BAB II
TINJAUAN PUSTAKA

2.1 Landasan Teori

2.1.1 *Root*

Root pada perangkat *android* mengacu pada proses yang memungkinkan pengguna memperoleh akses penuh atau kontrol administratif terhadap sistem operasi perangkat mereka. Meskipun *rooting* memberikan kebebasan lebih kepada pengguna, hal ini juga meningkatkan risiko keamanan, seperti kebocoran data sensitif. Penelitian oleh (Soewito & Suwandaru, 2022) menunjukkan bahwa deteksi perangkat yang telah di *root* menjadi esensial untuk menjaga keamanan sistem, karena metode deteksi yang ada saat ini sering kali dapat di *bypass* dengan teknik seperti *Java function hooking*, yang memungkinkan aplikasi berjalan pada perangkat yang telah di *root* tanpa terdeteksi. Proses *rooting* pada umumnya melibatkan pembukaan *bootloader*, instalasi *custom recovery*, dan pemasangan aplikasi *superuser* seperti *Magisk* atau *KernelSu. Rooting* memiliki manfaat bagi pengguna yang ingin melakukan kustomisasi lebih dalam, misalnya menghapus aplikasi bawaan yang tidak diinginkan (*bloatware*), mengubah tema sistem, atau meningkatkan performa perangkat dengan memodifikasi *kernel*.

Namun, *rooting* juga memiliki risiko yang signifikan. Ketika sebuah perangkat di *root*, batasan keamanan sistem *android* menjadi longgar, dan aplikasi memiliki potensi untuk mengakses data sistem secara lebih bebas. *Malware* atau aplikasi jahat yang berjalan pada perangkat yang di *root* dapat memperoleh hak akses *superuser*, yang memungkinkan mereka memodifikasi berkas sistem atau bahkan mencuri data tanpa izin pengguna. Selain itu, *rooting* juga mengakibatkan

hilangnya garansi perangkat serta peningkatan risiko kerusakan sistem atau "*bricking*" (perangkat tidak dapat digunakan). Metode umum yang digunakan untuk mendeteksi status *root* pada perangkat *android* melibatkan beberapa pendekatan teknis, seperti:

1. Pemeriksaan berkas sistem, deteksi berkas yang biasanya ada di perangkat yang telah di *root*, seperti su atau *busybox*.
2. API keamanan, beberapa API keamanan, seperti *SafetyNet* dari Google, dapat memverifikasi status *root* dengan mendeteksi perubahan pada sistem operasi.
3. Pemeriksaan akses *superuser*, memeriksa apakah terdapat aplikasi seperti *Magisk* atau *KernelSU* yang memberikan akses *root*. Pendekatan ini berguna untuk mengidentifikasi perangkat yang di *root* dan mencegah aplikasi tertentu berjalan jika perangkat dianggap tidak aman.

2.1.2 *Flag Secure*

Flag secure adalah fitur dalam pengembangan aplikasi *android* yang digunakan untuk mencegah tindakan seperti tangkapan layar (*screenshot*) dan perekaman layar pada konten yang dianggap sensitif. Implementasi *flag secure* dalam aplikasi bertujuan untuk melindungi data digital dari reproduksi tanpa izin, seperti yang diterapkan dalam e-*library* Institut Teknologi Dirgantara Adisutjipto untuk mencegah pengguna mengambil tangkapan layar atau menyalin berkas buku digital secara ilegal (Sudaryanto dkk., 2022).

Flag secure biasanya diimplementasikan pada aplikasi yang menangani data sensitif, seperti aplikasi perbankan, aplikasi kesehatan, dan aplikasi yang

memproses data pengguna secara rahasia. Dengan adanya *flag secure*, pengembang dapat memastikan bahwa konten sensitif tidak akan bocor melalui metode tangkapan layar atau *screen recording* yang tidak diotorisasi. Hal ini penting untuk melindungi privasi pengguna, terutama di perangkat yang rentan terhadap akses tidak sah atau di *root*. Penggunaan *flag secure* memberikan beberapa manfaat penting, seperti:

1. Perlindungan data sensitif: Menjamin data pada aplikasi tidak dapat diakses oleh pihak ketiga melalui tangkapan layar.
2. Meningkatkan kepercayaan pengguna: Pengguna merasa lebih aman menggunakan aplikasi dengan *flag secure*, terutama pada aplikasi yang mengelola informasi pribadi atau data finansial.
3. Mengurangi risiko kebocoran data: Dengan mengaktifkan *flagsecure*, risiko kebocoran data melalui metode perekaman visual dapat di minimalisir. Implementasi *flag secure* ini memberikan jaminan keamanan tambahan bagi aplikasi yang memproses data penting atau sensitif.

2.1.3 *Packagename*

Menurut dokumentasi resmi di situs developer.android.com, *package name* adalah pengenal unik yang digunakan untuk mengidentifikasi aplikasi *android* anda secara universal. Nama ini diformat sebagai nama paket bergaya bahasa Java lengkap untuk aplikasi *android*. Nama tersebut dapat berisi huruf besar atau kecil, angka, dan garis bawah ('_'). Namun, setiap bagian dari nama paket harus dimulai dengan huruf. Penamaan yang tepat memastikan tidak ada dua aplikasi dengan

nama paket yang sama, sehingga mencegah konflik selama instalasi atau pembaruan aplikasi.

2.1.4 *Android*

Sistem operasi *android* adalah platform *mobile* berbasis Linux yang bersifat terbuka. Ini mencakup sistem operasi dasar, *middleware*, dan berbagai aplikasi. *android* memberikan kebebasan bagi para *developer* untuk berkreasi dan mengembangkan aplikasi mereka sendiri. Google awalnya membeli *Android* Inc., sebuah perusahaan yang mengembangkan perangkat lunak untuk ponsel pintar, dan kemudian menjadikannya platform *mobile* yang sangat adaptif(Mayrhofer dkk., 2021). Lisensi inti *android* menggunakan GNU GPLv2 (*copyleft*), yang memastikan bahwa semua perbaikan atau modifikasi juga bersifat terbuka. Sementara itu, distribusinya menggunakan lisensi *Apache Software* (ASL/*Apache2*) yang lebih fleksibel dan memungkinkan distribusi yang luas.

2.1.5 OWASP

OWASP (*Open Web Application Security Project*) adalah organisasi nirlaba global yang berfokus pada peningkatan keamanan perangkat lunak. Untuk aplikasi *mobile*, OWASP menyediakan *Mobile Application Security Verification Standard* (MASVS), sebuah kerangka kerja yang mendefinisikan standar keamanan dan dapat digunakan untuk verifikasi. (The OWASP Foundation, 2023). Standar ini mencakup berbagai area, namun yang paling relevan dengan penelitian ini adalah kategori *MASVS-V8: Resiliency Against Reverse Engineering and Tampering.* Kategori ini berisi serangkaian kontrol yang bertujuan untuk mempersulit analisis dan modifikasi aplikasi oleh pihak yang tidak berwenang. Kontrol-kontrol inilah

yang menjadi landasan dan pembanding dalam pengembangan fitur-fitur keamanan pada *library*. Panduan ini memberikan rekomendasi kepada pengembang untuk meningkatkan keamanan dan privasi aplikasi. Beberapa poin yang relevan meliputi:

1. *MSTG-RESILIENCE-1* (Integritas Aplikasi)

Aplikasi harus dapat mendeteksi dan merespons upaya modifikasi atau perusakan (*tampering*) pada kode dan kontainernya. Fitur *IntegrityDetector* pada *library* dirancang untuk memenuhi kontrol ini.*MSTG-PLATFORM-4* Aplikasi harus mengidentifikasi dan memitigasi potensi *bypass* keamanan seperti manipulasi *flag secure*.

2. *MSTG-RESILIENCE-2* (*Anti-Reverse Engineering*)

Aplikasi harus menerapkan pertahanan untuk menghambat analisis dinamis dan rekayasa balik (*reverse engineering*), misalnya melalui *obfuscation*. Penggunaan *library* Lsparanoid dalam proyek ini adalah salah satu implementasi dari kontrol ini.

3. *MSTG-RESILIENCE-3* (Deteksi *Root*)

Aplikasi harus dapat mendeteksi apakah ia berjalan pada perangkat yang telah di *root* dan meresponsnya dengan tepat. *RootDetector library*, dengan kemampuannya menembus *Magisk Hide* dan *Zygisk*, merupakan implementasi kuat dari kontrol ini.

4. *MSTG-RESILIENCE-4* (Deteksi *Emulator*)

Aplikasi harus dapat mendeteksi apakah ia dijalankan di dalam lingkungan emulator dan mengelola risiko yang terkait. Fitur *EmulatorDetector* secara spesifik menjawab kebutuhan kontrol ini.

5. *MSTG-RESILIENCE-5 (Anti-Tampering)*

Aplikasi harus dapat mendeteksi dan merespons upaya *hooking* pada level *runtime* atau modifikasi memori. *AntiTamperingDetector* pada *library* adalah implementasi langsung dari kontrol keamanan ini.

2.1.6 *Kotlin*

Menurut dokumentasi resmi di situs developer.*Android*.com, *kotlin* adalah bahasa pemrograman modern yang diketik secara statis dan mendukung paradigma pemrograman berorientasi objek serta fungsional. Dirancang untuk *interoperabilitas* penuh dengan *Java*, *Kotlin* memungkinkan pengembang untuk memanfaatkan ekosistem Java yang luas sambil menawarkan sintaksis yang lebih ringkas dan fitur-fitur canggih. Sejak diperkenalkan, *Kotlin* telah menjadi bahasa resmi untuk pengembangan aplikasi *android*, dengan lebih dari 60% pengembang *android* profesional menggunakannya.

Kotlin menawarkan berbagai fitur yang meningkatkan produktivitas dan keamanan kode, termasuk penanganan null yang lebih baik untuk mengurangi *NullPointerExceptions*, serta dukungan untuk pemrograman asinkron melalui coroutines. Selain itu, Kotlin mendukung pengembangan lintas platform, memungkinkan kode yang sama digunakan untuk aplikasi *Android*, iOS, dan

aplikasi *server side*. Dengan komunitas yang aktif dan dukungan dari JetBrains dan Google, *Kotlin* terus berkembang dan menjadi pilihan utama bagi pengembang yang mencari bahasa pemrograman modern dan efisien.

2.1.7 *Library*

Library adalah perangkat lunak yang dirancang untuk mendukung pengembang dalam menjalankan fungsi-fungsi tertentu secara efisien. Dalam konteks keamanan *android*, *library* untuk deteksi *rooting* dan *flag secure* bertujuan untuk mengidentifikasi status keamanan perangkat serta mencegah manipulasi fitur bawaan yang dapat mengurangi perlindungan aplikasi.

Deteksi *rooting* dilakukan dengan memeriksa keberadaan indikator tertentu, seperti berkas sistem yang dimodifikasi, akses *superuser*, atau aplikasi pihak ketiga yang sering digunakan untuk *rooting*. Sementara itu, fitur *flag secure* dirancang untuk melindungi data sensitif dari potensi pelanggaran, seperti tangkapan layar atau perekaman layar oleh aplikasi yang tidak sah.

Pendekatan berbasis *library* menawarkan fleksibilitas tinggi bagi pengembang untuk mengintegrasikan mekanisme keamanan tanpa mengganggu kinerja aplikasi. *library* semacam ini dapat digunakan untuk mendeteksi status *rooting* dan manipulasi *flag secure* secara real-time, memberikan pengembang kemampuan untuk mengambil tindakan yang sesuai, seperti menolak akses atau memperingatkan pengguna tentang potensi risiko.

Dengan memanfaatkan *library* ini, pengembang dapat memastikan bahwa aplikasi mereka tetap aman meskipun dijalankan pada perangkat dengan tingkat

ancaman yang lebih tinggi. Pendekatan ini menjadi landasan penting dalam meningkatkan keamanan dan privasi di ekosistem *android*, terutama pada aplikasi yang menangani data sensitif seperti keuangan, kesehatan, dan identitas pengguna.

2.1.8 *Mobile D*

Gambar 2. 1 Tahapan *Mobile D*

Metode *Mobile D* adalah salah satu metodologi pengembangan perangkat lunak yang dirancang khusus untuk aplikasi *mobile*. Metode ini merupakan adaptasi dari metodologi Agile, yang berfokus pada kelincahan dan fleksibilitas dalam pengembangan perangkat lunak. *Mobile D* menggabungkan prinsip-prinsip Agile dengan pendekatan yang lebih spesifik untuk kebutuhan pengembangan aplikasi *mobile* yang cocok untuk penelitian ini(Widayati & Nasir, 2018). *Mobile D* mencakup lima fase utama(Abrahamsson dkk, 2004) :

1. *Explore*

Pada fase ini, dilakukan perencanaan dan penyusunan proyek yang akan dikerjakan. Tahap ini meletak isu-isu dasar pengembangan sistem, antara lain arsitektur produk, proses pengembangan, dan lingkungan pengembangan.

2. *Initialize*

Fase ini bertujuan menyiapkan dan memverifikasi semua isu-isu kritis dalam pengembangan yang menentukan keberhasilan proyek. Di akhir tahap ini, diharapkan semua sumber daya telah siap untuk memulai membangun sistem.

3. Productionize

Pada tahap ini, semua kebutuhan fungsional diimplementasikan pada produk dengan fokus pada pengembangan fitur utama.

4. Stabilize

Fase ini melibatkan pengujian dan perbaikan untuk memastikan stabilitas dan kualitas produk, termasuk identifikasi dan perbaikan cacat.

5. System Test and Fix

Tahap akhir ini melibatkan pengujian sistem secara keseluruhan dan perbaikan akhir sebelum rilis produk.

2.1.9 Metode *Bypass*

Dalam konteks keamanan aplikasi *android*, metode *bypass* merupakan topik kritis yang mengkaji cara-cara penyerang untuk mengelabui mekanisme deteksi keamanan yang telah diterapkan. Secara umum, *bypass* mengacu pada teknik-teknik yang digunakan untuk menghindari atau menonaktifkan fungsi keamanan, seperti deteksi *root* dan implementasi *flag secure*, sehingga aplikasi yang seharusnya dilindungi menjadi rentan terhadap serangan.

Beberapa penelitian terbaru menunjukkan bahwa metode *bypass* pada deteksi *root* cukup kompleks. Soewito dan Suwandaru (2022) mengemukakan bahwa teknik *Java function hooking* dapat digunakan untuk menghindari deteksi *root*, dengan cara menyisipkan kode tambahan yang menyembunyikan keberadaan berkas atau aplikasi *root* yang umum ditemukan pada perangkat yang telah di *root*. Selain itu,(Rahman dkk 2021) menemukan bahwa penggunaan aplikasi *root hiding*, seperti *Magisk Hide*, memungkinkan penyerang untuk menyembunyikan status *root*

dari mekanisme deteksi standar, sehingga meningkatkan risiko kebocoran data sensitif.

Tidak hanya pada deteksi *root*, *bypass* terhadap mekanisme *flag secure* juga telah menjadi perhatian dalam penelitian keamanan *mobile*. Kumar menyatakan bahwa penyerang dapat menggunakan teknik *dynamic instrumentation* untuk memodifikasi perilaku aplikasi secara *realtime*, yang pada gilirannya menonaktifkan *flag secure* (Kumar dkk,2022). Teknik ini memungkinkan pengambilan tangkapan layar dan perekaman layar pada aplikasi, meskipun fitur *flag secure* telah diaktifkan. Kombinasi *bypass* untuk deteksi *root* secara simultan dapat meningkatkan kemungkinan terjadinya kebocoran data, karena mekanisme pertahanan yang ada menjadi tidak efektif.

Sintesis dari berbagai temuan tersebut menegaskan bahwa, meskipun terdapat banyak upaya untuk mendeteksi dan mencegah akses *root* serta memastikan *flag secure* aktif, masih ada celah yang dapat dimanfaatkan oleh penyerang. Oleh karena itu, pengembangan *library* yang mampu mengatasi teknik-teknik *bypass* tersebut menjadi sangat penting. *Library* ini diharapkan tidak hanya mendeteksi status *root* dan *flag secure* secara akurat, tetapi juga tahan terhadap upaya *bypass* yang dilakukan dengan metode modern.

2.1.10 Pentingnya Deteksi *Root* Dan Validasi *Flag Secure*

Deteksi *root* dan validasi *flag secure* merupakan dua aspek kritis dalam meningkatkan keamanan pada perangkat *android*. *Rooting* memberikan akses penuh kepada pengguna atau aplikasi pihak ketiga untuk memodifikasi sistem operasi, sehingga meningkatkan risiko terhadap serangan *malware*, pencurian data,

dan eksploitasi sistem. Misalnya, sebuah aplikasi perbankan yang dijalankan pada perangkat yang telah di *root* akan lebih rentan terhadap serangan, karena penyerang dapat mengeksploitasi akses administratif untuk mengubah fungsionalitas aplikasi dan mengakses data transaksi secara tidak sah (Bialon, 2020).

Sementara itu, *flag secure* (atau *flag secure*) merupakan mekanisme yang digunakan untuk mencegah pengambilan tangkapan layar atau perekaman layar dari konten aplikasi yang sensitif. Validasi *flag secure* sangat penting karena jika fitur ini tidak aktif atau dimanipulasi, maka informasi penting seperti *PIN*, data *login*, atau transaksi finansial dapat dengan mudah direkam oleh aplikasi jahat. Contohnya, jika sebuah aplikasi perbankan gagal mengaktifkan *flag secure*, maka penyerang dapat menggunakan aplikasi pihak ketiga untuk mengambil *screenshot* dari layar aplikasi, yang berpotensi mengakibatkan pelanggaran privasi dan kerugian finansial (Sudaryanto dkk., 2022).

Oleh karena itu, perancangan *library* yang mampu mendeteksi status *root* (ada/tidak) dan memastikan *flag secure* aktif secara efektif menjadi solusi penting untuk mengatasi celah keamanan tersebut. Dengan adanya *library* ini, pengembang dapat mengintegrasikan langkah-langkah keamanan tambahan ke dalam aplikasi, sehingga secara proaktif mencegah risiko yang timbul akibat perangkat yang di *root* dan penonaktifan *flag secure*. Pendekatan ini tidak hanya meningkatkan keamanan aplikasi secara teknis, tetapi juga memberikan jaminan perlindungan privasi kepada pengguna.

2.1.11 Contoh Bahaya

Terdapat beberapa kasus yang telah terjadi karena adanya perangkat *root* terhadap sebuah aplikasi milik gojek.com yang terkena serangan oleh aplikasi *Fake GPS* atau perangkat yang sudah di *inject* sebuah fitur *mock* gps telah menjadi salah satu metode yang digunakan oleh beberapa oknum pengemudi layanan transportasi *online*, seperti Gojek dan Grab, untuk memanipulasi lokasi mereka demi mendapatkan keuntungan secara tidak sah. Dengan memanfaatkan aplikasi ini, pengemudi dapat memalsukan posisi geografis mereka tanpa harus berpindah tempat secara fisik(gojek.com).

Teknik ini sering kali memerlukan modifikasi pada sistem *android*, termasuk pengaktifan opsi pengembang dan, dalam beberapa kasus, akses *root* pada perangkat. Namun, tindakan kecurangan semacam ini tidak hanya merugikan perusahaan penyedia layanan, tetapi juga pengemudi lain yang bekerja dengan jujur. Sebagai respons, perusahaan seperti Gojek telah mengembangkan sistem deteksi untuk mengidentifikasi penggunaan aplikasi ilegal, seperti *Fake* GPS dan aplikasi modifikasi lainnya, guna memastikan keadilan dan integritas dalam ekosistem layanan mereka.

Kasus nyata lainnya adalah ditemukannya aplikasi *spyware* bernama *LianSpy*. Aplikasi ini menyamar sebagai aplikasi sistem atau layanan keuangan untuk mendapatkan izin akses dari pengguna. Setelah memperoleh akses *root*, *LianSpy* dapat merekam aktivitas layar, mencuri data pribadi, dan mengirimkannya ke server penyerang, semuanya tanpa disadari oleh pengguna(the-sun.com).

2.1.12 *Black Box Testing*

Black box testing, atau pengujian kotak hitam, adalah metode pengujian perangkat lunak yang berfokus pada pengujian fungsionalitas aplikasi tanpa memperhatikan struktur internal atau kode sumbernya. Pengujian ini dilakukan dengan memberikan *input* tertentu dan mengamati *output* yang dihasilkan untuk memastikan kesesuaian dengan spesifikasi yang ditetapkan. Penguji tidak memerlukan pengetahuan tentang bagaimana sistem bekerja secara internal (Pipit Muliyah, dkk., 2020). Berikut adalah beberapa teknik yang umum digunakan dalam *black box testing*:

1. *Equivalence Partitioning* (Pembagian Ekuivalensi)

Teknik ini membagi domain *input* menjadi beberapa kelas ekuivalen, di mana setiap kelas mewakili sekumpulan nilai *input* yang diasumsikan akan diproses dengan cara yang sama oleh sistem. Dengan menguji satu nilai dari setiap kelas, penguji dapat mengurangi jumlah kasus uji yang diperlukan tanpa mengurangi cakupan pengujian.

2. *Boundary Value Analysis* (Analisis Nilai Batas**)**

Teknik ini berfokus pada pengujian nilai-nilai batas dari kelas ekuivalen, karena kesalahan sering terjadi pada batas-batas ini. Pengujian dilakukan pada nilai tepat di tepi batas, serta sedikit di atas dan di bawahnya. Contohnya, dengan rentang *input* valid 1 hingga 100, pengujian akan dilakukan pada nilai 0, 1, 2, 99, 100, dan 101.

3. *Decision Table Testing* (Pengujian Tabel Keputusan)

Teknik ini menggunakan tabel keputusan untuk memetakan berbagai kombinasi *input* dan *output* yang diharapkan. Setiap baris dalam tabel mewakili kombinasi kondisi *input* tertentu dan aksi yang sesuai. Pendekatan ini membantu dalam mengidentifikasi kondisi yang mungkin terlewatkan dan memastikan bahwa semua kemungkinan kombinasi diuji. Misalnya, dalam sistem penentuan diskon berdasarkan kategori pelanggan dan jumlah pembelian, tabel keputusan dapat digunakan untuk menguji semua skenario kombinasi tersebut.

4. *State Transition Testing* (Pengujian Transisi Status)

Teknik ini menguji perubahan status sistem berdasarkan kondisi atau peristiwa tertentu. Pengujian ini cocok untuk sistem yang berperilaku berbeda tergantung pada status saat ini dan *input* yang diberikan. Diagram transisi status digunakan untuk merepresentasikan status, transisi, dan peristiwa yang memicu perubahan tersebut. Contohnya, dalam aplikasi ATM, pengujian transisi status dapat memastikan bahwa sistem berpindah dari status '*Idle'* ke '*Processing'* saat kartu dimasukkan, dan ke status '*Transaction* Complete' setelah transaksi selesai.

5. *All-Pairs Testing* (Pengujian Semua Pasangan)

Juga dikenal sebagai *pairwise* testing, teknik ini menguji semua kemungkinan kombinasi pasangan *input*. Pendekatan ini efektif untuk menemukan *bug* yang muncul akibat interaksi antara dua parameter. Misalnya, jika sebuah aplikasi memiliki tiga parameter *input* dengan masing-masing tiga kemungkinan nilai, maka

semua kombinasi pasangan nilai akan diuji untuk memastikan tidak ada interaksi yang menyebabkan kesalahan.

6. *Cause-Effect Graphing* (Pembuatan Grafik Sebab-Akibat)

Teknik ini melibatkan pembuatan grafik yang menggambarkan hubungan antara penyebab (*input* atau kondisi) dan efek (*output* atau aksi) dalam sistem. Grafik ini membantu dalam mengidentifikasi kombinasi *input* yang menyebabkan *output* tertentu, memastikan bahwa semua hubungan sebab-akibat diuji. Misalnya, dalam sistem kontrol lampu lalu lintas, grafik sebab-akibat dapat digunakan untuk memetakan kondisi sensor dan waktu terhadap perubahan lampu merah, kuning, atau hijau.

7. *Error Guessing* (Menebak Kesalahan)

Berdasarkan pengalaman dan intuisi, penguji mencoba menebak area atau *input* yang mungkin menyebabkan kesalahan dalam sistem. Teknik ini tidak terstruktur seperti teknik lainnya, tetapi dapat efektif dalam mengidentifikasi *bug* yang tidak terduga. Misalnya, penguji mungkin mencoba memasukkan karakter khusus atau *string* yang sangat panjang ke dalam *field* input untuk melihat bagaimana sistem menangani *input* yang tidak biasa tersebut.

2.1.13 *White Box Testing*

White box testing, juga dikenal sebagai *clear box* testing atau *structural testing*, adalah metode pengujian perangkat lunak yang berfokus pada pemeriksaan struktur internal dan kode sumber dari aplikasi yang diuji. Dalam pendekatan ini, penguji memerlukan pemahaman mendalam tentang desain dan implementasi

perangkat lunak untuk membuat kasus uji yang efektif. Tujuan utamanya adalah memastikan bahwa semua jalur logika dalam program telah diuji dan berfungsi sesuai dengan spesifikasi yang ditetapkan(Sie dkk., 2022). Berikut adalah beberapa teknik yang umum digunakan dalam *White Box Testing*:

1. *Basis Path Testing* (Pengujian Jalur Dasar)

Teknik ini memungkinkan penguji untuk mengukur kompleksitas logika dari desain prosedural dan mendefinisikan himpunan dasar dari jalur eksekusi program. Langkah-langkahnya meliputi pembuatan *flowgraph* dari kode program, perhitungan kompleksitas *siklomatik*, dan identifikasi jalur independen yang harus diuji. Tujuannya adalah memastikan bahwa setiap jalur eksekusi diuji setidaknya sekali.

2. *Control Flow Testing* (Pengujian Aliran Kontrol)

Teknik ini memeriksa aliran kontrol dalam program untuk memastikan bahwa semua struktur kontrol, seperti percabangan dan *loop*, berfungsi sesuai dengan yang diharapkan. Pengujian ini membantu mengidentifikasi jalur yang mungkin terlewatkan atau tidak diuji dengan benar.

3. *Data Flow Testing* (Pengujian Aliran Data)

Fokus pada aliran data melalui variabel dalam program, teknik ini mengidentifikasi anomali seperti penggunaan variabel yang belum dinasalisasi atau variabel yang tidak pernah digunakan setelah diberi nilai. Tujuannya adalah memastikan bahwa data diproses dan dimanipulasi dengan benar sepanjang eksekusi program.

4. *Branch Testing* (Pengujian Cabang)

Teknik ini memastikan bahwa setiap cabang dari struktur keputusan, seperti pernyataan *if-else* dan *switch-case*, telah dieksekusi setidaknya sekali selama pengujian. Hal ini penting untuk memverifikasi bahwa semua kemungkinan jalur logika telah diuji dan berfungsi sesuai dengan spesifikasi.

5. *Loop Testing* (Pengujian Loop)

Teknik ini berfokus pada pengujian struktur loop dalam program, seperti *for, while*, dan *do-while loops*. Tujuannya adalah memastikan bahwa *loop* berfungsi dengan benar dalam berbagai skenario, termasuk eksekusi nol kali, satu kali, dan *multiple* kali, serta mengidentifikasi potensi masalah seperti loop tak berujung atau kondisi batas yang tidak tepat.

6. *Condition Coverage* (Cakupan Kondisi)

Teknik ini menguji setiap kondisi dalam pernyataan logika untuk memastikan bahwa setiap kondisi dapat dievaluasi ke nilai true dan *false* setidaknya sekali selama pengujian. Hal ini membantu memastikan bahwa semua kemungkinan hasil dari kondisi logika telah diuji.

2.2 Kajian Penelitian

Tabel 2. 1 Kajian Penelitian

No.	Judul Penelitian	Identitas Jurnal	Hasil/Isi Penelitian
1.	*Android* Sensitive Data Leakage Prevention with *rooting* Detection Using Java Function Hooking	Soewito & Suwandaru, 2022 Jurnal: Generation: Jurnal Teknik Informatika	Membahas teknik pencegahan kebocoran data sensitif pada *android* melalui deteksi *rooting* menggunakan Java function hooking. Metode ini dapat dihindari oleh teknik tertentu.

No.	Judul Penelitian	Identitas Jurnal	Hasil/Isi Penelitian
2.	On *Root* Detection Strategies for *Android* Devices	Bialon, Raphael , 2020, Computer Science	Menganalisis berbagai strategi deteksi *root* pada perangkat *android*, termasuk metode yang berasal dari ekosistem Linux, dan memperkenalkan pendekatan deteksi *root* jarak jauh untuk pemeriksaan integritas antar perangkat dalam jaringan nirkabel.
3.	A Systematic Assessment On *Android* Third-Party *Library* Detection Tools	Zhan, Xian Liu, Tianming Liu, Yepang Liu, Yang Li, Li Wang, Haoyu Luo, Xiapu, 2020, Software Engineerin	Melakukan evaluasi komprehensif terhadap alat deteksi *library* pihak ketiga pada *android*, berdasarkan enam kriteria untuk menilai efektivitas dan akurasi alat-alat tersebut.
4.	*Android* Security And Its *Rooting*— A Possible Improvement Of Its Security Architecture	Rahimi, Nick Nolen, John Gupta, Bidyut, 2019, Journal of Information Security	Mengeksplorasi bagaimana pengguna dapat meningkatkan fitur intrinsik *android* melalui penggunaan layanan Google, *rooting*, custom kernels, dan teknik rom, serta membahas risiko keamanan yang terkait dengan akses tidak sah ke data sensitif.
5.	Pemanfaatan *Flag Secure* Untuk Proteksi Buku Digital Pada E-*Library* Institut Teknologi Dirgantara Adisutjipto	Sudaryanto, Sudaryanto Wintolo, Hero Aryanto, Nandra Putra Informatika, Program Studi, 2022, Jurnal Komputasi	Membahas implementasi *flag secure* untuk melindungi konten buku digital dalam aplikasi e-*library*, mencegah tangkapan layar dan perekaman layar yang tidak sah.
6	*Android Rooting*: An Arms Race Between Evasion And Detection	Nguyen-Vu, Long Chau, Ngoc Tu Kang, Seongeun Jung, Souhwan, 2017, Security and Communication Networks	Penelitian ini membahas dinamika antara metode *rooting* pada perangkat *android* dan teknik deteksi *rooting*. Para peneliti mengkaji berbagai metode untuk mendeteksi perangkat yang telah di-*root* pada level Java dan native. Studi ini menunjukkan bahwa *rooting* menjadi semakin umum dan menimbulkan kekhawatiran keamanan terkait deteksi dan penghindarannya.
7	*Android* Malware Detection And Adversarial Methods	Niu, Weina Zhang, Xiaosong Yan, Ran Gong, Jiacheng, 2024, *Android* Malware Detection and Adversarial Methods	Buku ini secara komprehensif mengeksplorasi latar belakang, metode, pendekatan adversarial, dan tren masa depan terkait deteksi malware *android*. Penulis membahas metode deteksi malware umum, teknik adversarial yang digunakan untuk menghindari deteksi, dan strategi untuk meningkatkan ketahanan sistem terhadap serangan tersebut.
8	Joker: Trusted Detection Of Kernel *Root*kits In	Guri, Mordechai Poliak, Yuri Shapira, Bracha	Penelitian ini memperkenalkan 'JoKER', sebuah sistem yang bertujuan mendeteksi *root*kit pada kernel *android* dengan

No.	Judul Penelitian	Identitas Jurnal	Hasil/Isi Penelitian
	Android Devices Via Jtag Interface	Elovici, Yuval, 2015, Proceedings - 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2015	memanfaatkan antarmuka Joint Test Action Group (JTAG) perangkat keras untuk forensik memori yang tepercaya. Kerangka kerja ini terdiri dari komponen yang mengekstrak area memori kernel dan merekonstruksinya untuk analisis lebih lanjut. Hasil penelitian menunjukkan bahwa meskipun tujuan utama JTAG adalah pengujian sistem, antarmuka ini juga dapat digunakan untuk deteksi malware di mana metode tradisional gagal.
9	*Android Rooting*: Methods, Detection, And Evasion	Andrea-Susana Cuadros Casta, 2015, Telecommunications Engineering	Penelitian ini membahas berbagai metode *rooting* pada perangkat *android*, teknik deteksi yang digunakan untuk mengidentifikasi perangkat yang telah di-*root*, dan strategi penghindaran yang digunakan untuk menghindari deteksi. Studi ini memberikan wawasan mendalam tentang implikasi keamanan *rooting*, termasuk dampaknya pada sistem dan data pengguna. Penulis juga mengeksplorasi tantangan dalam mendeteksi *rooting* secara efektif serta metode yang digunakan untuk melewati mekanisme keamanan
10	Detecting *Android Root* Exploits By Learning From *Root* Providers	Ball, Justin R, 2015,	Penelitian ini memperkenalkan '*root*explorer', sebuah sistem yang dirancang untuk mendeteksi eksploitasi *root* pada perangkat *android* dengan mempelajari perilaku aplikasi *root* komersial. Dengan menganalisis lingkungan eksekusi yang diperlukan oleh aplikasi *root*, sistem ini dapat mengidentifikasi eksploitasi *root* yang disematkan dalam malware. Evaluasi eksperimental menunjukkan bahwa *root*explorer berhasil mendeteksi semua sampel malware yang diketahui melakukan eksploitasi *root* tanpa menghasilkan positif palsu.
11	*Android* Security And Its *Rooting*—A Possible Improvement Of Its Security Architecture	Rahimi, Nick Nolen, John Gupta, Bidyut, 2019, Journal of Information Security	Penelitian ini mengeksplorasi bagaimana pengguna dapat meningkatkan fitur bawaan *android* melalui penggunaan layanan Google, proses *rooting*, kernel kustom, dan teknik rom. Studi ini menyoroti potensi peningkatan keamanan dan fungsionalitas sistem *android* melalui metode tersebut, meskipun juga mengakui risiko keamanan yang terkait dengan proses *rooting*.
12	A Systematic Assessment On *Android* Third-Party *Library* Detection Tools	*Android* Security and Its *ROOTing*—A Possible Improvement of Its	Penelitian ini melakukan evaluasi komprehensif terhadap alat deteksi pustaka pihak ketiga (TPL) pada platform *android*. Para peneliti menilai berbagai alat yang tersedia berdasarkan enam kriteria: akurasi

No.	Judul Penelitian	Identitas Jurnal	Hasil/Isi Penelitian
		Security Architecture	konstruksi TPL, efektivitas, efisiensi, akurasi identifikasi versi, ketahanan terhadap obfuscation kode, dan kemudahan penggunaan. Studi ini bertujuan untuk memberikan pemahaman yang lebih baik tentang teknik deteksi TPL saat ini dan menawarkan panduan untuk penelitian di masa depan.
13	Machine Learning-Based Framework For Automatic Malware Detection Using *Android* Traffic Data	Alo, Uzoma Rita Nweke, Henry Friday Ele, Sylvester I., 2021, Journal of Theoretical and Applied Information Technology	Penelitian ini mengusulkan kerangka kerja berbasis pembelajaran mesin untuk mendeteksi aplikasi berbahaya melalui analisis data lalu lintas *android*. Empat algoritma pembelajaran mesin dievaluasi, dan hasilnya menunjukkan bahwa algoritma pohon keputusan dan ensemble berbasis pohon memberikan hasil yang lebih unggul dibandingkan dengan support vector machine dan regresi logistik. Pendekatan ini bertujuan untuk meningkatkan akurasi deteksi malware secara otomatis dengan memanfaatkan data lalu lintas jaringan dari perangkat *android*.
14	*Android* dan masa depan: analisis dampak terhadap pengguna	Rahma, Aziza Ashari Habib, Muhammad, 2021, Jurnal Pendidikan Dan Pengabdian Masyarakat	Penelitian ini membahas dampak penggunaan sistem operasi *android* terhadap pengguna, termasuk aspek positif dan negatifnya. Studi ini memberikan wawasan tentang bagaimana *android* memengaruhi kehidupan sehari-hari pengguna dan implikasinya di masa depan.
15	The *android* platform security model	Mayrhofer, René Stoep, Jeffrey Vander Brubaker, Chad Kralevich, Nick, 2021, ACM Transactions on Privacy and Security	Artikel ini menjelaskan model keamanan platform *android*, termasuk analisis ancaman dan ekosistem yang mendasarinya. Penulis membahas bagaimana berbagai langkah keamanan dalam implementasi *android* bekerja sama untuk mengurangi ancaman dan tantangan yang dihadapi.

Dari berbagai penelitian di atas, terlihat bahwa fokus utama dari penelitian di atas adalah deteksi *rooting* atau implementasi *flag secure* secara terpisah. Peneliti menawarkan kebaruan dengan mengembangkan sebuah *library* yang mendeteksi status *root* dan *flag secure* pada perangkat *android* secara bersamaan. Dengan lapisan keamanan yang lebih luas yang ditawarkan oleh pendekatan terpadu ini, pengembang aplikasi dapat secara efektif mencegah kebocoran data sensitif dan melindungi konten digital dari akses atau manipulasi yang tidak sah.

Ciri khas penelitian ini adalah integrasi deteksi *rooting* dan status *flag secure* ke dalam *library* yang dapat digunakan oleh pengembang aplikasi dengan mudah. Solusi ini meningkatkan keamanan aplikasi *android* secara keseluruhan, berbeda dengan penelitian sebelumnya yang biasanya berfokus pada satu aspek keamanan. Selain itu, dengan menggabungkan kedua metode ini, *library* yang di kembangkan dapat memberikan respon yang lebih adaptif terhadap berbagai ancaman keamanan, memastikan bahwa aplikasi hanya berjalan pada lingkungan yang aman dan sesuai dengan kebijakan keamanan yang ditetapkan.