = local.file_match.labs.targets
  forward_to = [loki.write.vmctf.receiver]
}

// Sumber 2: syslog (deny-event AppArmor)
local.file_match "syslog" {
  path_targets = [{ __path__ = "/var/log/syslog", vm = "VM-3", job = "syslog" }]
  sync_period  = "10s"
}
loki.source.file "syslog" {
  targets    = local.file_match.syslog.targets
  forward_to = [loki.write.vmctf.receiver]
}

// Sumber 3: fail2ban.log (block-event)
local.file_match "fail2ban" {
  path_targets = [{ __path__ = "/var/log/fail2ban.log", vm = "VM-3", job = "fail2ban" }]
  sync_period  = "10s"
}
loki.source.file "fail2ban" {
  targets    = local.file_match.fail2ban.targets
  forward_to = [loki.write.vmctf.receiver]
}
```

Gambar 4. 103 Daftar *File Log* di VM-3 yang Dikirim ke Grafana Loki (Server Pusat)

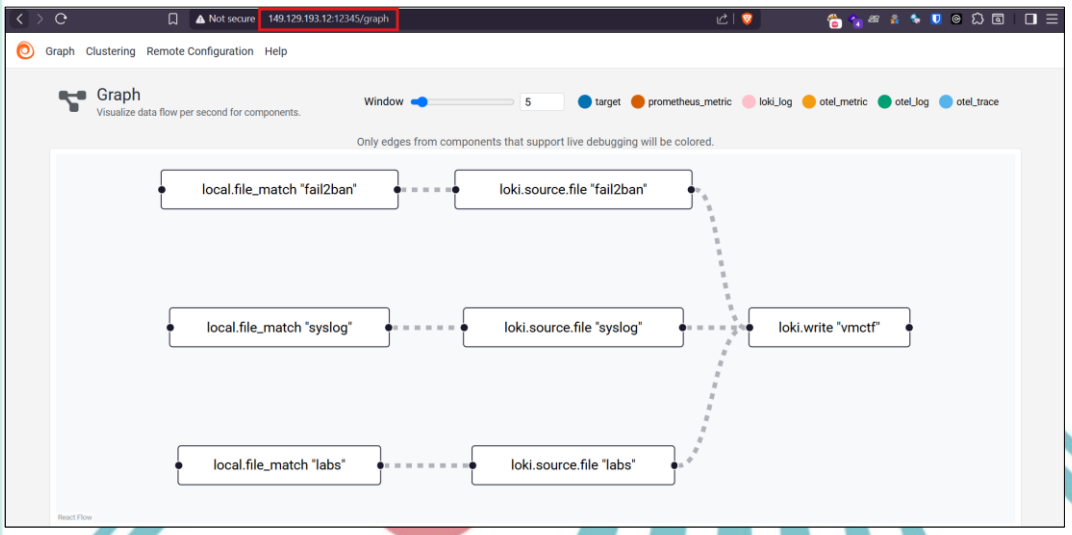
Setelah itu, jalankan layanan Grafana Alloy secara permanen di masing-masing VM menggunakan perintah yang tertera pada Gambar 4.104.

```
ctf@VM-CTF:~$ sudo systemctl restart alloy.service
ctf@VM-CTF:~$ sudo systemctl enable --now alloy.service
ctf@VM-CTF:~$ sudo systemctl is-enabled alloy.service
enabled
ctf@VM-CTF:~$
```

Gambar 4. 104 Jalankan Grafana Alloy Secara Permanen di Masing-masing VM

Pastikan jika halaman dasbor Grafana Alloy dapat diakses melalui internet pada *port* bawaannya, yaitu *port* 12345.

Sebagai contoh, Gambar 4.105 merupakan halaman dasbor Grafana Alloy VM-3.



Gambar 4. 105 Tampilan Dasbor Grafana Alloy

4.3.4.5 Instalasi Grafana Loki

Grafana Loki merupakan alat untuk mengumpulkan dan menyimpan data log berbentuk teks secara efisien dengan mengindeks labelnya saja, sehingga bukan teks log secara keseluruhan. Instalasi Grafana Loki bisa menggunakan repositori apt Grafana, yang sudah ditambahkan pada saat instalasi Grafana sebelumnya. Langsung saja instalasi Grafana Loki di server pusat menggunakan perintah yang tertera pada Gambar 4.106.

```
ctf@VM-CTF:~$ sudo apt-get update && sudo apt-get install -y loki
Hit:1 http://mirrors.tencentyun.com/ubuntu noble InRelease
Hit:2 http://mirrors.tencentyun.com/ubuntu noble-updates InRelease
Hit:3 http://mirrors.tencentyun.com/ubuntu noble-backports InRelease
Hit:4 http://mirrors.tencentyun.com/ubuntu noble-security InRelease
Get:5 https://apt.grafana.com stable InRelease [7,661 B]
Fetched 7,661 B in 1s (14.8 kB/s)
```

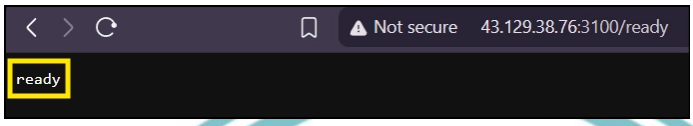
Gambar 4. 106 Instalasi Grafana Loki di Server Pusat

Jalankan Grafana Loki secara permanen di belakang layar menggunakan perintah yang tertera pada Gambar 4.107.

```
ctf@VM-CTF:~$ sudo systemctl enable --now loki.service
ctf@VM-CTF:~$ sudo systemctl is-enabled loki.service
enabled
ctf@VM-CTF:~$
```

Gambar 4. 107 Jalankan Grafana Loki Secara Permanen

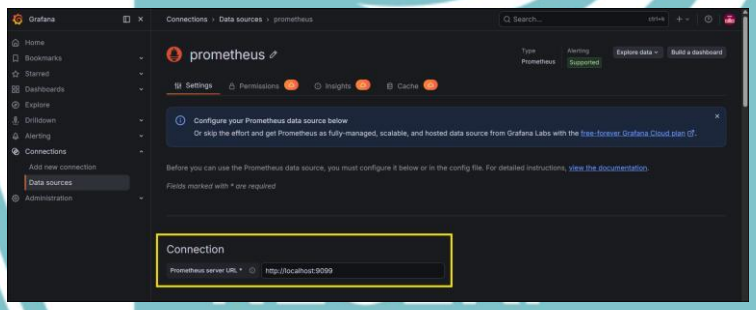
Pastikan Grafana Loki yang telah berjalan di server pusat sudah berhasil berjalan di *port* bawaannya, yaitu 3100. Jika berhasil berjalan, maka statusnya adalah *"ready"*. Status Grafana Loki yang berhasil berjalan di *port* 3100 di server pusat ditunjukkan pada Gambar 4.108.



Gambar 4. 108 Status Grafana Loki yang Sukses Berjalan di Server Pusat

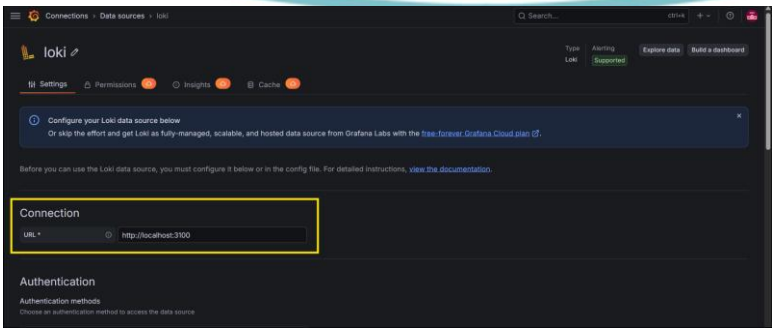
4.3.4.6 Membuat Visualisasi Dasbor di Grafana

Prometheus dan Grafana Loki yang berhasil memperoleh data dari masing-masing VM sudah bisa mulai dimonitoring melalui dasbor Grafana. Namun, koneksi *data source* perlu ditambahkan terlebih dahulu. Pertama, tambahkan *data source* Prometheus yang berjalan di *port* 9099. Alur penambahan *data source* Prometheus di Grafana untuk menarik metrik sumber daya di ketiga VM ditunjukkan pada Gambar 4.109.



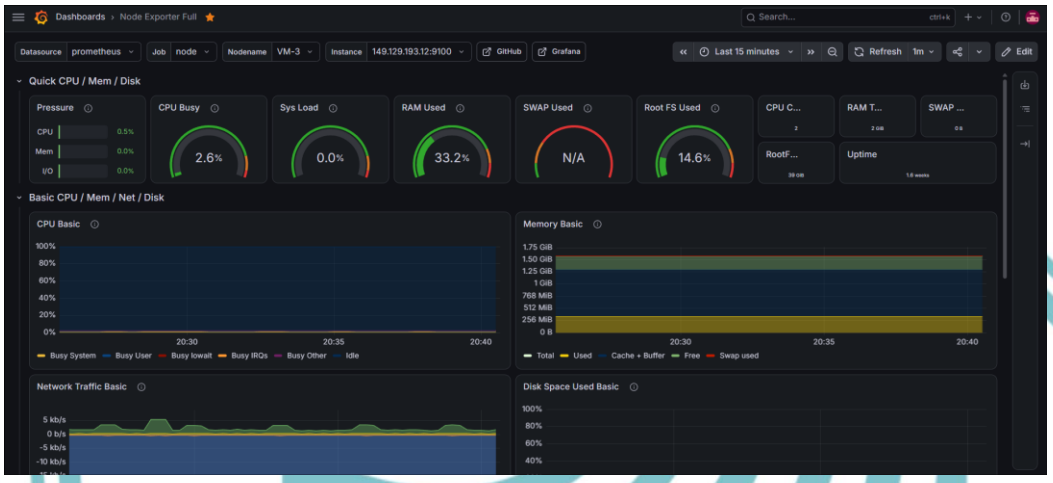
Gambar 4. 109 Menambahkan *Data Source* Prometheus

Kemudian, tambahkan *data source* Grafana Loki yang berjalan di *port* 3100. Penambahan *data source* Grafana Loki untuk menerima data *log* (berupa teks) yang dikirimkan oleh Grafana Alloy ditunjukkan pada Gambar 4.110.



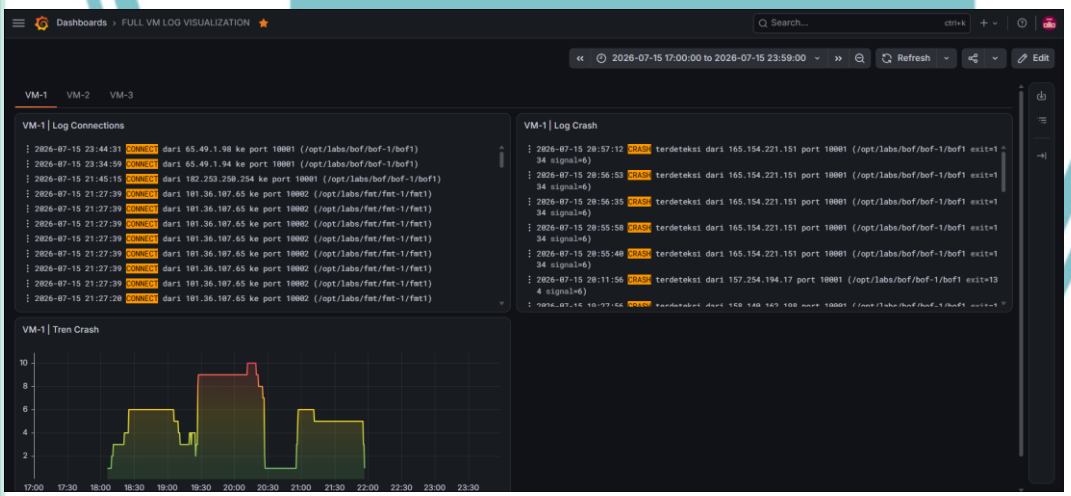
Gambar 4. 110 Menambahkan *Data Source* Grafana Loki

Dengan ditambahkannya *data source* dari Prometheus dan Grafana Loki, maka proses monitoring dan pembuatan dasbor untuk visualisasi data sudah bisa dilakukan. Gambar 4.111 menunjukkan dasbor yang berhasil memonitoring sumber daya pada setiap VM secara *real-time*.



Gambar 4. 111 Visualisasi Sumber Daya Setiap VM Melalui Dasbor Grafana

Selain itu, Gambar 4.112 menunjukkan bagian-bagian data *log* yang akan diaudit selama proses eksploitasi atau durasi ajang CTF berlangsung. Data *log* tersebut diambil dari *data source* Grafana Loki.

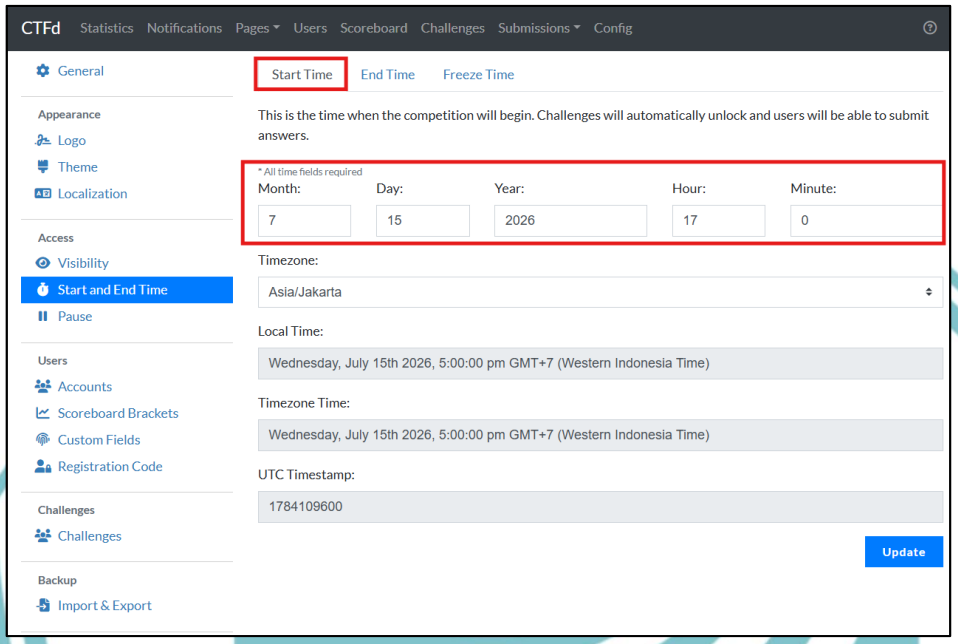


Gambar 4. 112 Kumpulan Data Log yang Akan Diaudit Melalui Dasbor Grafana

4.3.5 Persiapan Platform CTFd

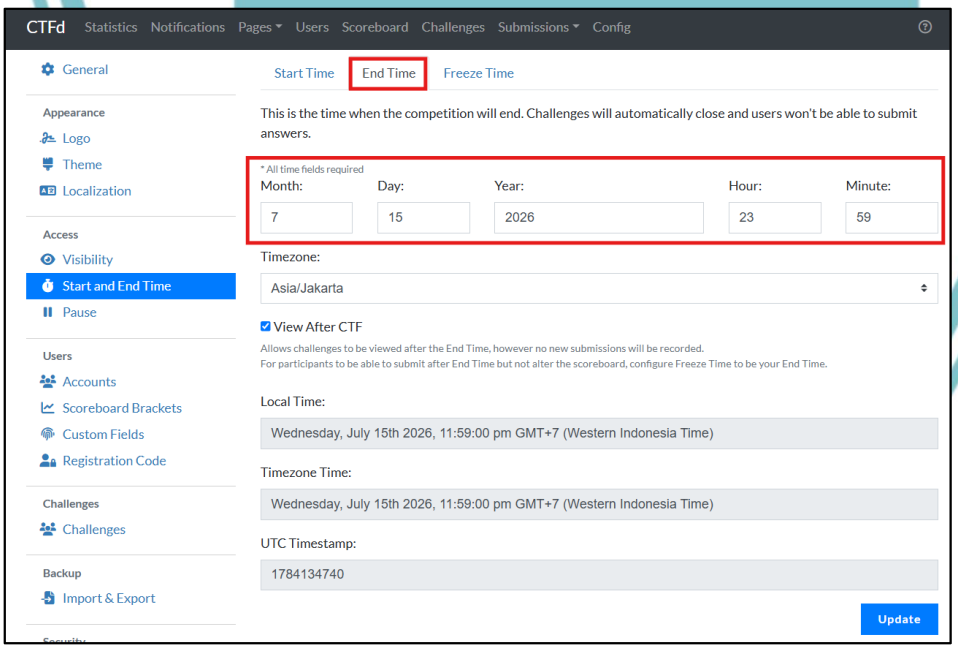
Platform CTFd yang sudah terpasang di server pusat (VM-CTF) perlu diatur durasi waktu mulai dan berakhirnya ajang CTF. Di mana, ajang CTF pada penelitian ini diatur berlangsung pada tanggal 15 Juli 2026 mulai dari pukul 17:00 WIB sampai

pukul 23:59 WIB (selama 7 jam). Durasi waktu mulai dan berakhirnya ajang CTF dapat diatur pada platform CTFd melalui panel admin. Gambar 4.113 menunjukkan waktu mulai ajang CTF.



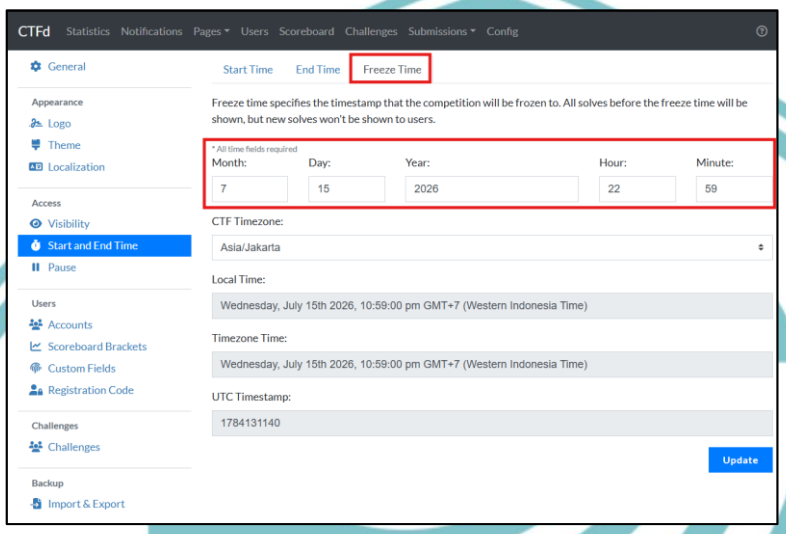
Gambar 4. 113 Pengaturan Waktu Mulainya CTF

Waktu berakhirnya ajang CTF, yaitu 23:59 WIB ditunjukkan pada Gambar 4.114.



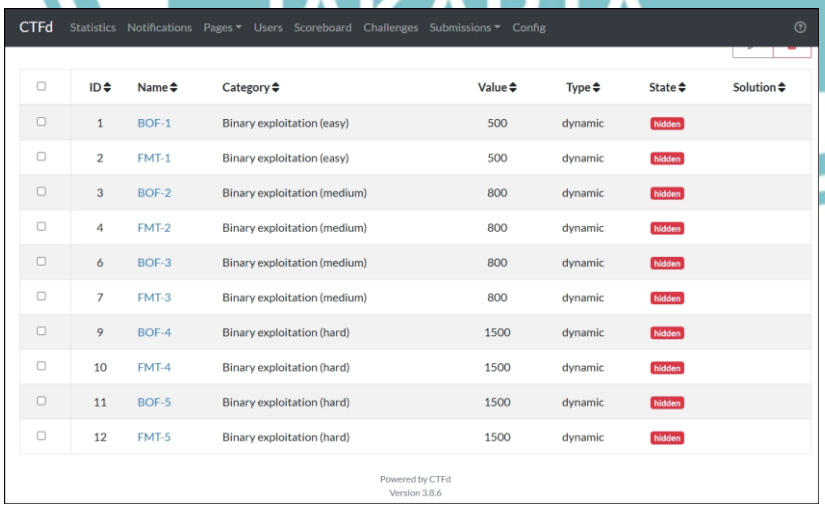
Gambar 4. 114 Pengaturan Waktu Berakhirnya CTF

Terdapat waktu *freeze scoreboard*, yaitu 1 jam sebelum ajang berakhir, di mana *freeze time* di atur pada pukul 22:59 WIB. *Freeze time* masih mengizinkan penyerang untuk mengirimkan *flag* untuk mendapatkan poin tambahan, tetapi grafik pada *scoreboard* sudah tidak dapat berubah lagi. Pengaturan waktu *freeze* pada platform CTFd ditunjukkan pada Gambar 4.115.



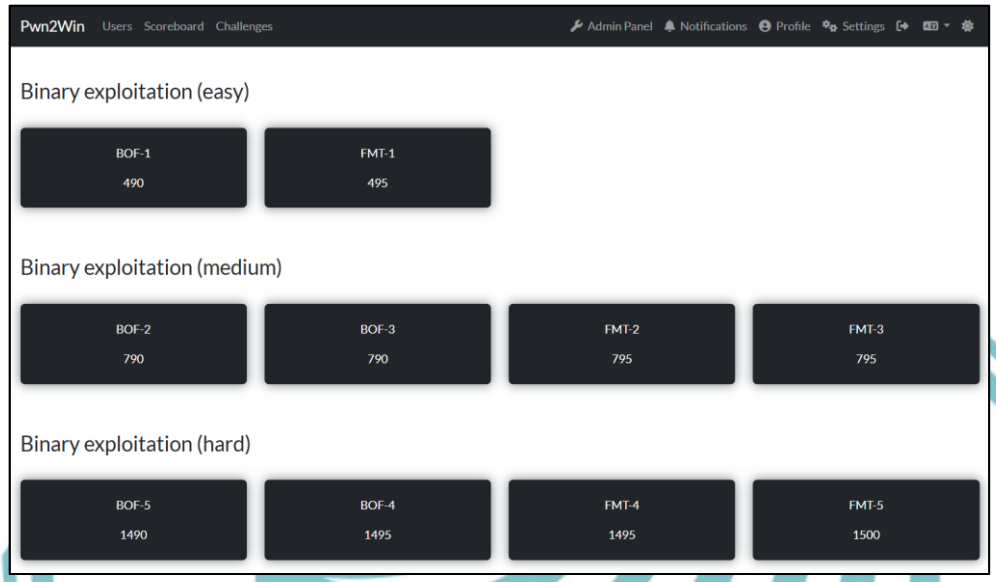
Gambar 4. 115 Pengaturan Waktu *Freeze* CTF

Kesepuluh *file ELF binary* yang sudah dikompilasi pada bagian 4.3.2 kemudian diunggah ke platform CTF agar dapat diunduh dan dieksploitasi oleh penyerang. Selain itu, berikan akses ke layanan *binary* yang bisa diakses melalui jaringan. Gambar 4.116 menunjukkan kesepuluh *file binary* yang telah diunggah ke platform CTFd.



Gambar 4. 116 Menambahkan *File Binary* dan Akses *Remote Binary* di Platform CTFd

Daftar *file binary* yang telah diunggah ke platform CTFd tertera pada Gambar 4.117.



Gambar 4. 117 Daftar *File Binary* yang Disiapkan Untuk Dieksploitasi

4.4 Pengujian

Pengujian dilakukan untuk memverifikasi bahwa sistem keamanan yang telah dirancang dan diimplementasikan berjalan sesuai rancangan, sekaligus mengevaluasi sejauh mana AppArmor dan Fail2Ban membatasi dampak serta memblokir upaya eksploitasi *buffer overflow* dan *format string*. Pengujian disusun ke dalam lima skenario yang dijalankan secara berurutan, yaitu pengujian fungsionalitas, pembatasan dampak oleh AppArmor, pemblokiran alamat IP oleh Fail2Ban, pembanjiran koneksi (*TCP connection flooding*), dan spam *input* perusak. Pengujian fungsionalitas dilakukan pada seluruh mesin (VM-CTF, VM-1, VM-2, dan VM-3), pengujian pembatasan dampak oleh AppArmor dilakukan pada VM-2 dan VM-3, sedangkan pengujian pemblokiran alamat IP, pembanjiran koneksi, dan spam *input* perusak difokuskan pada VM-3 yang memuat kombinasi AppArmor dan Fail2Ban. Setiap skenario diuraikan ke dalam empat bagian, yaitu deskripsi pengujian, prosedur pengujian, data hasil pengujian, serta analisis data dan evaluasi. Seluruh data hasil pengujian bersumber dari *file log* dan metrik yang dikumpulkan melalui *Node Exporter* dan Grafana Alloy, kemudian divisualisasikan pada dasbor Grafana.

4.4.1 Deskripsi Pengujian

4.4.1.1 Pengujian Fungsionalitas Sistem

Pengujian fungsionalitas dilakukan pada seluruh mesin virtual (VM-CTF, VM-1, VM-2, dan VM-3) dan bertujuan memastikan seluruh komponen sistem telah berjalan dan saling terhubung sebelum pengujian inti dilaksanakan. Pengujian ini tidak menilai ketahanan sistem terhadap serangan, melainkan memverifikasi bahwa setiap layanan berada dalam kondisi aktif, dapat diakses, dan tetap berjalan setelah *reboot*. Komponen yang diperiksa mencakup profil dan layanan AppArmor pada VM-2 dan VM-3, layanan serta konfigurasi Fail2Ban pada VM-3, layanan platform CTFd pada server pusat (VM-CTF), tumpukan pemantauan (Grafana, Prometheus, dan Loki) pada VM-CTF, agen *Node Exporter* dan Grafana Alloy pada VM-1 sampai VM-3, keterjangkauan seluruh *binary* melalui jaringan, serta aksesibilitas platform CTFd dari sudut pandang penyerang.

4.4.1.2 Pengujian Pembatasan Dampak Eksploitasi oleh AppArmor

Pengujian ini dilakukan pada VM-2 dan VM-3, serta bertujuan membuktikan bahwa AppArmor membatasi dampak lanjutan dari eksploitasi yang telah berhasil, bukan menambal kerentanan *binary* itu sendiri. Pengujian membandingkan dua konfigurasi profil, yaitu profil longgar (dipasang pada BOF-2/FMT-2 di VM-2 dan BOF-4/FMT-4 di VM-3) dan profil ketat (dipasang pada BOF-3/FMT-3 di VM-2 dan BOF-5/FMT-5 di VM-3). Pada profil longgar, penyerang yang berhasil mengeksploitasi *binary* masih dapat memperoleh *shell* dan membaca *file* sensitif, sehingga dampak eksploitasi meluas. Pada profil ketat, eksploitasi tetap berjalan, tetapi upaya *"exec /bin/sh"* serta pembacaan *file* seperti *"/etc/shadow"* ditolak AppArmor. Pengujian ini menegaskan peran AppArmor sebagai lapisan pencegah di tingkat sistem operasi.

4.4.1.3 Pengujian Pemblokiran Alamat IP oleh Fail2Ban

Pengujian ini dilakukan khusus pada VM-3 dan bertujuan mengevaluasi kemampuan Fail2Ban dalam memblokir alamat IP penyerang berdasarkan bukti *crash* berulang. Fail2Ban bekerja pada lapisan jaringan dan memblokir alamat IP ketika jumlah *crash* dari satu sumber mencapai ambang batas (*maxretry*), yaitu 3 kali dalam kurun sepuluh menit. Selain itu, Fail2Ban juga mengatur ambang batas

koneksi pengguna, sehingga upaya pembanjiran koneksi lebih dari 50 kali dalam kurun waktu 1 menit akan diblokir.

Berbeda dengan AppArmor yang menolak operasi di tingkat sistem operasi, Fail2Ban menghentikan serangan berulang di tingkat jaringan. Pengujian juga mencatat waktu awal *crash*, waktu blokir, serta selisih keduanya (*blocking time*) untuk menilai responsivitas mekanisme.

4.4.1.4 Pengujian Pembanjiran Koneksi (*TCP Connection Flooding*)

Pengujian ini dilakukan khusus pada VM-3 dan bertujuan mengukur beban server serta memverifikasi pemblokiran alamat IP penyerang yang membuka banyak koneksi dalam waktu singkat. Ambang *flood-detection* ditetapkan sebesar 50 koneksi dari satu sumber alamat IP yang sama dalam kurun 1 menit. Skenario ini merepresentasikan indikasi kuat upaya membuat layanan tidak dapat diakses (*Denial of Service*). Jenis pembanjiran yang diuji adalah *TCP connection flooding*, yaitu membanjiri layanan dengan koneksi TCP penuh yang terbentuk sempurna dari satu sumber, bukan *SYN flooding* yang mengeksploitasi koneksi setengah-terbuka. Pemilihan ini sesuai dengan mekanisme deteksi Fail2Ban yang menghitung jumlah koneksi pada *file "/var/log/labs/conn.log"*, bukan menganalisis *flag* pada segmen TCP. Beban server diamati melalui metrik CPU, memori, dan lalu lintas jaringan yang dipantau Node Exporter dan Prometheus.

Pengujian pembanjiran koneksi ini dilaksanakan pada sesi terpisah dari empat pengujian lainnya. Pemisahan ini dilakukan atas dasar pertimbangan bahwa pada skenario utama penyerang difokuskan untuk mengeksploitasi kerentanan *binary*, sedangkan pembanjiran koneksi merepresentasikan pola serangan volumetrik yang berbeda karakteristiknya, sehingga pengujiannya dipisahkan agar beban server yang terukur benar-benar berasal dari aktivitas pembanjiran dan tidak bercampur dengan lalu lintas eksploitasi lain.

4.4.2 Prosedur Pengujian

4.4.2.1 Prosedur Pengujian Fungsionalitas Sistem

Prosedur pengujian fungsionalitas dilakukan dengan memeriksa status tiap komponen pada seluruh mesin virtual menggunakan perintah verifikasi. Detail prosedur pengujian fungsionalitas sistem ditunjukkan pada Tabel 4.5.

Tabel 4. 5 Prosedur Pengujian Fungsionalitas Sistem

No	Skenario Pengujian	Hasil yang Diharapkan
1.	Memeriksa profil dan layanan AppArmor di VM-2 dan VM-3	Layanan AppArmor aktif dan seluruh profil *binary* ter-*enforce*
2.	Memeriksa layanan dan konfigurasi Fail2Ban di VM-3	Layanan Fail2Ban aktif dan *jail crash-detection* serta *flood-detection* berstatus aktif
3.	Memeriksa layanan CTFd dan *reverse proxy* Nginx (HTTPS) di VM-CTF	Platform CTFd dapat diakses via HTTPS tanpa galat sertifikat
4.	Memeriksa layanan Grafana, Prometheus, dan Loki di VM-CTF	Ketiga layanan pemantauan aktif dan dasbor Grafana menampilkan data
5.	Memeriksa agen Node Exporter dan Grafana Alloy di VM-1 s.d. VM-3	Metrik (*pull* Prometheus) dan log (*push* Loki) diterima server pusat
6.	Memastikan seluruh *binary* (VM-1 s.d. VM-3) dapat diakses via jaringan	*Socat* berstatus *LISTEN* dan penyerang menerima respons awal tiap *binary*
7.	Mengakses platform CTFd via peramban (sudut pandang penyerang)	Sepuluh *file ELF binary* terekspos dan dapat diunduh

8.	Memastikan seluruh layanan tetap aktif setelah *reboot*	Seluruh layanan berstatus *enabled* (*systemctl is-enabled*)

4.4.2.2 Prosedur Pengujian Pembatasan Dampak Eksploitasi oleh AppArmor

Prosedur dilakukan dengan menjalankan eksploitasi terhadap *binary* berprofil longgar dan ketat di VM-2 dan VM-3 dengan objektif yang sama, kemudian mencatat setiap *deny-event* (*apparmor="DENIED"*). Detail prosedur pengujian pembatasan dampak eksploitasi oleh AppArmor ditunjukkan pada Tabel 4.6.

Tabel 4. 6 Prosedur Pengujian Pembatasan Dampak Eksploitasi oleh AppArmor

No	Skenario Pengujian	Hasil yang Diharapkan
1.	Eksploitasi *binary* berprofil longgar (BOF-2/FMT-2 di VM-2, BOF-4/FMT-4 di VM-3)	Penyerang memperoleh *shell* dan membaca *file* sistem sensitif (dampak meluas)
2.	Eksploitasi *binary* berprofil ketat (BOF-3/FMT-3 di VM-2, BOF-5/FMT-5 di VM-3)	Eksploitasi berjalan, tetapi *exec /bin/sh* dan pembacaan *file* sensitif ditolak
3.	Mencatat peristiwa pencegahan setiap upaya pelanggaran (*deny-event*)	Setiap penolakan tercatat sebagai *apparmor="DENIED"* pada file log sistem di Linux, yaitu *"/var/log/syslog"*
4.	Membandingkan hasil profil longgar vs ketat pada objektif sama	Terlihat kontras dampak: profil longgar tembus, profil ketat terkurung

4.4.2.3 Prosedur Pengujian Pemblokiran Alamat IP oleh Fail2Ban

Prosedur dilakukan pada VM-3 dengan memicu *crash* berulang dari satu alamat IP yang sama hingga mencapai ambang batas, memverifikasi pemblokiran beserta waktu awal *crash* dan waktu blokir, dan memastikan alamat IP yang diblokir tidak

lagi dapat mengakses layanan *binary* melalui jaringan. Detail prosedur pengujian pemblokiran alamat IP oleh Fail2Ban ditunjukkan pada Tabel 4.7.

Tabel 4. 7 Prosedur Pengujian Pemblokiran Alamat IP oleh Fail2Ban

No	Skenario Pengujian	Hasil yang Diharapkan
1.	Memicu *crash* berulang (*exit* 139/134) dari satu IP hingga *maxretry*	Fail2Ban memblokir IP setelah *crash* ke-3 dalam window 10 menit
2.	Memverifikasi *block-event* beserta waktu awal *crash* dan waktu blokir	Tercatat *NOTICE [crash-detection] Ban* pada *file "/var/log/fail2ban.log"*
3.	Menguji akses layanan dari IP yang telah diblokir	Koneksi ditolak (*connection refused/timeout*)
4.	Mencatat selisih waktu awal *crash* dan waktu blokir (*blocking time*)	Selisih waktu terukur untuk tiap pemblokiran

4.4.2.4 Prosedur Pengujian Pembanjiran Koneksi (*TCP Connection Flooding*)

Prosedur dilakukan pada VM-3 dengan membanjiri koneksi ke *service binary* dari satu sumber, mengukur beban server melalui *Node Exporter* dan Prometheus, memverifikasi pemblokiran setelah ambang koneksi terlewati, serta mencatat kondisi server sebelum, selama, dan sesudah pemblokiran. Pengujian pembanjiran koneksi dilakukan pada sesi terpisah dari fase eksploitasi *binary* agar beban kerja tidak bercampur dengan proses eksploitasi *binary* yang sedang berlangsung. Detail prosedur pengujian *TCP connection flooding* ditunjukkan pada Tabel 4.8.

Tabel 4. 8 Prosedur Pengujian *TCP Connection Flooding*

No	Skenario Pengujian	Hasil yang Diharapkan

1.	Membanjiri koneksi ke *service binary* dari satu sumber	Tercatat lonjakan koneksi masif dari satu alamat IP
2.	Mengukur beban server (CPU, memori, jaringan) via *Node Exporter*/Prometheus	Kenaikan beban terpantau pada dasbor Grafana
3.	Memverifikasi pemblokiran IP setelah melewati ambang koneksi	Tercatat *NOTICE [flood-detection] Ban* pada *file "/var/log/fail2ban.log"*
4.	Mencatat kondisi server sebelum, selama, dan sesudah pemblokiran	Beban menurun kembali setelah IP pembanjir diblokir

4.4.3 Data Hasil Pengujian

4.4.3.1 Hasil Pengujian Fungsionalitas Sistem

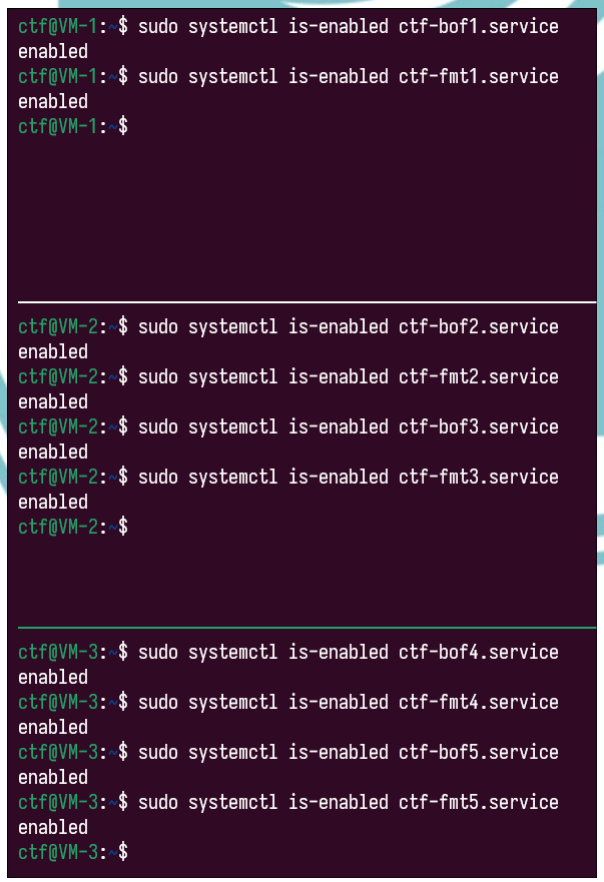
Hasil pemeriksaan seluruh komponen pada ketiga VM (VM-1 s.d VM-3) menunjukkan bahwa sistem berada dalam kondisi siap uji. Rekapitulasi ditunjukkan pada Tabel 4.10.

Tabel 4. 9 Hasil Pengujian Fungsionalitas Sistem

No	Komponen yang Diuji	Hasil Pengujian	Status
1.	*Binary* (VM-1 s.d. VM-3)	Seluruh *binary* berjalan pada port 10001–10010 dengan *socat* berstatus *LISTEN*	Berhasil
2.	AppArmor (VM-2 & VM-3)	Layanan AppArmor aktif dan seluruh profil *binary* telah dimuat ke mode *Enforce*	Berhasil
3.	Fail2Ban (VM-3)	Konfigurasi *Jail "crash-detection"* dan *"flood-detection"* telah aktif di VM-3	Berhasil

4.	CTFd + Nginx HTTPS (VM-CTF)	Dapat diakses via HTTPS; sertifikat SSL valid dan *auto-renew*	Berhasil
5.	Grafana, Prometheus, Loki (VM-CTF)	Ketiga layanan aktif dan dasbor menampilkan metrik & log secara *real-time* pada ketiga VM	Berhasil
6.	*Node Exporter* & Grafana Alloy (VM-1 s.d. VM-3)	Data metrik *resource* dan log ketiga VM terkirim ke server pusat	Berhasil
7.	Persistensi layanan pasca-*reboot*	Seluruh layanan berstatus *enabled* setelah VM dinyalakan ulang	Berhasil

Kesepuluh *binary* yang telah berjalan di VM-1 s.d VM-3 ditunjukkan pada Gambar 4.118.



Gambar 4. 118 Semua *Binary* Telah Berjalan di Ketiga VM

Service AppArmor yang telah berjalan di VM-2 dan VM-3 ditunjukkan pada Gambar 4.119.

```
ctf@VM-2:~$ sudo systemctl is-enabled apparmor.service
enabled
ctf@VM-2:~$ sudo aa-status | grep -E "bof-*|fmt-*"
   /opt/labs/bof/bof-2/bof2
   /opt/labs/bof/bof-3/bof3
   /opt/labs/fmt/fmt-2/fmt2
   /opt/labs/fmt/fmt-3/fmt3
   /opt/labs/fmt/fmt-2/fmt2//null-/usr/bin/dash
   /opt/labs/fmt/fmt-2/fmt2//null-/usr/bin/dash//null-/usr/bin/dash
ctf@VM-2:~$
```

```
ctf@VM-3:~$ sudo systemctl is-enabled apparmor.service
enabled
ctf@VM-3:~$ sudo aa-status | grep -E "bof-*|fmt-*"
   /opt/labs/bof/bof-4/bof4
   /opt/labs/bof/bof-5/bof5
   /opt/labs/fmt/fmt-4/fmt4
   /opt/labs/fmt/fmt-5/fmt5
ctf@VM-3:~$
```

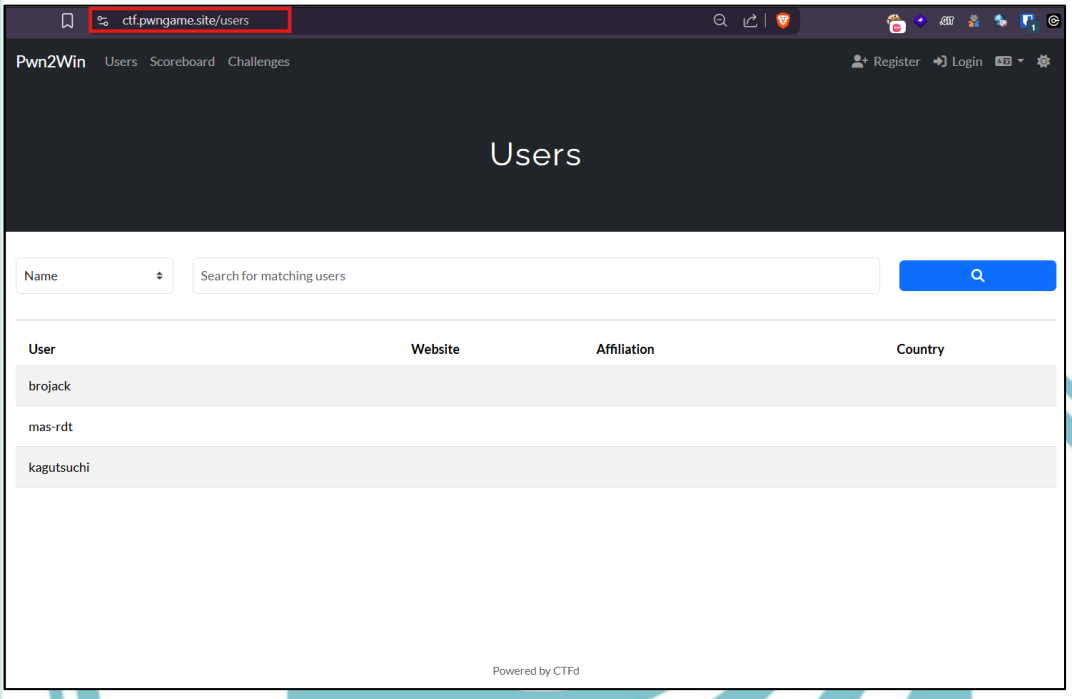
Gambar 4. 119 *Service* AppArmor yang Telah Berjalan di VM-2 dan VM-3

Hasil pemeriksaan status konfigurasi Fail2Ban *"crash-detection"* dan *"flood-detection"* yang juga telah berjalan di VM-3 ditunjukkan pada Gambar 4.120.

```
ctf@VM-3:~$ sudo systemctl is-enabled fail2ban.service
enabled
ctf@VM-3:~$ sudo fail2ban-client status crash-detection
Status for the jail: crash-detection
|- Filter
|  |- Currently failed: 0
|  |- Total failed:     1
|  `- File list:        /var/log/labs/crash.log
`- Actions
   |- Currently banned: 3
   |- Total banned:     3
   `- Banned IP list:   158.140.162.198 180.252.161.153 182.253.250.254
ctf@VM-3:~$
```

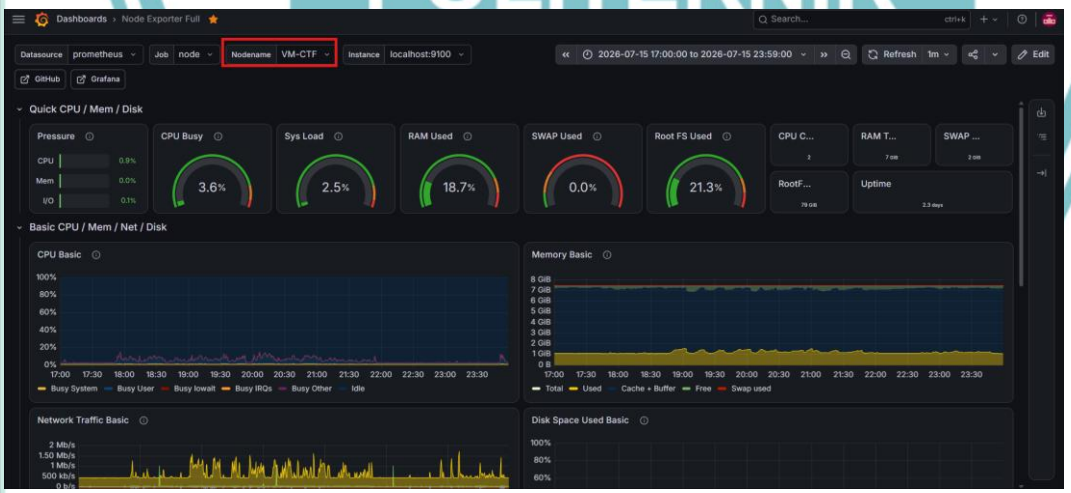
Gambar 4. 120 *Service* Fail2Ban Telah Berjalan di VM-3

Platform web CTFd yang bisa diakses melalui alamat domain "https://ctf.pwngame.site" ditunjukkan pada Gambar 4.121.



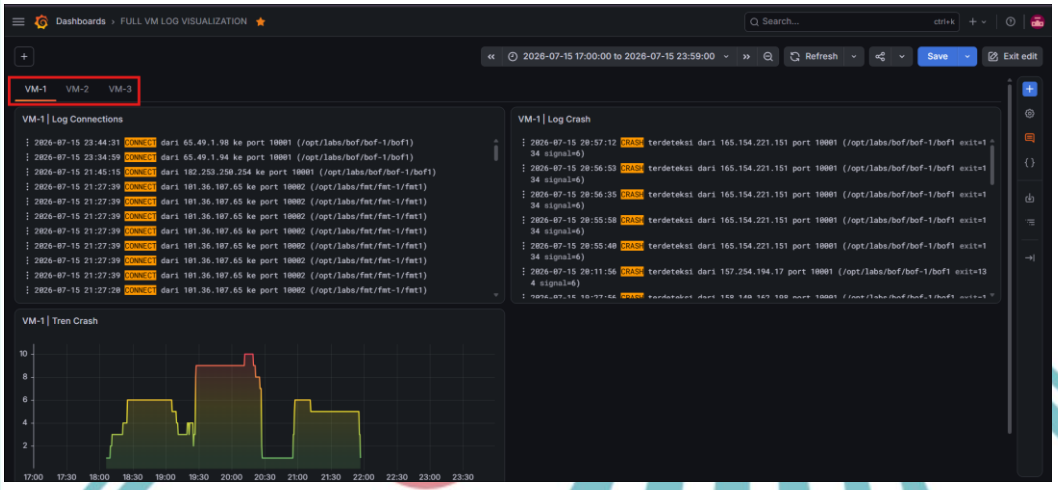
Gambar 4. 121 Tampilan Dasbor Platform CTFd

Tampilan dasbor Grafana yang berhasil menampilkan visualisasi sumber daya pada server pusat ditunjukkan pada Gambar 4.122.



Gambar 4. 122 Dasbor Grafana Berhasil Memvisualisasikan Sumber Daya Server

Gambar 4.123 menampilkan detail *log* pada VM-1 yang berhasil dimonitoring melalui Grafana.



Gambar 4. 123 Grafana Sukses Menampilkan Data Log pada VM-1

4.4.3.2 Hasil Pengujian Pembatasan Dampak Eksploitasi oleh AppArmor

File "/var/log/syslog" pada VM-2 dan VM-3 mencatat sejumlah *deny-event* yang membuktikan AppArmor menolak operasi terlarang pada profil ketat. Rekapitulasi ditunjukkan pada Tabel 4.11.

Tabel 4. 10 Rekapitulasi *Deny-event* AppArmor pada Profil Ketat

No	Binary (Profil)	Lokasi Pengujian (VM)	Port	Operasi Ditolak	Keterangan
1.	BOF-3	VM-2	10005	*exec /usr/bin/whoami*	Eksekusi perintah di luar daftar izin ditolak (mode *enforce*)
2.	BOF-3	VM-2	10005	*open /etc/*, .bashrc*	Pembacaan *file* sistem sensitif dan *file* pengguna ditolak

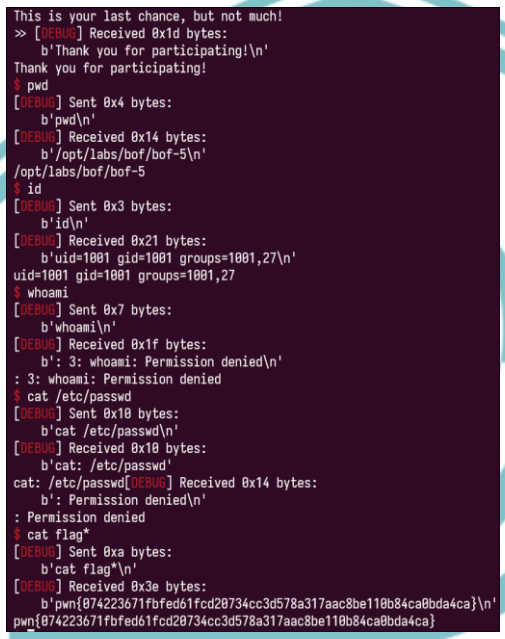
4.	FMT-3	VM-2	10006	*exec whoami, ls, id*	Rangkaian perintah enumerasi ditolak; *binary* tetap terkurung
5.	BOF-5	VM-3	10009	*exec whoami, hostname, xxd*	Beberapa upaya eksekusi perintah ditolak berturut-turut
6.	FMT-5	VM-3	10010	*open /etc/group, nsswitch.conf*	Pembacaan *file* identitas dan resolusi nama ditolak

Eksploitasi yang menargetkan *binary* dengan profil longgar mengizinkan penyerang untuk membuka file-file sensitif dan perintah apapun. Eksploitasi yang berhasil dilakukan yang menargetkan *binary* "BOF-4" di VM-3 ditunjukkan pada Gambar 4.124.

```
[ ] Switching to interactive mode
[DEBUG] Received 0xf bytes:
    b"Done, thank's!\n"
Done, thank's!
$ pwd
[DEBUG] Sent 0x4 bytes:
    b'pwd\n'
[DEBUG] Received 0x14 bytes:
    b'/opt/labs/bof/bof-4\n'
/opt/labs/bof/bof-4
$ whoami
[DEBUG] Sent 0x7 bytes:
    b'whoami\n'
[DEBUG] Received 0x4 bytes:
    b'ctf\n'
ctf
$ id
[DEBUG] Sent 0x3 bytes:
    b'id\n'
[DEBUG] Received 0x8 bytes:
    b'uid=1001'
uid=1001[DEBUG] Received 0x2e bytes:
    b'(ctf) gid=1001(ctf) groups=1001(ctf),27(sudo)\n'
(ctf) gid=1001(ctf) groups=1001(ctf),27(sudo)
$ cat /etc/passwd
[DEBUG] Sent 0x10 bytes:
    b'cat /etc/passwd\n'
[DEBUG] Received 0x7f2 bytes:
    b'root:x:0:0:root:/root:/bin/bash\n'
    b'daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin\n'
    b'bin:x:2:2:bin:/bin:/usr/sbin/nologin\n'
```

Gambar 4. 124 Eksploitasi yang Menargetkan *Binary* Berprofil Longgar Berhasil Dilakukan

Di sisi lain, eksploitasi juga berhasil dilakukan yang menargetkan *binary* "BOF-5" di VM-3 dengan profil ketat juga berhasil dilakukan. Namun, terdapat pembatasan ruang lingkup *binary* tersebut agar tidak bisa membuka *file-file* sensitif dan perintah yang tidak diizinkan. Eksploitasi yang berhasil dilakukan yang menargetkan *binary* "BOF-5" di VM-3 ditunjukkan pada Gambar 4.125.

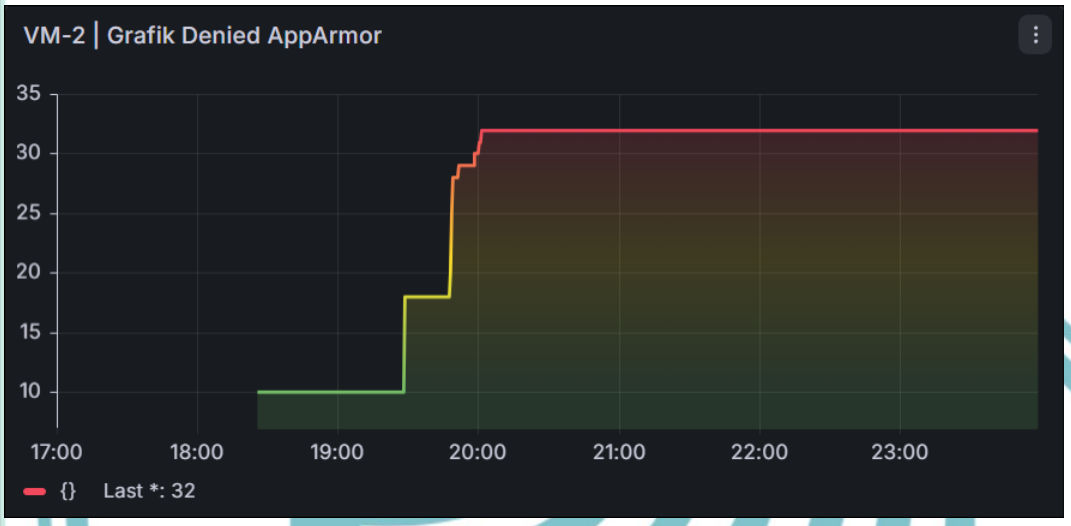


Gambar 4. 125 Eksploitasi Terhadap *Binary* Berprofil Ketat Berhasil, Tetapi Sangat Terbatas Dalam Kontrol *Shell*

Dengan begini, penyerang hanya bisa membaca *file flag* saja dan tidak bisa mendapatkan akses *shell* secara penuh. Data pelanggaran yang telah tercatat oleh AppArmor berhasil ditampilkan dan divisualisasikan di dasbor Grafana. Gambar 4.126 merupakan daftar pelanggaran yang terekam di *file "/var/log/syslog"* di VM-2.

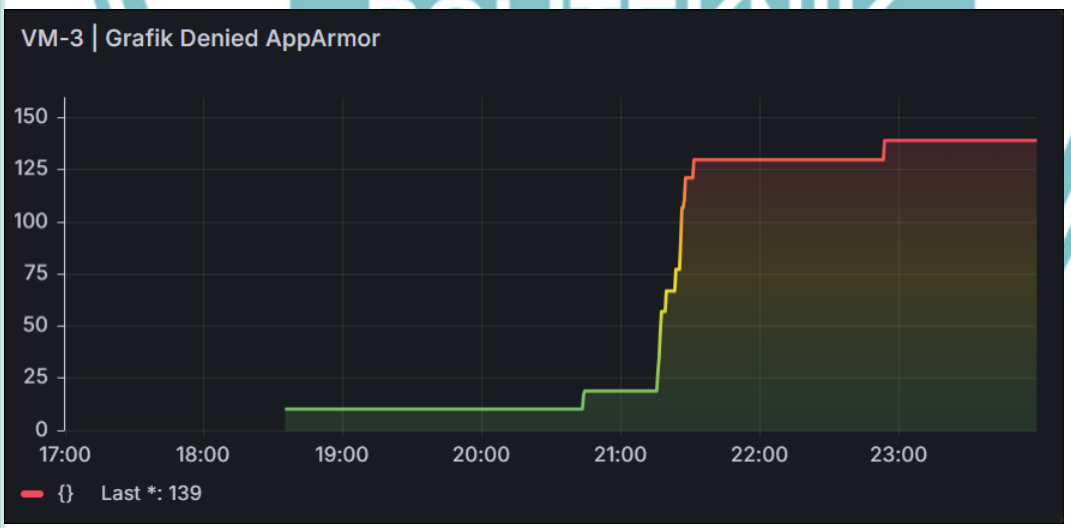
Gambar 4. 126 Kumpulan Pelanggaran yang Berhasil Dicegah oleh AppArmor Selama Proses Eksploitasi

Selama durasi eksploitasi, total pelanggaran yang berhasil dicegah oleh AppArmor di VM-2 sebanyak 32 kali. Grafik tren pelanggaran yang berhasil dicegah oleh AppArmor tertera pada Gambar 4.127.



Gambar 4. 127 Grafik Tren Pelanggaran yang Berhasil Dicegah oleh AppArmor di VM-2

Di VM-3, total pelanggaran yang berhasil dicegah oleh AppArmor ada sebanyak 139 kali. Grafik tren pelanggaran yang berhasil dicegah oleh AppArmor di VM-3 ditunjukkan pada Gambar 4.128.



Gambar 4. 128 Grafik Tren Pelanggaran yang Berhasil Dicegah oleh AppArmor di VM-3

4.4.3.3 Hasil Pengujian Pemblokiran Alamat IP oleh Fail2Ban

Berdasarkan file *"/var/log/labs/crash.log"* dan *"/var/log/labs/fail2ban.log"* pada VM-3, terdapat empat peristiwa pemblokiran akibat upaya eksploitasi yang gagal mencapai tahap akses *shell*, sehingga menyebabkan proses *binary* mengalami *crash*. Sesuai dengan konfigurasi *"crash-detection"* pada Fail2Ban, di mana total *crash* yang terjadi sebanyak 3 kali berturut-turut kurang dari waktu 10 menit, maka alamat IP penyerang akan diblokir oleh Fail2Ban. Rekapitulasi jumlah peristiwa *crash* per alamat IP ditunjukkan pada Tabel 4.12.

Tabel 4. 11 Rekapitulasi Peristiwa *Crash* per Alamat IP dan Status Pemblokiran (VM-3)

No.	Alamat IP	Jumlah *Crash*	Status	Keterangan
1.	138.199.22.99	1	Tidak diblokir	Di bawah ambang batas, sehingga lolos sesuai desain toleransi *false-positive*
2.	158.140.162.198	3	Diblokir	Mencapai ambang tepat pada batas *window* (kurun waktu) 10 menit
3.	180.252.161.153	7	Diblokir	Melewati ambang batas - diblokir dua kali pada sesi berbeda
4.	182.253.250.254	7	Diblokir	Melewati ambang batas - *crash* berulang pada *binary* "FMT-4"

Detail peristiwa pemblokiran alamat IP penyerang beserta waktu awal *crash*, waktu blokir, dan sinyal *error* ditunjukkan pada Tabel 4.13.

Tabel 4. 12 Detail Peristiwa Pemblokiran *Crash-Detection* (VM-3)

No.	Waktu Awal Crash	Waktu Blokir	Alamat IP	Port	Sinyal Eror	Keterangan
1.	20:11:21	20:21:01	158.140.162.198	10007	SIGSEGV (11)	BOF-4; *segmentation fault*
2.	21:34:34	21:38:49	182.253.250.254	10008	SIGSEGV (11)	FMT-4; *segmentation fault*
3.	22:54:23	22:56:12	180.252.161.153	10009	SIGABRT (6)	BOF-5; *stack smashing*

Ketiga alamat IP penyerang berhasil diblokir oleh Fail2Ban karena berkali-kali gagal melakukan upaya eksploitasi ditunjukkan pada Gambar 4.129.

```
ctf@VM-3:~$ sudo fail2ban-client status crash-detection
Status for the jail: crash-detection
|- Filter
|  |- Currently failed: 0
|  |- Total failed:     3
|  `- File list:        /var/log/labs/crash.log
`- Actions
   |- Currently banned: 3
   |- Total banned:     3
   `- Banned IP list:   158.140.162.198 180.252.161.153 182.253.250.254
ctf@VM-3:~$
```

Gambar 4. 129 Alamat IP Penyerang yang Berhasil Diblokir oleh Fail2Ban Akibat Berulang Kali Menyebabkan *Crash*

Sebagai bukti jika salah satu penyerang sudah tidak bisa lagi mengakses *binary* melalui jaringan (*Connection Refused*) ditunjukkan pada Gambar 4.130.

```
┌─[brojack@parrot]─[~/github/CTFDump/ctf.pwngame.site/Binary exploitation (hard)
/BOF-5/read]
└──╼ $python solver2.py
[-] Opening connection to 149.129.193.12 on port 10009: Failed
[ERROR] Could not connect to 149.129.193.12 on port 10009
Traceback (most recent call last):
  File "/home/brojack/github/CTFDump/ctf.pwngame.site/Binary exploitation (hard)
/BOF-5/read/solver2.py", line 22, in <module>
    p = remote(HOST, PORT)
  File "/usr/lib/python3/dist-packages/pwnlib/tubes/remote.py", line 85, in __in
it__
    self.sock   = self._connect(fam, typ)
                  ~~~~~~~~~~~~~^^^^^^^^^^
  File "/usr/lib/python3/dist-packages/pwnlib/tubes/remote.py", line 134, in _co
nnect
    self.error("Could not connect to %s on port %s", self.rhost, self.rport)
    ~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/lib/python3/dist-packages/pwnlib/log.py", line 439, in error
    raise PwnlibException(message % args)
pwnlib.exception.PwnlibException: Could not connect to 149.129.193.12 on port 10
009
```

Gambar 4. 130 Salah Satu IP Penyerang Sudah Tidak Bisa Mengakses *Binary*

Detail alamat IP penyerang pada log Fail2Ban yang berhasil diblokir akibat seringnya terjadi *crash* ditunjukkan pada Gambar 4.131.

```
2026-07-15 22:56:12,854 fail2ban.actions        [43313]: NOTICE  [crash-detectio
n] Ban 180.252.161.153
2026-07-15 21:38:49,380 fail2ban.actions        [43313]: NOTICE  [crash-detectio
n] Ban 182.253.250.254
2026-07-15 20:21:01,881 fail2ban.actions        [43313]: NOTICE  [crash-detectio
n] Ban 158.140.162.198
```

Gambar 4. 131 *Log* Fail2Ban yang Berhasil Memblokir Ketiga Alamat IP Penyerang

4.4.3.4 Hasil Pengujian Pembanjiran Koneksi (*TCP Connection Flooding*)

Pengujian ini dilaksanakan secara mandiri (terpisah dari waktu ajang CTF), karena penyerang hanya berfokus eksploitasi *binary* untuk mendapatkan akses *shell*. Berdasarkan hasil pengujian yang tersimpan pada *file "/var/log/labs/conn.log"*, VM-3 mencatat adanya lonjakan pembukaan koneksi secara masif dari alamat IP *180.252.168.207* menuju *port* 10009 (BOF-5).

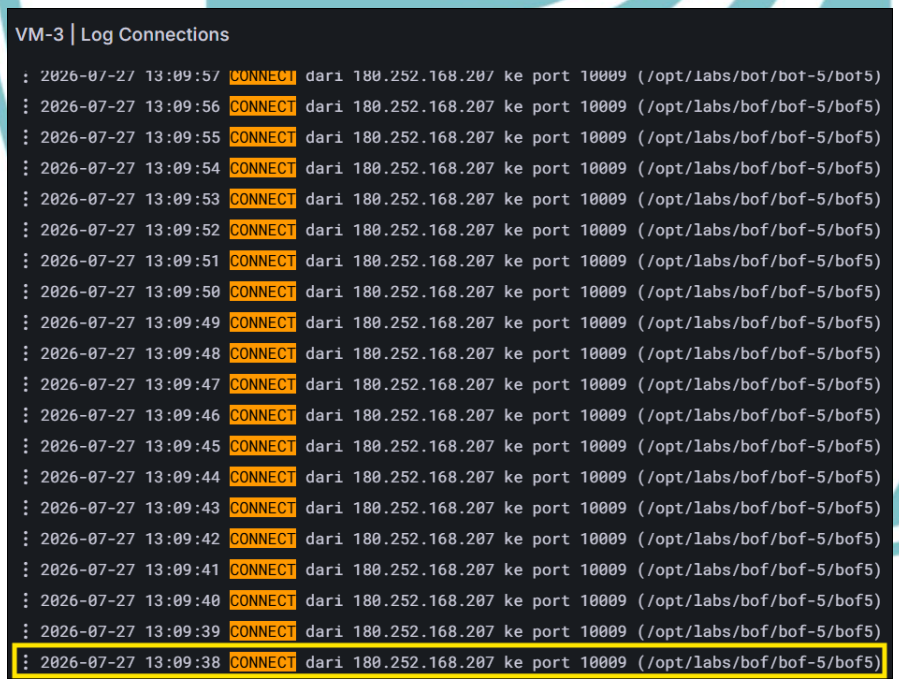
Alamat IP tersebut berupaya membuka koneksi setiap 1 detik yang dimulai pada pukul 13:09:38. Asumsi waktu yang dibutuhkan jika 1 detik sama dengan 1 koneksi,

maka untuk mencapai ambang batas *flood detection* adalah 50 – 60 detik. Fail2Ban memblokir alamat IP tersebut pada pukul 13:10:28, sebagaimana ditunjukkan pada Tabel 4.13.

Tabel 4. 13 Detail Peristiwa Pemblokiran *Flood-Detection* (VM-3)

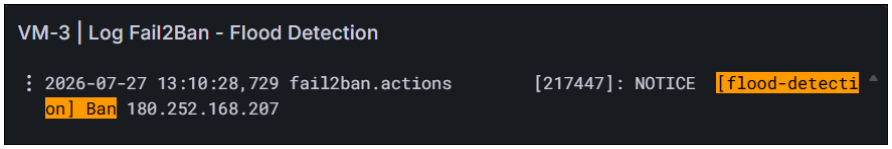
No.	Waktu Awal Serangan	Waktu Blokir	Alamat IP	*Port*	*Binary*	Keterangan
1.	13:09:38	13:10:28	180.252.168.207	10009	BOF-5	Sekitar 51 koneksi dari satu sumber dalam < 1 menit

Waktu mulainya serangan pembanjiran koneksi oleh alamat IP *180.252.168.207* tertera pada Gambar 4.132.



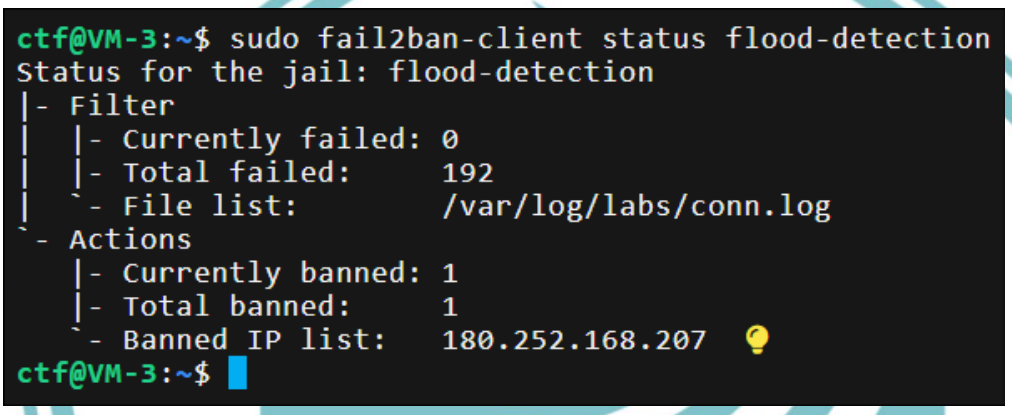
Gambar 4. 132 Penyerang Memulai Serangan Pembanjiran Koneksi

Log Fail2Ban berhasil mencatat sumber alamat IP yang melakukan serangan pembanjiran koneksi ditunjukkan pada Gambar 4.133.



Gambar 4. 133 *Log* Fail2Ban yang Berhasil Memblokir Serangan Pembanjiran Koneksi

Alamat IP yang berhasil dikurung atau diblokir oleh Fail2ban juga dapat dilihat pada status *jail "flood-detection"*, sebagaimana yang tertera pada Gambar 4.134.



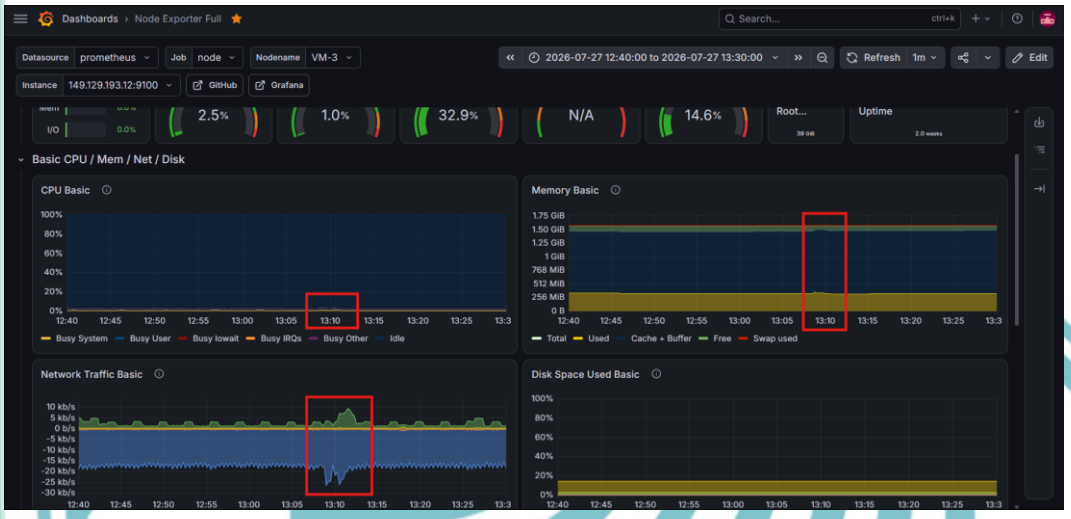
Gambar 4. 134 Alamat IP Penyerang yang Berhasil Diblokir oleh Fail2Ban Akibat Pembanjiran Koneksi

Adapun beban server yang juga dihitung selama serangan pembanjiran koneksi. Upaya tersebut terekam melalui *Node Exporter* yang ditunjukkan pada Tabel 4.14.

Tabel 4. 14 Beban Server VM-3 Kondisi Normal dan Terjadi Serangan Pembanjiran Koneksi

	Kondisi Normal			***TCP Connection Flooding***		
	CPU (%)	**Mem (MiB)**	**Rx Eth0**	**CPU (%)**	**Mem (MiB)**	**Rx Eth0**
Min	0,49%	327 MiB	1,15 Kb/s	0,53%	327 MiB	1,68 Kb/s
Max	1%	361 MiB	7,09 Kb/s	1,32%	348 MiB	9,67 Kb/s
Avg	0,68%	335 MiB	2,29 Kb/s	0,93%	336 MiB	5,54 Kb/s

Dari Tabel 4.15, beban server saat ratusan koneksi membanjiri VM-3 tidak terlalu melonjak secara signifikan, sebagaimana yang tertera pada grafik pada Gambar 4.135.



Gambar 4. 135 Grafik Penggunaan CPU, RAM, dan *Incoming Traffic* pada VM-3 Selama Pembanjiran Koneksi

4.4.4 Analisis Data dan Evaluasi

4.4.4.1 Analisis Pengujian Fungsionalitas Sistem

Seluruh komponen pada ketiga VM (VM-1 s.d VM-3) berstatus berhasil, sehingga sistem dinyatakan layak melanjutkan ke pengujian inti. Perlu ditekankan bahwa pengujian ini bersifat verifikasi ketersediaan, bukan verifikasi sistem keamanan. Keberhasilan pada pengujian ini tidak memberikan dampak terhadap ketahanan eksploitasi, melainkan memastikan tiap lapisan pertahanan berada pada posisi yang benar untuk diuji. Dengan konfirmasi bahwa platform CTFd dapat diakses melalui internet dan *service-service* yang diperlukan sudah aktif secara permanen, serta Grafana dapat berfungsi untuk memvisualisasikan *log* dan sumber daya setiap VM, maka keabsahan data pada pengujian selanjutnya dapat dipertanggungjawabkan.

4.4.4.2 Analisis Pengujian Pembatasan Dampak Eksploitasi oleh AppArmor

Kontras antara kedua profil di VM-2 dan VM-3 memperlihatkan peran AppArmor secara jelas. Pada profil longgar, eksploitasi yang berhasil berlanjut hingga penyerang memperoleh *shell* interaktif dan membaca file sensitif di server, sehingga dampak tidak terbendung. Sebaliknya, pada profil ketat, eksploitasi *binary* tetap terjadi, tetapi setiap upaya eskalasi, seperti eksekusi perintah *"whoami"*,

"hostname", *"ls"*, *"xxd"*, maupun pembacaan *file* identitas akan langsung ditolak dan tercatat sebagai peristiwa pelanggaran *(deny-event)*.

Hal ini menegaskan bahwa AppArmor tidak mencegah terjadinya eksploitasi, melainkan membatasi dampak lanjutannya di tingkat sistem operasi. Konsekuensi penting yang harus diakui secara eksplisit adalah bahwa eksploitasi yang berhasil tanpa memicu operasi terlarang tidak akan menghasilkan *deny-event*. Dengan demikian, klaim efektivitas AppArmor selalu berkualifikasi, yaitu efektif membatasi dampak, bukan menambal kerentanan pada *binary* itu sendiri.

4.4.4.3 Analisis Pengujian Pemblokiran Alamat IP oleh Fail2Ban

Dari empat alamat IP yang tercatat melakukan *crash* pada VM-3, tiga di antaranya mencapai atau melewati ambang *maxretry* dan seluruhnya berhasil diblokir. Dengan demikian, efektivitas pemblokiran terhadap alamat IP yang melewati ambang adalah tiga dari tiga, atau memiliki 100% efektivitas pemblokiran.

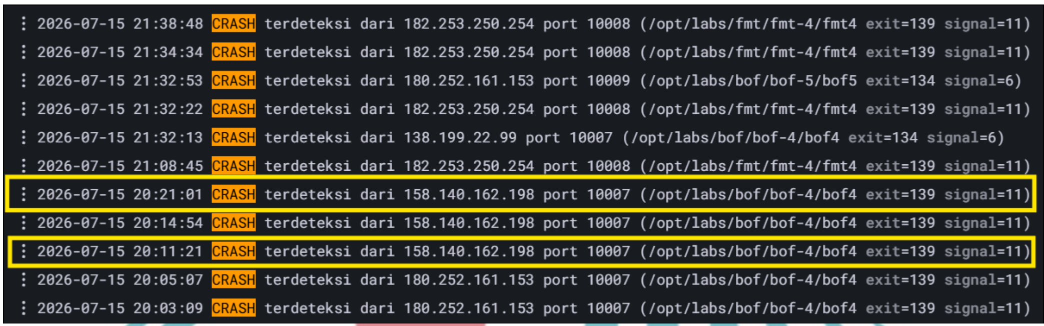
Ketiga alamat IP yang berhasil diblokir itu sebabkan karena memang melewati ambang batas yang telah ditentukan, bukan terhadap seluruh alamat IP yang pernah menyebabkan *crash*. Analisis pemblokira alamat IP pada Tabel 4.10 adalah sebagai berikut:

1) Alamat IP *138.199.22.99* yang hanya memicu sekali *crash*, sehingga tidak diblokir oleh Fail2Ban. Hal ini bukan kegagalan sistem, melainkan konsekuensi desain toleransi *false-positive* yang menetapkan *maxretry* sama dengan tiga. Alamat IP eksternal yang terdeteksi melakukan *crash "Stack Smashing Detected"* (sinyal nomor 6) tertera pada Gambar 4.136.

```
2026-07-15 22:54:23 CRASH terdeteksi dari 180.252.161.153 port 10009 (/opt/labs/bof/bof-5/bof5 exit=134 signal=6)
2026-07-15 21:43:49 CRASH terdeteksi dari 182.253.250.254 port 10008 (/opt/labs/fmt/fmt-4/fmt4 exit=142 signal=14)
2026-07-15 21:38:49 CRASH terdeteksi dari 182.253.250.254 port 10008 (/opt/labs/fmt/fmt-4/fmt4 exit=139 signal=11)
2026-07-15 21:38:48 CRASH terdeteksi dari 182.253.250.254 port 10008 (/opt/labs/fmt/fmt-4/fmt4 exit=139 signal=11)
2026-07-15 21:34:34 CRASH terdeteksi dari 182.253.250.254 port 10008 (/opt/labs/fmt/fmt-4/fmt4 exit=139 signal=11)
2026-07-15 21:32:53 CRASH terdeteksi dari 180.252.161.153 port 10009 (/opt/labs/bof/bof-5/bof5 exit=134 signal=6)
2026-07-15 21:32:22 CRASH terdeteksi dari 182.253.250.254 port 10008 (/opt/labs/fmt/fmt-4/fmt4 exit=139 signal=11)
2026-07-15 21:32:13 CRASH terdeteksi dari 138.199.22.99 port 10007 (/opt/labs/bof/bof-4/bof4 exit=134 signal=6)
2026-07-15 21:08:45 CRASH terdeteksi dari 182.253.250.254 port 10008 (/opt/labs/fmt/fmt-4/fmt4 exit=139 signal=11)
2026-07-15 20:21:01 CRASH terdeteksi dari 158.140.162.198 port 10007 (/opt/labs/bof/bof-4/bof4 exit=139 signal=11)
2026-07-15 20:14:54 CRASH terdeteksi dari 158.140.162.198 port 10007 (/opt/labs/bof/bof-4/bof4 exit=139 signal=11)
2026-07-15 20:11:21 CRASH terdeteksi dari 158.140.162.198 port 10007 (/opt/labs/bof/bof-4/bof4 exit=139 signal=11)
2026-07-15 20:05:07 CRASH terdeteksi dari 180.252.161.153 port 10007 (/opt/labs/bof/bof-4/bof4 exit=139 signal=11)
2026-07-15 20:03:09 CRASH terdeteksi dari 180.252.161.153 port 10007 (/opt/labs/bof/bof-4/bof4 exit=139 signal=11)
```

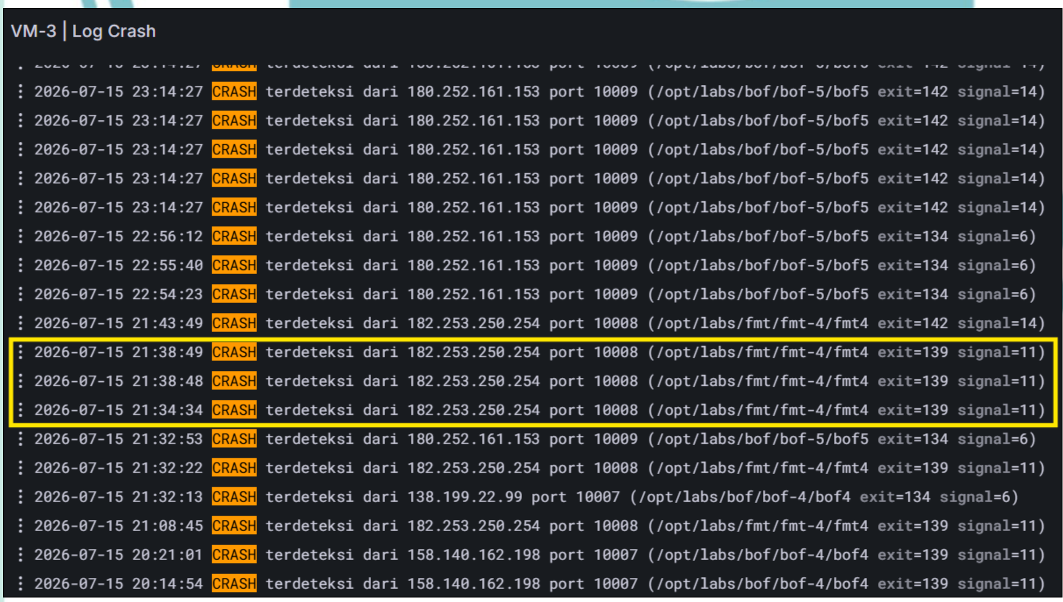
Gambar 4. 136 *Crash* Terdeteksi Dilakukan oleh Alamat IP 138.199.22.99

2) Pemblokiran pada alamat IP *158.140.162.198* terjadi dengan *blocking time* sekitar sepuluh menit. Di mana, *crash* awal terjadi pada pukul 20:11:21 dan berhasil diblokir pada pukul 20:21:01, tepat pada batas *window*. Data *log* pemblokiran alamat IP *158.140.162.198* tertera pada Gambar 4.137.



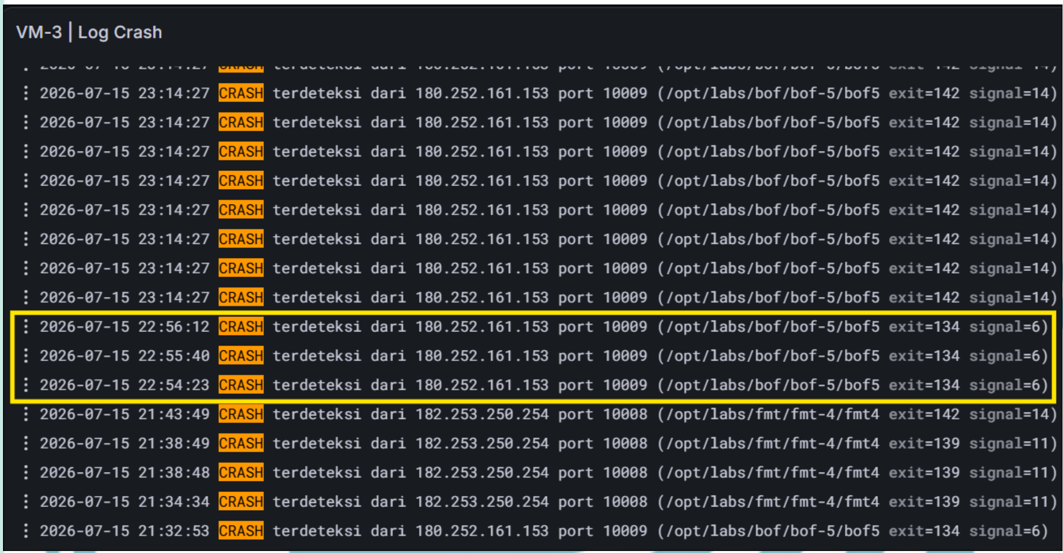
Gambar 4. 137 Detail *Log* Alamat IP Penyerang ke-1 Berhasil Diblokir

3) Pemblokiran pada alamat IP *182.253.250.254* terjadi karena eksploitasi memicu *error* sinyal nomor 11 *"Segmentation Fault"*. Waktu awal muncul *crash* adalah pukul 21:34:34 dan berhasil diblokir pada pukul 21:38:49. Data *log* pemblokiran alamat IP *182.253.250.254* tertera pada Gambar 4.138.



Gambar 4. 138 Detail *Log* Alamat IP Penyerang ke-2 Berhasil Diblokir

4) Pemblokiran pada alamat IP *180.252.161.153* terjadi 2 akibat memicu *crash signal* 6 *"Stack Smashing Detected"*. Waktu awal muncul *crash* adalah pukul 22:54:23 dan waktu blokir adalah 22:56:12. Data *log* pemblokiran alamat IP *180.252.161.153* tertera pada Gambar 4.139.


VM-3 | Log Crash

```
2026-07-15 23:14:27 CRASH terdeteksi dari 180.252.161.153 port 10009 (/opt/labs/bof/bof-5/bof5 exit=142 signal=14)
2026-07-15 23:14:27 CRASH terdeteksi dari 180.252.161.153 port 10009 (/opt/labs/bof/bof-5/bof5 exit=142 signal=14)
2026-07-15 23:14:27 CRASH terdeteksi dari 180.252.161.153 port 10009 (/opt/labs/bof/bof-5/bof5 exit=142 signal=14)
2026-07-15 23:14:27 CRASH terdeteksi dari 180.252.161.153 port 10009 (/opt/labs/bof/bof-5/bof5 exit=142 signal=14)
2026-07-15 23:14:27 CRASH terdeteksi dari 180.252.161.153 port 10009 (/opt/labs/bof/bof-5/bof5 exit=142 signal=14)
2026-07-15 23:14:27 CRASH terdeteksi dari 180.252.161.153 port 10009 (/opt/labs/bof/bof-5/bof5 exit=142 signal=14)
2026-07-15 23:14:27 CRASH terdeteksi dari 180.252.161.153 port 10009 (/opt/labs/bof/bof-5/bof5 exit=142 signal=14)
2026-07-15 23:14:27 CRASH terdeteksi dari 180.252.161.153 port 10009 (/opt/labs/bof/bof-5/bof5 exit=142 signal=14)
2026-07-15 22:56:12 CRASH terdeteksi dari 180.252.161.153 port 10009 (/opt/labs/bof/bof-5/bof5 exit=134 signal=6)
2026-07-15 22:55:40 CRASH terdeteksi dari 180.252.161.153 port 10009 (/opt/labs/bof/bof-5/bof5 exit=134 signal=6)
2026-07-15 22:54:23 CRASH terdeteksi dari 180.252.161.153 port 10009 (/opt/labs/bof/bof-5/bof5 exit=134 signal=6)
2026-07-15 21:43:49 CRASH terdeteksi dari 182.253.250.254 port 10008 (/opt/labs/fmt/fmt-4/fmt4 exit=142 signal=14)
2026-07-15 21:38:49 CRASH terdeteksi dari 182.253.250.254 port 10008 (/opt/labs/fmt/fmt-4/fmt4 exit=139 signal=11)
2026-07-15 21:38:48 CRASH terdeteksi dari 182.253.250.254 port 10008 (/opt/labs/fmt/fmt-4/fmt4 exit=139 signal=11)
2026-07-15 21:34:34 CRASH terdeteksi dari 182.253.250.254 port 10008 (/opt/labs/fmt/fmt-4/fmt4 exit=139 signal=11)
2026-07-15 21:32:53 CRASH terdeteksi dari 180.252.161.153 port 10009 (/opt/labs/bof/bof-5/bof5 exit=134 signal=6)
```

Gambar 4. 139 Detail *Log* Alamat IP Penyerang ke-3 Berhasil Diblokir

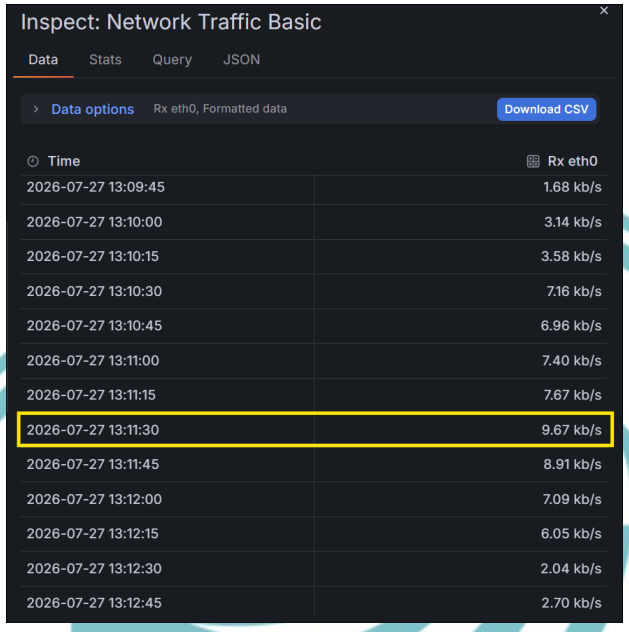
Dari hasil pemblokiran ketiga alamat IP penyerang, terdapat variasi *crash* yang terpicu, yaitu sinyal nomor 6 *"Stack Smashing Detected"* dan sinyal nomor 11 *"Segmentation Fault"*. Adapun sinyal nomor 14 adalah SIGALRM yang berarti program otomatis keluar jika tidak ada interaksi dengan pengguna (kondisi *idle*).

4.4.4.3 Analisis Pengujian Pembanjiran Koneksi (*TCP Connection Flooding*)

Data log koneksi yang tertera pada Tabel 4.15 mengonfirmasi adanya serangan pembanjiran ke salah satu *binary* yang berjalan di VM-3, yaitu "BOF-5" di *port* 10009. Berdasarkan log serangan pada Gambar 4.132, penyerang membuka 1 koneksi setiap detik, sehingga waktu yang diperlukan untuk mencapai ambang batas (50 koneksi) adalah 50 – 60 detik. Dari hasil pengujian, Fail2Ban berhasil memblokir alamat IP penyerang setelah ambang *flood-detection* terlewati, dengan selisih waktu pada orde detik.

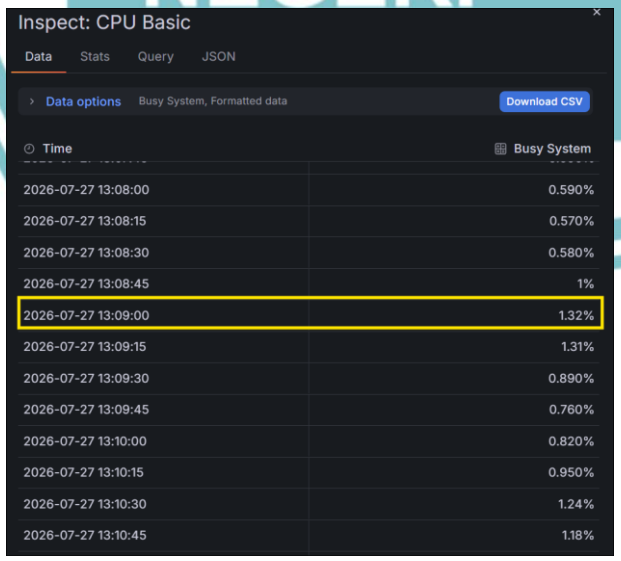
Data metrik pada Tabel 4.16 memperkuat temuan ini secara berimbang. Lalu lintas jaringan masuk (*Rx eth0*) menunjukkan kenaikan paling nyata, yaitu dari rata-rata kondisi normal 2,29 kb/s menjadi puncak 9,67 kb/s atau naik sekitar 323%, tepat

setelah serangan *TCP connection flooding* diblokir Fail2Ban. Data lonjakan *incoming traffic* pada kasus pembanjiran koneksi tertera pada Gambar 4.140.



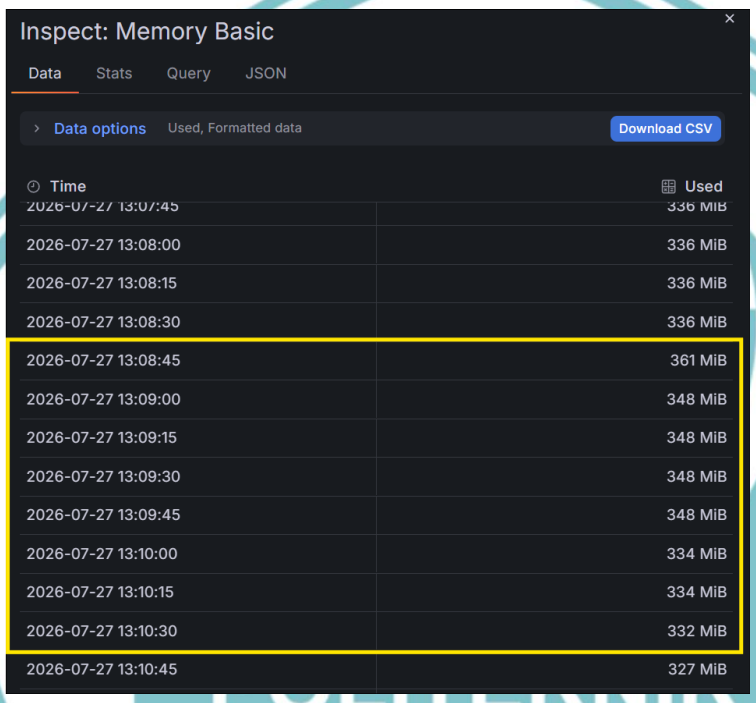
Gambar 4. 140 Data Lonjakan Trafik Jaringan Ketika Terjadi *TCP Connection Flooding*

Beban CPU juga meningkat, yaitu dari rata-rata normal 0,68% menjadi puncak 1,32% atau naik sekitar 95%, meskipun secara absolut nilainya tetap rendah. Data peningkatan penggunaan CPU pada pengujian pembanjiran koneksi tertera pada Gambar 4.141.



Gambar 4. 141 Data Lonjakan Penggunaan CPU Ketika Terjadi *TCP Connection Flooding*

Sebaliknya, memori terpakai tidak terpengaruh secara signifikan, di mana rata-ratanya nyaris tidak berubah dan puncak tertinggi (361 MiB) justru terjadi pada kondisi normal, bukan saat pembanjiran koneksi. Hal ini konsisten dengan karakteristik *TCP connection flooding* yang membebani pemrosesan koneksi dan lalu lintas jaringan, bukan alokasi memori. Rata-rata kestabilan memori saat pembanjiran koneksi berlangsung tertera pada Gambar 4.142.



Gambar 4. 142 Data Kestabilan Konsumsi Memori Ketika Terjadi *TCP Connection Flooding*

Nilai beban memori secara keseluruhan tetap rendah, jauh di bawah kapasitas total ketersediaan pada VM-3, yaitu 1,58 GiB (~2 GB). Hal tersebut dikarenakan Fail2Ban memblokir alamat IP penyerang dengan cepat sebelum akumulasi koneksi menekan sumber daya server. Keberhasilan pemblokiran pada pengujian ini berjalan secara linier dengan konfigurasi ambang batas. Namun, generalisasi terhadap seluruh pola *flooding* memerlukan pengujian dengan variasi laju koneksi yang lebih luas.

BAB V
PENUTUP

5.1 Kesimpulan

Berdasarkan perancangan dan pengujian sistem keamanan pada Ubuntu Server terhadap eksploitasi celah *buffer overflow* dan *format string*, dapat diambil kesimpulan sebagai berikut:

1. AppArmor efektif dalam membatasi dampak lanjutan dari eksploitasi yang telah berhasil, bukan menambal kerentanan *binary* itu sendiri. Hal ini dibuktikan secara kontras antara profil AppArmor longgar dan profil ketat pada VM-2 dan VM-3. Di mana, profil longgar masih mengizinkan penyerang memperoleh *shell* dan membaca *file-file* sistem sensitif. Sedangkan, pada profil ketat, setiap upaya eskalasi seperti eksekusi perintah yang tidak diizinkan, seperti *"whoami"*, *"hostname"*, *"ls"*, dan *"xxd"*, serta pembacaan file-file sensitif berhasil ditolak dan tercatat sebagai peristiwa pelanggaran *(deny-event)*. Dengan demikian, klaim efektivitas AppArmor bersifat berkualifikasi, yaitu efektif sebagai lapisan pencegah di tingkat sistem operasi (Ubuntu Server).
2. Fail2Ban efektif memblokir alamat IP penyerang berdasarkan bukti *crash* berulang di tingkat jaringan. Setiap alamat IP yang melewati ambang batas *(maxretry)*, yaitu 3 alamat IP yang melakukan *crash* berulang serta alamat IP lain yang melakukan pembanjiran IP berhasil diblokir oleh Fail2Ban. Alamat IP yang hanya menimbulkan *crash* di bawah ambang tidak diblokir, didesain menggunakan 3 kali toleransi pelanggaran agar tidak terjadi *false-positive*. Pemblokiran didasarkan pada bukti *crash* yang terjadi akibat input melebihi batas maksimum karakter yang telah dialokasikan atau kesalahan dalam proses eksploitasi. Sinyal *crash* dari hasil pengujian ada beragam, yang mencakup *error Segmentation Fault* (menghasilkan sinyal nomor 11) dan *Stack Smashing Detected* (menghasilkan sinyal nomor 6) yang memicu pemblokiran. Selain itu, terdapat sinyal nomor 14 *(SIGALRM*) yang secara otomatis menginterupsi proses jika tidak ada interaksi selama jangka waktu tertentu (kondisi *idle*).

3. Kombinasi AppArmor dan Fail2Ban membentuk arsitektur pertahanan berlapis yang saling melengkapi sesuai peran masing-masing. Pada pengujian pembanjiran koneksi (*TCP connection flooding*), Fail2Ban memblokir alamat IP pembanjir dalam orde detik, sehingga beban server tetap rendah, yaitu CPU pada puncak 1,32% dan lalu lintas jaringan (*Rx eth0*) pada puncak 9,67 kb/s. Di sisi lain, penggunaan memori tidak terpengaruh secara signifikan. Batasan sistem yang harus diakui secara eksplisit adalah bahwa eksploitasi yang berhasil tanpa menimbulkan *crash* tidak terdeteksi oleh Fail2Ban, dan pada kondisi tersebut pertahanan sepenuhnya bertumpu pada AppArmor sebagai lapisan pencegah.

5.2 Saran

Rancangan sistem keamanan yang telah terbukti efektif masih dapat dikembangkan dalam aspek lainnya. Ada beberapa saran yang dapat dilakukan untuk penelitian selanjutnya, di antaranya:

1. Notifikasi pemblokiran saat ini hanya tercatat pada *log* dan memerlukan pemeriksaan manual. Disarankan menambahkan mekanisme notifikasi real-time, seperti bot Telegram agar tim *Security Operation Center* (SOC) memperoleh peringatan langsung saat terjadi pemblokiran, sehingga respons terhadap serangan dapat dilakukan lebih cepat.
2. Perlu ditambahkan mekanisme deteksi terhadap eksploitasi yang berhasil tanpa menimbulkan *crash*. Misalnya, melalui pemantauan perilaku proses (*runtime behavior monitoring*) atau *audit log* pada level *syscall*, karena eksploitasi jenis ini merupakan celah yang lolos dari deteksi Fail2Ban.
3. Cakupan celah keamanan dalam penelitian ini terbatas pada *buffer overflow* dan *format string*. Misalnya, *heap overflow*, *use-after-free* (UAF), atau *double free*. Tujuannya untuk menilai generalisasi efektivitas pendekatan *containment* dan *enforcement* berlapis ini.
4. Penerapan *secure coding* pada tahap pengembangan *binary* perlu diintegrasikan sebagai lapisan pertahanan tambahan, karena AppArmor dan Fail2Ban membatasi dampak dan memblokir serangan, tetapi tidak menghilangkan akar kerentanan pada kode itu sendiri.

DAFTAR PUSTAKA

Adiyoso, W., & Susetyo, Y. A. (2023). PEMBANGUNAN COMPILER DOMAIN SPECIFIC LANGUAGE SEBAGAI GENERATOR FORM HTML MENGGUNAKAN PYTHON SLY. *Jurnal Indonesia : Manajemen Informatika dan Komunikasi*, *4*(2), 449–461. https://doi.org/10.35870/jimik.v4i2.231

Alviano, M., & Sestito, P. (2025). User Armor: An Extension for AppArmor. *Algorithms*, *18*(4), 185. https://doi.org/10.3390/a18040185

Amrullah, H. I., Amarta, A. N. F., & Rilvani, E. (2025). Fleksibilitas dan Kesederhanaan Arsitektur Sistem Operasi Linux. *Neptunus: Jurnal Ilmu Komputer Dan Teknologi Informasi*, *3*(1), 59–64. https://doi.org/10.61132/neptunus.v3i1.630

Anderson, S. N., Blanco, R., Lampropoulos, L., Pierce, B. C., & Tolmach, A. (2023). Formalizing Stack Safety as a Security Property. *2023 IEEE 36th Computer Security Foundations Symposium (CSF)*, 356–371. https://doi.org/10.1109/CSF57540.2023.00037

Ansar, M. I. (2025a). *Apa itu NGINX? Fungsi, Manfaat, dan Kelebihannya*. https://digitalsolusigrup.co.id/nginx-adalah/

Ansar, M. I. (2025b, September 10). *Apa Itu Grafana? Fungsi, Fitur, dan Manfaat Penggunaannya*. https://digitalsolusigrup.co.id/grafana-adalah/

Balqis, P., & Rahman, R. (2025). Analisis Keamanan Layanan SSH terhadap Brute Force Attack. *Merkurius : Jurnal Riset Sistem Informasi dan Teknik Informatika*, *3*(4), 240–246. https://doi.org/10.61132/merkurius.v3i4.949

Bosman, E., & Bos, H. (2014). Framing Signals—A Return to Portable Shellcode. *2014 IEEE Symposium on Security and Privacy*, 243–258. https://doi.org/10.1109/SP.2014.23

Boyanov, P. (2023). BASIC NETWORK PENETRATION TESTING WITH THE NETWORK TOOL NETCAT IN LINUX-BASED OPERATING

SYSTEMS. *Journal Scientific and Applied Research*, *25*(1), 15–30. https://doi.org/10.46687/jsar.v25i1.377

Butt, M. A., Ajmal, Z., Khan, Z. I., Idrees, M., & Javed, Y. (2022). An In-Depth Survey of Bypassing Buffer Overflow Mitigation Techniques. *Applied Sciences*, *12*(13), 6702. https://doi.org/10.3390/app12136702

Canella, C., Werner, M., Gruss, D., & Schwarz, M. (2021). Automating Seccomp Filter Generation for Linux Applications. *Proceedings of the 2021 on Cloud Computing Security Workshop*, 139–151. https://doi.org/10.1145/3474123.3486762

ÇeliK, E., & Dal, D. (2022). Comparison of the Sizes of Assembly Codes Generated by the GCC Compiler for Different CPU Architectures. *International Conference on Informatics and Computer Science*, 8–13. https://www.researchgate.net/publication/377400794_Comparison_of_the_Sizes_of_Assembly_Codes_Generated_by_the_GCC_Compiler_for_Different_CPU_Architectures

Chen, Z., Pan, B., & Sun, Y. (2019). A Survey of Software Reverse Engineering Applications. *In: Sun, X., Pan, Z., Bertino, E. (Eds) Artificial Intelligence and Security*, *11635*, 235–245. https://doi.org/10.1007/978-3-030-24268-8_22

Creswell, J. W., & Creswell, J. D. (2022). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches* (6th ed.). SAGE Publishing. https://www.sagepub.com/shop/buy-a-book/research-design-6-270550

Cybersecurity and Infrastructure Security Agency & Federal Bureau of Investigation. (2025, February 12). Secure by Design Alert: Eliminating Buffer Overflow Vulnerabilities. *FACT SHEET*. https://www.cisa.gov/resources-tools/resources/secure-design-alert-eliminating-buffer-overflow-vulnerabilities

Dawamsyach, F., Ruslianto, I., & Ristian, U. (2023). Implementasi IPS (Intrusion Prevention System) Fail2ban Pada Server Terhadap Serangan DDoS dan

Brute Force. *CESS (Journal of Computer Engineering, System and Science)*, *8*(1), 149–161.

Degioanni, L. (2018, June 29). *3 phases of Prometheus adoption.* https://www.sysdig.com/blog/3-phases-of-prometheus-adoption

Dickinson, S. (2019, April 26). An introduction to AppArmor. *Ubuntu Blog*. https://ubuntu.com/blog/an-introduction-to-apparmor

Elradi, M. D. (2025). Prometheus & Grafana: A Metrics-focused Monitoring Stack. *Journal of Computer Allied Intelligence*, *3*(3), 28–39. https://doi.org/10.69996/jcai.2025015

El-Zoghby, A. M., ElSayed, M. S., Jurcut, A., & Azer, M. A. (2025). Overview of the Code-Reuse Attacks Mitigations, and Evaluation using SMAA-2 Approach. *Advances in Knowledge-Based Systems, Data Science, and Cybersecurity*, 108–148.

Erawan, E., & Salman, M. (2023). Penguatan Keamanan Otomatis pada Sistem Operasi Ubuntu berbasis Image Mesin Virtual menggunakan solusi Packer. *Cakrawala Repositori IMWI*, *6*(4), 1089–1097. https://doi.org/10.52851/cakrawala.v6i4.451

Farhannullah, & Hardjianto, M. (2022). Sistem Monitoring Serangan SSH dengan Metode Intrusion Prevention System (IPS) Fail2ban Menggunakan Python Pada Sistem Operasi Linux. *Jurnal Ticom: Technology of Information and Communication*, *11*(1), 33–38. https://doi.org/10.70309/ticom.v11i1.68

Full Stack Python. (n.d.-a). *Terminal Multiplexer*. Terminal Multiplexer. Retrieved June 25, 2026, from https://www.fullstackpython.com/tmux.html

Full Stack Python. (n.d.-b). *Why Use Python?* Full Stack Python. Retrieved June 25, 2026, from https://www.fullstackpython.com/why-use-python.html

Gaidis, A. J., Moreira, J., Sun, K., Milburn, A., Atlidakis, V., & Kemerlis, V. P. (2023). FineIBT: Fine-grain Control-flow Enforcement with Indirect Branch Tracking. *Proceedings of the 26th International Symposium on*

Research in Attacks, Intrusions and Defenses, 527–546. https://doi.org/10.1145/3607199.3607219

GEF Documentation. (n.d.). *GEF - GDB Enhanced Features*. GEF - GDB Enhanced Features. Retrieved June 25, 2026, from https://hugsy.github.io/gef/

George, G., Kotey, J., Ripley, M., Sultana, K. Z., & Codabux, Z. (2021). A Preliminary Study on Common Programming Mistakes that Lead to Buffer Overflow Vulnerability. *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, 1375–1380. https://doi.org/10.1109/COMPSAC51774.2021.00194

Gherghe, A. L., & Tudose, C. (2025). *A Novel Framework for Evaluating Application Performance in Distributed Systems*. Computer Science and Mathematics. https://doi.org/10.20944/preprints202511.0602.v1

HackerArise. (2023, November 26). Reverse Engineering Malware, Part 03: IDA Pro Introduction [Malware Analysis]. *HackerArise*. https://hackers-arise.com/reverse-engineering-malware-part-3-ida-pro-introduction/

Hadi, M., Azizah, N., & Jaya, F. (2025). Sistem Informasi Perpustakaan Berbasis Website Menggunakan Metode Extreme Programming Di SDI Nurul Manshur. *Jurnal Teknik Mesin, Elektro dan Ilmu Komputer*, *5*(3), 113–132. https://doi.org/10.55606/teknik.v5i3.7776

Hanaputra, R. R., Andzani, N. S., Bintara, M. A., & Rosadi, A. (2023). Implementasi Penggunaan Pwntools Guna Mempermudah Eksploitasi Buffer pada Ubuntu 32-Bit. *Journal on Education*, *5*(3), 5700–5706.

Imtiaz, N., & Williams, L. (2021). *Memory Error Detection in Security Testing* (arXiv:2104.04385). arXiv. https://doi.org/10.48550/arXiv.2104.04385

Kali Linux Documentation. (n.d.-a). *Netcat*. Netcat. Retrieved June 25, 2026, from https://www.kali.org/tools/netcat/

Kali Linux Documentation. (n.d.-b). *Socat*. Socat. Retrieved June 25, 2026, from https://www.kali.org/tools/socat/

Karim, M. M. (2015). *Source Code based Buffer Overflow Detection Technology* [Northwestern Polytechnical University]. https://doi.org/10.13140/RG.2.1.3837.9124

Kim, K., Kim, J., & Lee, S. (2024). StackGuard+: Interoperable alternative to canary-based protection of stack smashing. *Electronics Letters*, *60*(19), e13310. https://doi.org/10.1049/ell2.13310

Kyi, W., Koide, H., & Sakurai, K. (2019). Analyzing the Effect of Moving Target Defense for a Web System. *International Journal of Networking and Computing*, *9*(2), 188–200. https://doi.org/10.15803/ijnc.9.2_188

Lam, R. (2024, April 10). *Introducing an OpenTelemetry Collector distribution with built-in Prometheus pipelines: Grafana Alloy*. https://grafana.com/blog/Grafana Alloy-opentelemetry-collector-with-prometheus-pipelines/

Liu, C., Wu, Y.-J., Wu, J.-Z., & Zhao, C. (2023). A buffer overflow detection and defense method based on RISC-V instruction set extension. *Cybersecurity*, *6*(1), 45. https://doi.org/10.1186/s42400-023-00164-x

Lu, H. J., Matz, M., Girkar, M., Hubicka, J., Jaeger, A., & Mitchell, M. (2021, September 28). *System V Application Binary Interface AMD64 Architecture Processor Supplement (With LP64 and ILP32 Programming Models) Version 1.0* [Technical Specification]. https://cs61.seas.harvard.edu/site/2025/pdf/x86-64-abi-20210928.pdf

Machado, A. (2024a, October 28). *An In-Depth Exploration of GCC: History, Variants, and Internal Architecture* [Technology]. https://machaddr.substack.com/p/an-in-depth-exploration-of-gcc-history

Machado, A. (2024b, November 5). *The Evolution of the GNU C Library: History, Key Contributors, and Impact on the Linux Ecosystem* [Technology]. https://machaddr.substack.com/p/the-evolution-of-the-gnu-c-library

Malabay. (2016). PEMANFAATAN FLOWCHART UNTUK KEBUTUHAN DESKRIPSI PROSES BISNIS. *JURNAL ILMU KOMPUTER (JIK)*, *12*(1), 21–26. https://doi.org/10.47007/komp.v12i1.1579

Marco-Gisbert, H., & Ripoll, I. R. (2019). Address Space Layout Randomization Next Generation. *Applied Sciences*, *9*(14), 2928. https://doi.org/10.3390/app9142928

Maulana, M. H., Ismail, S. J. I., & Rizal, M. F. (2023). Membangun Server CTF (Capture the Flag) Untuk Komunitas Security Di Telkom University. *e-Proceeding of Applied Science*, *9*(5), 2332–2338.

Miller. (2019, June 16). A proactive approach to more secure code. *Security Research and Defense*. https://www.microsoft.com/en-us/msrc/blog/2019/07/a-proactive-approach-to-more-secure-code

MITRE. (2025, December 15). *2025 CWE Top 25 Most Dangerous Software Weaknesses* [Technology]. MITRE Organization. Common Weakness Enumeration. https://cwe.mitre.org/top25/archive/2025/2025_cwe_top25.html#top25list

National Security Agency USA. (2022, November 10). Software Memory Safety. *Cybersecurity Information Sheet*. https://www.nsa.gov/Press-Room/News-Highlights/Article/Article/3215760/nsa-releases-guidance-on-how-to-protect-against-software-memory-safety-issues/

OffSec Tools. (n.d.). *Pwntools*. Retrieved June 25, 2026, from https://offsec.tools/tool/pwntools

Panter, S. K., & Eisty, N. U. (2024). Rusty Linux: Advances in Rust for Linux Kernel Development. *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 496–502. https://doi.org/10.1145/3674805.3690756

Park, J., Kim, J., B. Gupta, B., & Park, N. (2021). Network Log-Based SSH Brute-Force Attack Detection Model. *Computers, Materials & Continua*, *68*(1), 887–901. https://doi.org/10.32604/cmc.2021.015172

Peppas, N. (n.d.). How to Install and Configure Fail2ban on Ubuntu Server 16.04 [Technology]. *Liquid Web by Nexcess*. Retrieved June 25, 2026, from https://www.liquidweb.com/blog/install-configure-fail2ban-ubuntu-server-16-04/

Pereira, P. R., & Sanches, J. (2025). Network and Systems Monitoring with Prometheus and Grafana. *Proceedings of 20th Iberian Conference on Information Systems and Technologies (CISTI 2025) - Volume 1*, *1716 LNNS*, 367–378. https://doi.org/10.1007/978-3-032-10929-3_32

Petrean, F., & Coleşa, A. (2024). PwnMaster: Automatic Buffer Overflow and Format String Vulnerability Detection and Exploitation. *2024 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, 1–5. https://doi.org/10.1109/AQTR61889.2024.10554220

Rahman, R., Nurfaida, & Rusmin, M. H. (2026). ANALISIS KEAMANAN FILE SERVER BERBASIS LINUX TERHADAP AKSES TIDAK SAH. *Jurnal Riset Sistem Informasi*, *3*(2), 58–62. https://doi.org/10.69714/38pr8578

Rasyid, A. H., & Santoso, J. D. (2026). Identifikasi Kerentanan Perangkat Lunak Pengolah Dokumen Berbasis Binary menggunakan Metode Whitebox Fuzzing Afl++. *Jurnal Multidisiplin Bhatara*, *3*(1). https://doi.org/https://doi.org/10.59095/jmb.v3i1.265

Ridho, M., Hafizh, A., Dani, I., & Ariyadi, T. (2025). Peningkatan Keamanan SSH Server Berbasis Linux melalui Implementasi Fail2Ban dan Uji Serangan Brute Force. *JURNAL PENELITIAN MULTIDISIPLIN BANGSA*, *1*(12), 2206–2214. https://doi.org/10.59837/jpnmb.v1i12.431

Rieger, G. (n.d.). Socat Man Pages. In *Linux manual pages*. Retrieved https://man7.org/linux/man-pages/man1/socat.1.html

Rohleder, R. (2019). Hands-On Ghidra—A Tutorial about the Software Reverse Engineering Framework. *Proceedings of the 3rd ACM Workshop on Software Protection*, 77–78. https://doi.org/10.1145/3338503.3357725

Sahara, A. D., Sapri, S., & Akbar, A. A. (2024). The Design And Implementation Of Computer Network Monitoring And Security System Using Linux Ubuntu Server. *Jurnal Media Computer Science*, *3*(1), 1–16. https://doi.org/10.37676/jmcs.v3i1.5328

Seyedhamed, G., Tapti, P., Shachee, M., & Michalis, P. (2020). Temporal System Call Specialization for Attack Surface Reduction. *29th USENIX Security Symposium*, 1749–1766. https://www.usenix.org/system/files/sec20-ghavamnia.pdf

Shao, M., Chen, B., Jancheska, S., Dolan-Gavitt, B., Garg, S., Karri, R., & Shafique, M. (2024). *An Empirical Evaluation of LLMs for Solving Offensive Security Challenges* (arXiv:2402.11814). arXiv. https://doi.org/10.48550/arXiv.2402.11814

Sharma, I. (2026, February 26). What is C Programming? Introduction, Features, and Applications [PROGRAMMING LANGUAGES]. *C Programming Introduction*. https://www.guvi.in/blog/what-is-c-programming/

Sublime Text. (n.d.). *Text Editing, Done Right*. Sublime Text. Retrieved https://www.sublimetext.com/

Ubuntu Handbook. (2022, April 21). Ubuntu 22.04 LTS Available to Download, See What's New! [News & Tutorial]. *Ubuntu Handbook*. https://ubuntuhandbook.org/index.php/2022/04/ubuntu-22-04-lts-available/

VM, S. (2024, February 12). *Log Monitoring | Grafana Loki Installation*. https://www.skynats.com/blog/log-monitoring-Grafana Loki-installation/

Xie, S., Zhang, C., Zhou, T., Liu, J., Hong, X., Li, Q., & Peng, X. (2026). *LogCopilot: Automating Log Aggregation Analysis through Large Language Models* (arXiv:2606.17094). arXiv. https://doi.org/10.48550/arXiv.2606.17094

Xie, T., Zhang, Y., Li, J., Liu, H., & Gu, D. (2016). New Exploit Methods against Ptmalloc of GLIBC. *2016 IEEE Trustcom/BigDataSE/ISPA*, 646–653. https://doi.org/10.1109/TrustCom.2016.0121

Xu, S., Jiang, Z., Wang, Y., & Xie, P. (2024). FormatAEG: A framework for bypassing ASLR defense and automated exploitation of format string vulnerability. *The Computer Journal*, *67*(11), 3056–3066. https://doi.org/10.1093/comjnl/bxae069

Xu, S., & Wang, Y. (2022). BofAEG: Automated Stack Buffer Overflow Vulnerability Detection and Exploit Generation Based on Symbolic Execution and Dynamic Analysis. *Security and Communication Networks*, *2022*, 1–9. https://doi.org/10.1155/2022/1251987

Yanto, R. (2024). Implementasi Ubuntu Web Server dengan Service Nginx pada STMIK Dharmapala Riau. *Journal of Enterpreneurship & Technopreneur (JoET)*, *1*(1), 26–38.

Yoon, H., & Lee, M. (2022). SGXDump: A Repeatable Code-Reuse Attack for Extracting SGX Enclave Memory. *Applied Sciences*, *12*(15), 7655. https://doi.org/10.3390/app12157655

Zhang, H., Wang, J., Wang, Y., Li, M., Song, J., & Liu, Z. (2023). ICVTest: A Practical Black-Box Penetration Testing Framework for Evaluating Cybersecurity of Intelligent Connected Vehicles. *Applied Sciences*, *14*(1), 204. https://doi.org/10.3390/app14010204

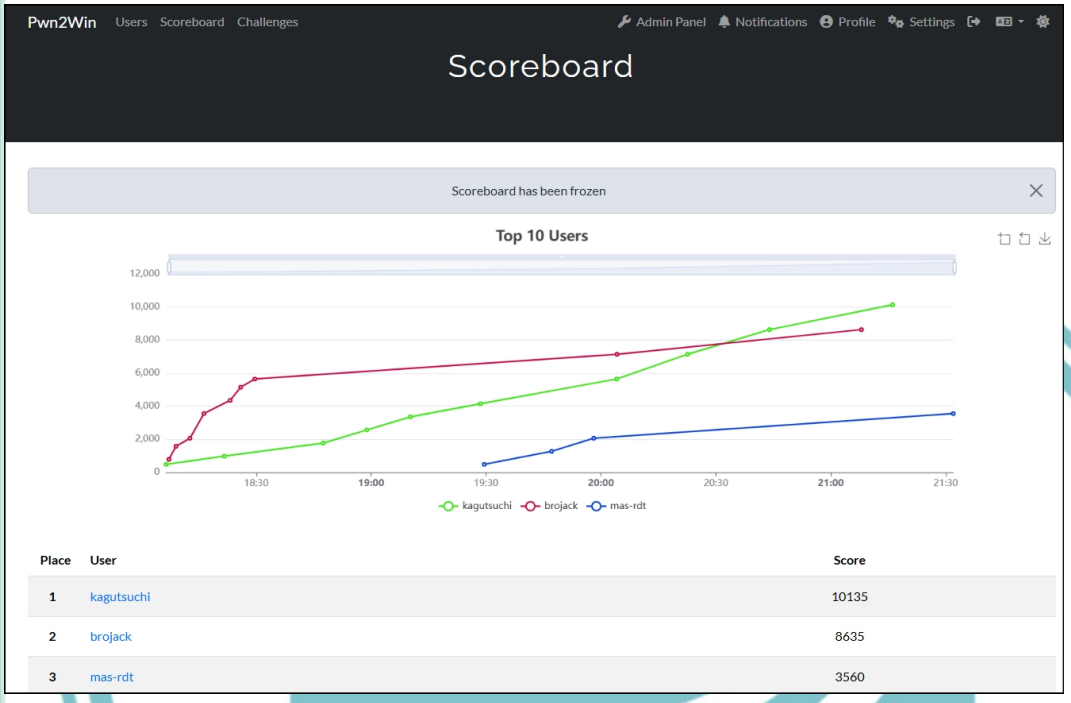
DAFTAR RIWAYAT HIDUP PENULIS

Wahyu Priambodo

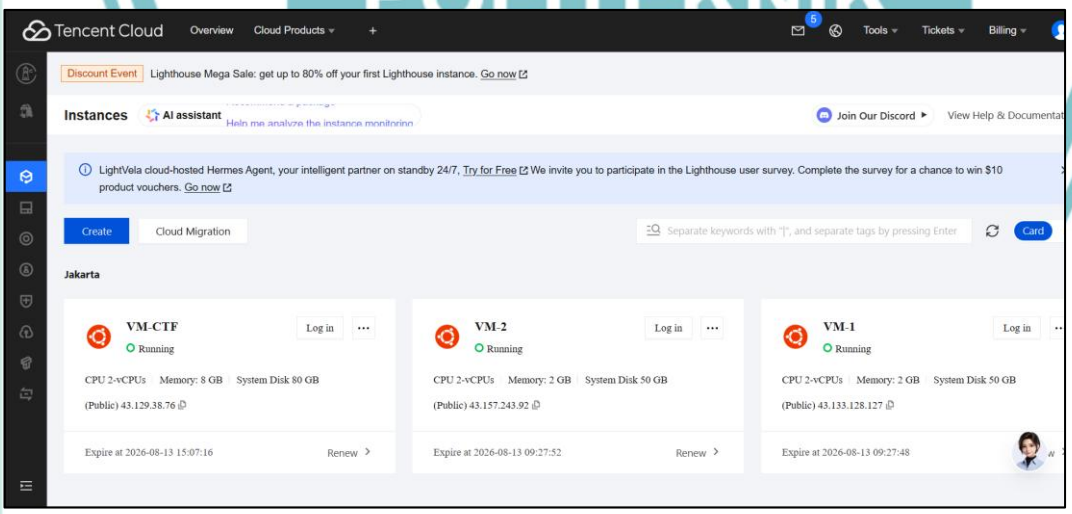
Lahir pada tahun 2003 dan merupakan anak ke-2 dari 2 bersaudara. Penulis menyelesaikan pendidikan dasar di SD Negeri 01 Bukit Duri, pendidikan menengah pertama di SMP Negeri 26 Jakarta, dan pendidikan menengah atas di SMK Negeri 1 Jakarta dengan jurusan Sistem Informatika, Jaringan, dan Aplikasi (SIJA) selama 4 tahun. Pada tahun 2022, penulis melanjutkan pendidikan di Politeknik Negeri Jakarta (PNJ), Jurusan Teknik Informatika dan Komputer dan Program Studi Teknik Multimedia dan Jaringan. Penulis memiliki ketertarikan untuk mendalami ilmu komputer, khususnya di bidang jaringan komputer dan keamanan siber.

LAMPIRAN

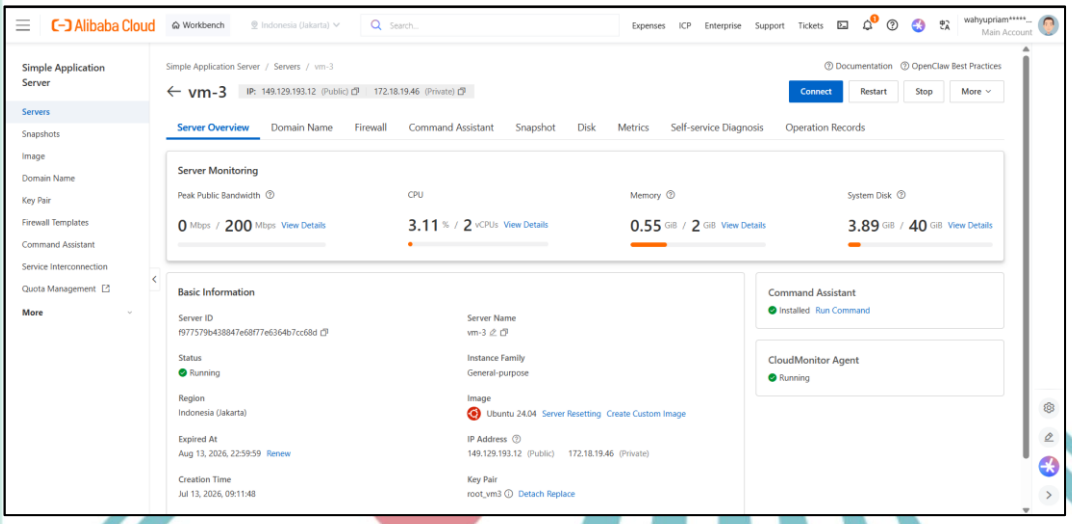
Lampiran 1. Bukti Pelaksanaan Ajang CTF Internal untuk Pengambilan Data



Lampiran 2. Bukti VM-CTF, VM-1, dan VM-2 yang Disewa dari *Provider – Tencent Cloud*



Lampiran 3. Bukti VM-3 yang Disewa dari *Provider – Alibaba Cloud*



Lampiran 4. Bukti Grafana Berhasil Memvisualisasikan Sumber Daya Server Lain Secara *Real-time*

Lampiran 5. Akses Platform CTF Melalui Internet

Ajang CTF yang telah dilaksanakan pada tanggal 15 Juli 2026 dari mulai pukul 17:00 WIB hingga 23:59 WIB diikuti oleh 3 peserta yang beraksi selayaknya penyerang. Di mana, CTF diselenggarakan menggunakan platform CTFd yang tersedia secara gratis dan menyediakan 10 *file binary* (objek penelitian) kepada penyerang untuk dieksploitasi. Adapun platform CTF dapat diakses melalui server utama yang sudah dikaitkan dengan nama domain dan koneksi HTTPS. Akses platform CTF melalui nama *domain* berikut: https://ctf.pwngame.site

Lampiran 4. Repositori Lab Penelitian (GitHub)

File-file objek penelitian, skrip eksploitasi, *service*, dan konfigurasi terhadap komponen-komponen yang diperlukan sudah penulis sertakan pada repositori GitHub. Repositori GitHub penulis untuk lab penelitian ini tertera pada tautan berikut: https://github.com/whyuhurtz/research-lab/