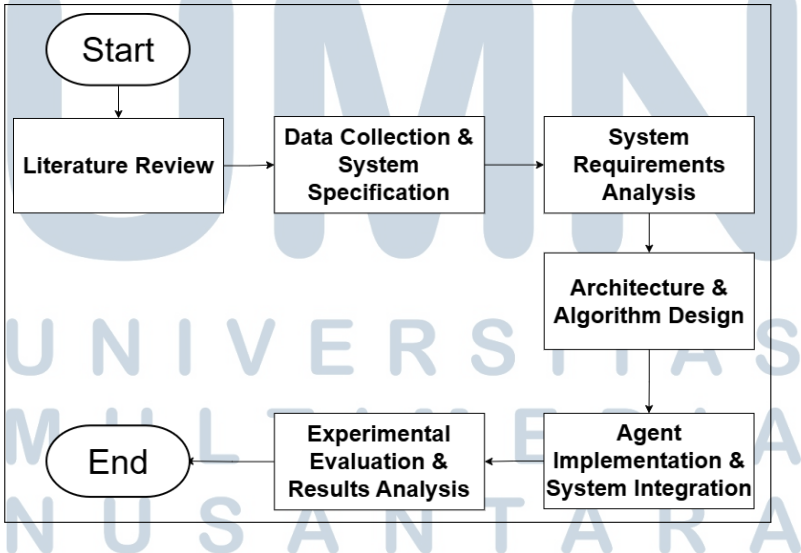
BAB 3
METODOLOGI PENELITIAN

Bab ini membahas metodologi penelitian untuk pengembangan BYEBAC sebagai AI *Agent* yang melakukan pengujian *endpoint back-end* secara otomatis pada aplikasi ESS untuk deteksi kerentanan *Broken Access Control* (BAC). Metodologi penelitian mengikuti pendekatan *Design Science Research* dengan tahapan berupa studi literatur, pengumpulan data dan analisis kebutuhan, perancangan arsitektur sistem, implementasi, pengujian, dan evaluasi.

3.1 Metode Penelitian

Penelitian ini menggunakan pendekatan *Design Science Research* untuk mengembangkan dan mengevaluasi BYEBAC sebagai alat bantu pengujian keamanan berbasis AI *Agent*. Metode penelitian dirancang untuk menjawab dua tujuan utama yaitu merancang sistem BYEBAC dengan fitur *automated test planning*, *resource ID discovery*, dan *multi-role testing*, serta membandingkan performa pengujian otomatis terhadap pendekatan manual dari aspek kecepatan, *coverage*, dan akurasi deteksi. Seperti ditunjukkan pada Gambar 3.1, alur penelitian terdiri dari enam tahap utama yang saling terkait:



Gambar 3.1. Flowchart penelitian

3.1.1 Studi Literatur

Tahap awal dimulai dengan studi literatur untuk mengidentifikasi kontribusi AI *Agent* dalam pengujian keamanan, pola serangan BAC (IDOR dan BOLA), serta kesenjangan penelitian terkait otomatisasi pengujian *endpoint back-end* berbasis OAS, RBAC dan *policy rules*. Literatur yang dikaji mencakup publikasi tahun 2018–2025 untuk memastikan relevansi dengan perkembangan terkini.

3.2 Pengumpulan Data dan Spesifikasi Sistem

Pada tahap ini, data dikumpulkan sebagai bahan pengujian BYEBAC sebagai alat bantu berbasis AI *Agent*. Spesifikasi sistem menjelaskan arsitektur dan fitur alat bantu pengujian keamanan otomatis berbasis AI *Agent*.

3.2.1 Pengumpulan Data

Berdasarkan *gap research* yang ditemukan, tahap pengumpulan data dilakukan dengan mengekstrak spesifikasi dari aplikasi ESS PT. XYZ. Aplikasi ESS memiliki total 56 *endpoint*, dengan 28 *endpoint* dikategorikan sebagai *critical* karena mengandung data konfidensial (seperti informasi pengguna) dan parameter identitas (*id*, *userId*, *resourceId*) yang berpotensi rentan terhadap serangan BAC. Data yang dikumpulkan meliputi:

- *OpenAPI Specification* (OAS) dalam format *JavaScript Object Notation* (JSON) yang berisi dokumentasi 28 *endpoint* dengan detail *path*, metode *HyperText Transfer Protocol* (HTTP), parameter, *request body*, dan *response schema*
- Data *permission* yang terdiri dari 5 *permission*: *update_data_personal*, *request_history_and_status*, *attachment*, *rbac_admin*, dan *manage_consent*
- Data *role* yang terdiri dari 2 *role* yaitu *Admin* dan *Employee*
- Kredensial pengujian untuk 3 akun uji (1 *admin*, 2 *employee*) yang merepresentasikan skenario *self-access* dan *other-access* untuk pengujian *Insecure Direct Object Reference* (IDOR) dan *Broken Object Level Authorization* (BOLA).

Untuk pengujian IDOR, digunakan *variant* "*self*" untuk akses normal dan "*other*" untuk pengujian akses horizontal terhadap sumber daya milik pengguna lain. Untuk pengujian BOLA sebagai *vertical escalation*, digunakan *variant* "*escalate*" dengan opsi tambahan "*as_role*" untuk menguji potensi eskalasi hak akses berdasarkan *role*.

3.2.2 Spesifikasi Sistem

BYEBAC dikembangkan menggunakan Python 3.12.6 dengan Gemini 3.0 *Pro Preview* sebagai *reasoning engine* yang membaca OAS, matriks RBAC, dan *policy rules* untuk menjalankan pengujian otomatis. Pengujian dilakukan pada lingkungan *pre-production* dan lokal dengan dependensi utama berupa *google-generativeai, requests,* dan *jinja2*.

3.3 Analisis Kebutuhan Sistem

Analisis kebutuhan sistem mengidentifikasi empat kebutuhan fungsional (*functional requirements*) dan lima kebutuhan non-fungsional (*non-functional requirements*) yang harus dipenuhi oleh BYEBAC untuk melakukan pengujian keamanan otomatis.

A Kebutuhan Fungsional

Kebutuhan fungsional berupa perilaku dan kemampuan utama yang wajib dimiliki BYEBAC agar mampu menjalankan alur pengujian keamanan secara *end-to-end*. Pada konteks penelitian ini, fokus bukan hanya pada eksekusi permintaan HTTP semata, tetapi juga pada kemampuan menemukan *resource IDs*, merencanakan skenario uji berdasarkan OAS dan matriks RBAC, mengklasifikasikan hasil respons secara konservatif, serta mengelola artefak uji secara terstruktur. Penjelasan kebutuhan fungsional tersebut adalah sebagai berikut:

- **FR-1: Auto-discovery Resource IDs**
  BYEBAC harus mampu menemukan *resource IDs* secara otomatis karena sebagian besar pengujian keamanan BAC memerlukan ID yang valid untuk skenario IDOR dan BOLA. Pendekatan *hardcoding IDs* tidak bersifat *scalable*, sehingga sistem menggunakan algoritma yang mengidentifikasi *detail endpoints* dengan *placeholder id*, memetakan *list endpoints* terkait,

mengambil ID dari *response*, dan menyimpannya dalam memori untuk pengisian *placeholder*.

- **FR-2: Intelligent Test Planning**
  BYEBAC harus melakukan *intelligent test planning* dengan memanfaatkan *Large Language Model* (LLM) dalam menganalisis OAS dan matriks RBAC. Proses *planning* menghasilkan *test cases* yang mencakup skenario BASELINE, IDOR, dan BOLA secara otomatis melalui tahap *discovery* dan *analysis*.

- **FR-3: Conservative Classification**
  BYEBAC harus menerapkan klasifikasi ketat untuk meminimalkan *False Positives*. Logika klasifikasi yang diterapkan berupa status 5xx termasuk sebagai ERROR, 404 sebagai NOT_FOUND, dan 400/409 sebagai *True Negative*. Hanya kasus dengan status aktual 200/201 yang seharusnya 401/403 dikelompokkan sebagai *False Negative*, sedangkan kasus dengan status aktual 401/403/404 yang seharusnya 200 dikategorikan sebagai *False Positive*.

- **FR-4: Organized Artifact Storage**
  Artefak disimpan secara hierarkis pada *artifacts/*{*role*}/{*BAC_type*}/{*target*}/{*filename*}, dengan *BAC_type* salah satu dari kategori uji coba yaitu BASELINE, IDOR, BOLA dan AUTH atau DISCOVERY. Parameter {*target*} bersifat opsional untuk spesifikasi peran tertentu (misalnya: *to_admin*, *to_employee*), dan *filename* mengikuti pola {*timestamp*}_{*METHOD*}_{*endpoint*}_{*resource_id*}*.json*. Struktur ini memudahkan untuk menganalisis temuan secara cepat, serta memungkinkan analisis komparatif respons per skenario BAC.

B Kebutuhan Non-Fungsional

Kebutuhan non-fungsional mendefinisikan karakteristik kualitas sistem yang harus dipenuhi agar BYEBAC dapat dijalankan dengan baik pada berbagai lingkungan. Dalam konteks penelitian ini, kebutuhan tersebut menekankan aspek kinerja eksekusi, perlindungan data sensitif, fleksibilitas konfigurasi lingkungan penerapan, kemudahan pengembangan dan perluasan fungsi, serta kemampuan pelacakan dan audit terhadap setiap pengujian yang dilakukan. Rincian kebutuhan non-fungsional sebagai berikut:

- **NFR-1: Performance**
  BYEBAC harus mendukung eksekusi konkuren dengan tingkat konkurensi 2 (*ThreadPoolExecutor*), melakukan dua tes uji sekaligus untuk mempercepat waktu eksekusi *testing* dibandingkan dengan eksekusi *sequential*.

- **NFR-2: Privacy & Security**
  BYEBAC harus menyembunyikan data sensitif dalam artefak secara otomatis. Token otorisasi diubah menjadi format "*Bearer eyJhbG...iOiJ*", dengan hanya 6 karakter pertama dan 4 karakter terakhir yang ditampilkan.

- **NFR-3: Configurability**
  BYEBAC harus mendukung konfigurasi berbasis *YAML Ain't Markup Language* (YAML) untuk memungkinkan penyesuaian tanpa perubahan kode, sehingga dapat diterapkan ke berbagai lingkungan (pengembangan, *staging*, produksi) dengan konfigurasi yang berbeda.

- **NFR-4: Extensibility**
  BYEBAC harus memiliki arsitektur modular untuk mendukung mutasi kustom (*custom mutations*), sehingga mudah diubah atau diperluas sesuai kebutuhan pengujian di masa depan.

- **NFR-5: *Auditability***
  BYEBAC harus menyediakan penyimpanan artefak yang komprehensif, memungkinkan audit dan pelacakan penuh terhadap setiap tes yang dilakukan, termasuk status, hasil, dan metadata terkait.

Sembilan kebutuhan ini diimplementasikan dalam basis kode BYEBAC dengan kriteria penerimaan yang jelas dan dapat divalidasi melalui kasus uji.

3.4 Perancangan Arsitektur dan Algoritma Sistem

Pada bagian ini dijelaskan perancangan arsitektur BYEBAC sebagai alat bantu berbasis AI *Agent* untuk pengujian otomatis kerentanan BAC. Arsitektur sistem terdiri dari beberapa modul utama berupa modul AI *Agent*, pengolahan data pengujian, dan pengujian otomatis. Sistem menggunakan algoritma untuk melakukan pemetaan *endpoint*, analisis peran pengguna, serta pelaksanaan pengujian otomatis pada setiap *endpoint*.

3.4.1 Perancangan Arsitektur Sistem

BYEBAC dibangun dengan desain modular dan prinsip *separation of concerns* sehingga memisahkan fungsi-fungsi utama sistem menjadi komponen yang independen namun tetap terintegrasi secara efisien. Sistem otomatisasi pengujian *endpoint* oleh BYEBAC dirancang dengan arsitektur yang terdiri dari enam komponen utama yang saling berinteraksi:

- **Orchestrator**: Komponen inti sebagai pengatur utama yang menangani seluruh proses pengujian mulai dari persiapan, eksekusi, hingga pembuatan laporan dan memastikan semua proses saling berurutan dan terkoordinasi satu sama lain.
- **HttpClient**: Mengatur permintaan HTTP dengan mekanisme *retry* otomatis (*exponential backoff* dan *jitter*) untuk mencegah pembebanan server, memanfaatkan sesi untuk *connection reuse*, dan menyimpan respons sebagai artefak agar komunikasi lebih cepat dan mudah dilacak.
- **AuthManager**: Mengelola alur autentikasi dengan *token caching* untuk menghindari *login* berulang, mendukung beberapa metode autentikasi, dan mengatur kredensial melalui berkas `.env` sehingga proses pengujian tetap aman dan efisien.
- **Memory**: Menyimpan data pengujian dalam memori menggunakan *dataclass* untuk menjaga *type safety*, mencakup daftar *test case*, hasil eksekusi, dan *resource IDs* per peran sehingga status pengujian dapat dikelola dengan efisien.
- **Evaluators**: Komponen evaluasi *stateless* yang menerima hasil dan kebijakan RBAC, mengklasifikasikan respons ke dalam *confusion matrix*, dan menghitung metrik untuk memastikan kesesuaian dengan kebijakan.
- **Reporters**: Menghasilkan laporan dalam format JSON untuk pemrosesan mesin dan *Markdown* untuk dokumentasi, termasuk pemetaan artefak dan ringkasan berbasis LLM.

Pola interaksi antar komponen mengikuti prinsip *dependency injection*, di mana *Orchestrator* membuat instansi semua dependensi dan meneruskannya ke fungsi lainnya. Hal ini memudahkan pengujian dan pemeliharaan sistem.

3.4.2 Perancangan Algoritma Sistem

Perancangan algoritma disusun sebagai rangkaian komponen yang saling melengkapi untuk menguji kontrol akses, meliputi perencana uji berbasis LLM, penemuan otomatis *resource* ID, deteksi IDOR berbasis pola, dan deteksi BOLA berbasis peran.

1. **LLM-Based Intelligent Test Planning**
   Pendekatan ini memungkinkan LLM menganalisis OAS dan matriks RBAC untuk menghasilkan daftar kasus uji yang terstruktur dengan prioritas. LLM menghasilkan *test planner* yang siap dieksekusi secara otomatis dengan skenario BASELINE, IDOR, dan BOLA berdasarkan analisis kebijakan akses dari matriks RBAC dan *policy rules*.

2. **Auto-Discovery Resource IDs**
   Algoritma ini melakukan otomatisasi persiapan *resource IDs* untuk setiap *endpoint* yang memiliki parameter identitas. Algoritma menurunkan pasangan *endpoint* detail dari *endpoint* daftar berdasarkan OAS (misalnya /user/{id} $\rightarrow$ /users), mengambil contoh entitas dari *response*, lalu mengekstraksi atribut identitas dengan prioritas ”id”, kemudian ”_id”, dan selanjutnya kunci yang memuat token ”id”. Identitas yang diperoleh dipetakan per peran dan dinormalisasi ke beberapa alias agar *request* konsisten pada tahap uji berikutnya, sekaligus mengurangi ketergantungan pada ID yang ditetapkan secara tetap (*hard-coded*).

3. **Rule-Based Pattern Detection**
   Algoritma ini mendeteksi pelanggaran otorisasi horizontal (IDOR) pada *endpoint* yang merujuk pada objek tertentu. Setiap sumber daya harus membatasi akses di mana akses oleh pemilik diperbolehkan, sedangkan akses oleh pengguna lain ditolak. Detektor mengenali pola *endpoint* yang memiliki parameter identitas, menetapkan ekspektasi untuk skenario *self-access* dan *other-access*, lalu membandingkan hasil aktual dengan ekspektasi. Jika akses lintas pengguna terhadap sumber daya milik pihak lain tetap berhasil, kejadian tersebut dikategorikan sebagai kerentanan IDOR.

4. **Policy-Based Algorithm**
   Algoritma ini mendeteksi eskalasi hak akses vertikal (BOLA) pada *endpoint* yang khusus untuk *admin*. *Endpoint* yang tercantum dalam daftar *critical*

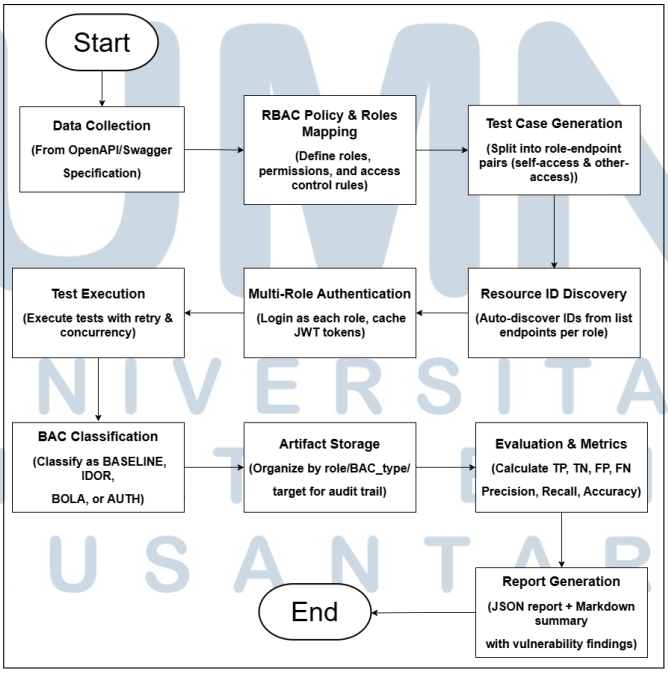
deny hanya boleh diakses oleh peran dengan izin yang sesuai (misalnya *rbac_admin*). Detektor memeriksa apakah *endpoint* termasuk *critical deny* dan apakah peran pengguna memiliki izin tersebut. Jika *endpoint admin* diakses oleh *employee* dan sistem mengembalikan kode status 200 (seharusnya 403/404), kejadian tersebut diklasifikasikan sebagai *False Negative* dan kerentanan BOLA.

3.5 Implementasi Agen dan Integrasi Sistem

Bagian ini menjelaskan implementasi BYEBAC berbasis LLM untuk otomatisasi pengujian keamanan *endpoint*. Integrasi sistem meliputi pemanggilan HTTP menggunakan pustaka requests, integrasi LLM, serta eksekusi konkuren menggunakan *ThreadPoolExecutor*.

3.5.1 Implementasi Agen

BYEBAC dikembangkan dengan sebuah orkestrator berbasis LLM yang mengkoordinasikan *agent* untuk mempelajari data yang disediakan dan mengeksekusi pengujian *endpoint* secara otomatis. Alur pengujian otomatis oleh BYEBAC ditunjukkan pada Gambar 3.2.



Gambar 3.2. Alur proses pengumpulan dan implementasi BYEBAC

1. **Data Collection**

   Tahap awal dengan pengumpulan data dari aplikasi ESS berupa OAS yang mencakup daftar *endpoint*, *HTTP method*, parameter, dan struktur respons sebagai dasar penyusunan skenario uji. Data tambahan berupa daftar dua *role* (*admin* dan *employee*) serta lima *permission*.

2. **RBAC Policy & Roles Mapping**

   Sistem memetakan matriks RBAC dengan menggabungkan daftar *role* dan *permission* yang telah dikumpulkan, kemudian dipetakan ke *endpoint* dalam OAS untuk membentuk *policy rules* yang mencakup *allowed_endpoints* dan *critical deny*. Hasil akhir berupa matriks RBAC lengkap dan daftar *endpoint* yang dapat diakses sesuai *role* dan *permission*.

3. **Test Case Generation**

   Berdasarkan OAS, matriks RBAC, dan *policy rules* dari tahap sebelumnya, AI *Agent* mempelajari dan menghasilkan serangkaian *test case* dengan prioritas. Skenario dibagi menjadi *self-access* (akses terhadap *resource* milik sendiri) dan *other-access* (akses terhadap *resource* milik pengguna lain) untuk mendeteksi kerentanan IDOR dan BOLA. AI *Agent* memprioritaskan *endpoint* berdasarkan tingkat risiko kerentanan dan urutan eksekusi yang optimal.

4. **Resource ID Discovery**

   Untuk *endpoint* dengan parameter ID pada *path*, sistem memperoleh nilai ID valid melalui *auto-discovery*. *Agent* memanggil *list endpoints* (misalnya `GET /users`) per *role*, mengekstrak ID dari respons, dan menyimpannya sebagai referensi untuk skenario *self-access* dan *other-access*.

5. **Multi-Role Authentication**

   BYEBAC melakukan autentikasi untuk tiga akun uji yang telah disiapkan mewakili kedua *role* (*admin* dan *employee*). BYEBAC melakukan *login* pada masing-masing akun, mengambil *JSON Web Token* (JWT), dan menyimpannya (*cache*) agar dapat digunakan kembali pada tahap eksekusi tanpa perlu *login* berulang.

6. **Test Execution**

   Seluruh *test case* dieksekusi terhadap *back-end* aplikasi ESS. BYEBAC mengirim permintaan HTTP sesuai *test case* dengan mekanisme *retry* dan

eksekusi konkuren untuk mempercepat proses pengujian. Setiap pasangan *request–response* dicatat sebagai artefak.

7. **BAC Classification**
   Hasil pengujian diklasifikasikan ke dalam kategori BASELINE, IDOR, BOLA, atau AUTH menggunakan *confusion matrix* dengan kategori TP, TN, FP, dan FN sebagaimana dijelaskan di Bab 2. Tahap ini menentukan apakah respons yang diterima merupakan kerentanan BAC atau bukan berdasarkan kode status HTTP dan *policy rules*.

8. **Artifact Storage**
   Semua artefak pengujian (*request*, *response*, label BAC, dan metadata) disimpan terstruktur dalam format JSON dan dikelompokkan berdasarkan *role*, tipe BAC, dan target *endpoint* untuk kebutuhan *audit trail*, analisis lanjutan, dan replikasi pengujian.

9. **Evaluation & Metrics**
   BYEBAC menghitung metrik kinerja deteksi kerentanan berupa jumlah TP, TN, FP, dan FN. Dari nilai tersebut dihitung *precision*, *recall*, *F1-score*, dan *accuracy* untuk menilai kemampuan BYEBAC dalam mengidentifikasi pelanggaran BAC.

10. **Report Generation**
    Tahap akhir pembuatan laporan dalam format JSON dan ringkasan *Markdown* yang dilengkapi *AI security summary*. Laporan memuat metrik evaluasi, *confusion matrix*, daftar *endpoint* yang terdeteksi rentan, dan temuan kerentanan sebagai *output* utama untuk analisis hasil penelitian.

3.5.2 Integrasi Sistem

Pada bagian ini, dijelaskan bagaimana berbagai komponen dari BYEBAC diintegrasikan untuk memastikan kelancaran operasi dan komunikasi antar *layer*. Integrasi BYEBAC memiliki alur *plan, discover, execute, observe, reflect*, dan *report* yang dikoordinasikan oleh *AgentOrchestrator*.

- **Inisialisasi**: Dimulai dengan mempelajari tiga sumber data yaitu OAS, matriks RBAC dan *policy rules*. Selain itu menyiapkan *prompt* untuk BYEBAC dalam melakukan pengujian.

- **Plan**: Menyusun dan memprioritaskan kasus uji dari OAS dan matriks RBAC. Setiap *endpoint* divalidasi dengan OAS dan diberi skor prioritas.
- **Discover**: Menerapkan *cache-first* di mana *resource ID* diambil dari *list endpoints* dan disimpan ke memori dengan alias yang seragam.
- **Execute**: Eksekusi paralel ringan (*ThreadPoolExecutor*) dan *retry* dengan *exponential backoff* dan *jitter*. Sistem melakukan *masking* data sensitif dan penyimpanan artefak terstruktur per peran/tipe uji.
- **Observe & Classify**: Bagian ini membuat ekspektasi (IDOR dan BOLA). *Observe* membuat metrik *precision*, *recall*, *F1-score*, dan *accuracy* dari hasil pengujian yang dilakukan. Sedangkan *Classify* memberikan label dari hasil *testing* yang dilakukan.
- **Reflect**: Temuan anomali disimpan dan dianalisis dengan LLM beserta statistik ringkas untuk menyusun saran perbaikan.
- **Report**: Bagian ini menghasilkan hasil pengujian dalam format JSON yang berisi ringkasan, *confusion matrix*, *coverage*, *time-to-detect*, dan *artifact*.

3.6 Evaluasi Eksperimental dan Analisis Hasil

Bagian ini memaparkan tujuan, lingkungan, konfigurasi pengujian, prosedur, skenario uji yang diterapkan, dan metode pengolahan data untuk mengevaluasi efektivitas BYEBAC dalam mendeteksi kerentanan BAC pada aplikasi ESS.

3.6.1 Evaluasi Eksperimental

Evaluasi eksperimental mengukur efektivitas BYEBAC dalam mendeteksi kerentanan BAC menggunakan *confusion matrix* dan *coverage analysis*. Eksperimen dilakukan pada 28 *endpoint* kritis aplikasi ESS dengan dua peran (*admin* dan *employee*) menggunakan tiga akun uji untuk skenario *self-access*, *other-access* dan *admin-only endpoints*.

Test case digenerasikan otomatis oleh LLM berdasarkan OAS, *policy rules* dan matriks RBAC, berupa BASELINE untuk memvalidasi akses normal (*employee* terhadap dirinya sendiri), IDOR untuk menguji kerentanan *horizontal*

antar pengguna, dan BOLA untuk menguji kerentanan vertikal *employee* ke *admin-only endpoints*. Setiap *test case* dieksekusi dengan mekanisme *retry*, dan respons disimpan sebagai artefak hierarkis berdasarkan *role*, tipe BAC, dan target *endpoint*. Pada Tabel 3.1 menjelaskan BYEBAC mengklasifikasikan hasil ke enam kategori berupa:

Tabel 3.1. Klasifikasi hasil pengujian keamanan otomatis

No	Code	Category	Response	Description
1	`TP`	*True Positive*	200, 201	Permintaan diizinkan sesuai kebijakan.
2	`TN`	*True Negative*	403, 404	Permintaan ditolak sesuai kebijakan.
3	`FN`	*False Negative*	200, 201	Permintaan ditolak namun diizinkan.
4	`FP`	*False Positive*	403, 404	Permintaan diizinkan namun ditolak.
5	`ERROR`	*Error*	5xx	Ketidakstabilan sistem.
6	`NOT_FOUND`	*Not Found*	404	Sumber daya tidak ditemukan.

Validasi dilakukan dengan pengujian otomatis BYEBAC sebanyak lima kali untuk mengukur konsistensi dan pengujian manual menggunakan *Postman* dengan skenario identik untuk perbandingan langsung.

3.6.2 Analisis Hasil

Analisis hasil dilakukan dengan mengklasifikasikan setiap respons berdasarkan perbandingan antara *response code* aktual dan *expected response* yang diturunkan dari kebijakan RBAC ke dalam TP, TN, FP, dan FN, kemudian dihitung metrik evaluasi seperti *Precision*, *Recall*, *F1-Score*, dan *Accuracy*. Data ini diinterpretasikan menggunakan *LLM-based triage* untuk mengidentifikasi pola serangan dan tingkat risiko, lalu divalidasi oleh peneliti melalui perbandingan dengan pengujian manual menggunakan *Postman*. Selain itu, dilakukan analisis *coverage* berdasarkan *endpoint*, *role*, jenis BAC, dan metode HTTP, serta perbandingan antara pendekatan manual dan otomatis dari sisi waktu pengujian, cakupan kombinasi *endpoint* $\times$ *role*, dan akurasi klasifikasi yang terlihat pada hasil perhitungan *confusion matrix*.