IMPLEMENTASI KEAMANAN *WEBSITE* DARI SERANGAN *CROSS SITE REQUEST FORGERY* MENGGUNAKAN ALGORITMA HMAC-SHA256 PADA *FRAMEWORK* LARAVEL

SKRIPSI

DISUSUN OLEH

ISMI QONTAS LUBIS
NPM. 2109020179

PROGRAM STUDI TEKNOLOGI IN*FORM*ASI
FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI IN*FORM*ASI
UNIVERSITAS MUHAMMADIYAH SUMATERA UTARA
MEDAN
2025

IMPLEMENTASI KEAMANAN *WEBSITE* DARI SERANGAN *CROSS SITE REQUEST FORGERY* MENGGUNAKAN ALGORITMA HMAC-SHA256 PADA *FRAMEWORK* LARAVEL

SKRIPSI

Diajukan sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer (S.Kom) dalam Program Studi Teknologi Informasi pada Fakultas Ilmu Komputer dan Teknologi Informasi, Universitas Muhammadiyah Sumatera Utara

ISMI QONTAS LUBIS
NPM. 2109020179

PROGRAM STUDI TEKNOLOGI INFORMASI
FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI
UNIVERSITAS MUHAMMADIYAH SUMATERA UTARA
MEDAN
2025

LEMBAR PENGESAHAN

LEMBAR PENGESAHAN

Judul Skripsi	:	IMPLEMENTASI KEAMANAN *WEBSITE* DARI SERANGAN *CROSS SITE REQUEST FORGERY* MENGGUNAKAN ALGORITMA HMAC-SHA256 PADA *FRAMEWORK* LARAVEL
Nama Mahasiswa	:	ISMI QONTAS LUBIS
NPM	:	2109020179
Program Studi	:	TEKNOLOGI INFORMASI

Menyetujui
Komisi Pembimbing

(Andi Zulherry, S.Kom.,M.Kom.)
NIDN. 0129117901

Ketua Program Studi

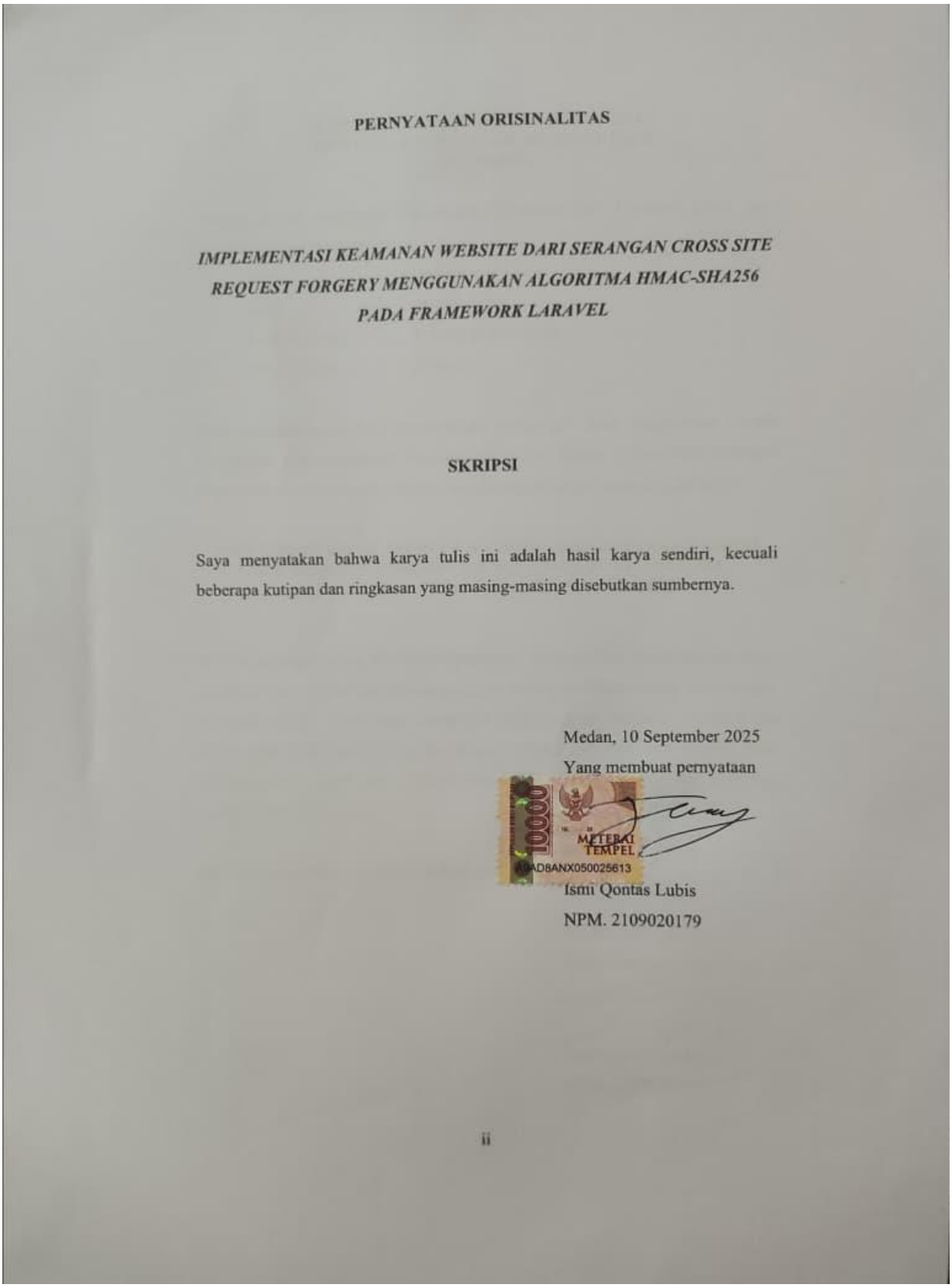
(Fatma Sari Hutagalung, S.Kom., M.Kom.)
NIDN. 0117019301

Dekan

(Dr. Al-Khowarizmi, S.Kom., M.Kom.)
NIDN. 0127099201

i

PERNYATAAN ORISINALITAS



PERNYATAAN ORISINALITAS

IMPLEMENTASI KEAMANAN WEBSITE DARI SERANGAN CROSS SITE REQUEST FORGERY MENGGUNAKAN ALGORITMA HMAC-SHA256 PADA FRAMEWORK LARAVEL

SKRIPSI

Saya menyatakan bahwa karya tulis ini adalah hasil karya sendiri, kecuali beberapa kutipan dan ringkasan yang masing-masing disebutkan sumbernya.

Medan, 10 September 2025

Yang membuat pernyataan

10000 METERAI TEMPEL AD8ANX050025613

Ismi Qontas Lubis

NPM. 2109020179

ii

PERNYATAAN PERSETUJUAN PUBLIKASI KARYA ILMIAH UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademika Universitas Muhammadiyah Sumatera Utara, saya bertanda tangan dibawah ini:

Nama	: Ismi Qontas Lubis
NPM	: 2109020179
Program Studi	: Teknologi Informasi
Karya Ilmiah	: Skripsi

Demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Muhammadiyah Sumatera Utara Hak Bedas Royalti Non-Eksekutif *(Non-Exclusive Royalty free Right)* atas penelitian skripsi saya yang berjudul:

IMPLEMENTASI KEAMANAN *WEBSITE* DARI SERANGAN *CROSS SITE REQUEST FORGERY* MENGGUNAKAN ALGORITMA HMAC-SHA256 PADA *FRAMEWORK* LARAVEL

Beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Non-Eksekutif ini, Universitas Muhammadiyah Sumatera Utara berhak menyimpan, mengalih media, memformat, mengelola dalam bentuk database, merawat dan mempublikasikan Skripsi saya ini tanpa meminta izin dari saya selama tetap mencantumkan nama saya sebagai penulis dan sebagai pemegang dan atau sebagai pemilik hak cipta.

Demikian pernyataan ini dibuat dengan sebenarnya.

Medan, 10 September 2025

Yang membuat pernyataan

Ismi Qontas Lubis

NPM. 2109020179

RIWAYAT HIDUP

DATA PRIBADI

Nama Lengkap	: Ismi Qontas Lubis
Tempat dan Tanggal Lahir	: Medan, 02 Oktober 2003
Alamat Rumah	: JL.Persatuan – Rela No.33
Telepon/HP	: 089613531255
E-mail	: ismiqontas@gmail.com
Instansi Tempat Kerja	: -
Alamat Kantor	: -

DATA PENDIDIKAN

SD	: SDT AL-BUKHARI MUSLIM MEDAN	TAMAT: 2015
SMP	: SMP AMIR HAMZAH MEDAN	TAMAT: 2018
SMA	: SMA NEGERI 4 MEDAN	TAMAT: 2021

KATA PENGANTAR

بِسْمِ اللهِ الرَّحْمٰنِ الرَّحِيْمِ

Puji syukur Penulis ucapkan kepada Allah SWT atas rahmat-Nya Penulis dapat menyelesaikan skripsi ini. Penulisan skripsi ini dilakukan dalam rangka memenuhi salah satu syarat untuk mencapai gelar Sarjana Komputer pada Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Muhammadiyah Sumatera Utara.

Penulis ingin mengucapkan terima kasih yang sebesar-besarnya kepada kedua orang tua yang telah memberikan dukungan moril maupun materil kepada Penulis. Dan Penulis menyadari bahwa tanpa bantuan, bimbingan, dan dukungan dari berbagai pihak sejak masa perkuliahan hingga masa penyusunan skripsi, sulit untuk menyelesaikan skripsi ini. Oleh karena itu dengan segala kerendahan hati, Penulis ingin menyampaikan terima kasih dan rasa hormat kepada:

1. Bapak Prof. Dr. Agussani, M.AP., selaku Rektor Universitas Muhammadiyah Sumatera Utara (UMSU)
2. Bapak Dr. Al-Khowarizmi, S.Kom., M.Kom., selaku Dekan Fakultas Ilmu Komputer dan Teknologi Informasi (FIKTI) UMSU.
3. Bapak Halim Maulana, S.T., M.Kom., selaku Wakil Dekan I Fakultas Ilmu Komputer dan Teknologi Informasi.
4. Bapak Dr. Lutfi Basit, S.Sos., M.I.Kom., selaku Wakil Dekan III Fakultas Ilmu Komputer dan Teknologi Informasi.
5. Ibu Fatma Sari Hutagalung, S.Kom., M.Kom., selaku Ketua Program Studi Teknologi Informasi yang telah memberikan bimbingan dan bantuan

selama Penulis menempuh studi serta yang selalu mengingatkan penulis untuk segera menyelesaikan skripsi ini.

6. Bapak Mhd. Basri, S.Si., M.Kom., selaku Sekretaris Program Studi Teknologi Informasi.
7. Bapak Andi Zulherry, S.Kom.,M.Kom., selaku Dosen Pembimbing yang telah banyak membantu penulis baik dukungan, saran, dan bimbingan dalam penyusunan penelitian ini.
8. Alm. Bapak Tsaufi Muslim Lubis, A.Md.Par., dan Ibu Sri Ramadona selaku Kedua Orang Tua penulis yang selalu mendoakan dan memberi dukungan baik mental maupun finansial selama ini.
9. Kepada Aidil Qontas Hamzah Lubis, S.E., dan Haflah Qontas, selaku abang dan adik penulis yang senantiasa memberikan dukungan dan semangat dalam penyelesaian skripsi ini.
10. Ucapan terima kasih kepada sahabat baik saya Ahmad Asadel dan Daris Fauzan Abila yang telah membantu dan memberi dukungan kepada penulis dalam menyelesaikan skripsi ini.
11. Terima kasih sebesar-besarnya untuk seluruh teman dan rekan seperjuangan dari grup Mulitmedia, dan D1-Esport. Dukungan dan masukan dari kalian semua sangat berarti hingga penulis bisa sampai di titik ini.
12. Kepada diri sendiri yang sudah berusaha keras dan berjuang sejauh ini. Mampu mengendalikan diri dari berbagai tekanan di luar keadaan dan tak pernah menyerah sesulit apapun. Mampu menguatkan dan meyakinkan bahwa semuanya akan selesai pada waktunya.

Perjalanan untuk sampai pada titik ini tidaklah mudah, dan Penulis sadar bahwa karya yang dihasilkan ini belumlah sempurna. Masih terdapat banyak kekurangan yang perlu diperbaiki. Namun, seperti sebuah langkah pertama, Penulis berharap skripsi ini bisa menjadi pijakan yang bermanfaat, baik bagi pembaca, rekan-rekan mahasiswa, maupun bagi Penulis sendiri untuk terus belajar.

Kritik dan saran yang membangun ibarat lentera yang akan menerangi jalan Penulis ke depan. Dengan segala kerendahan hati, Penulis menantikan masukan dari para pembaca. Terima kasih.

Medan, 4 Agustus 2025

Penulis

Ismi Qontas Lubis

IMPLEMENTASI KEAMANAN *WEBSITE* DARI SERANGAN *CROSS SITE REQUEST FORGERY* MENGGUNAKAN ALGORITMA HMAC-SHA256 PADA *FRAMEWORK* LARAVEL

ABSTRAK

Ancaman keamanan seperti *Cross-Site Request Forgery* (CSRF) menjadi tantangan serius bagi aplikasi *web*, bahkan yang dibangun dengan *framework modern* seperti Laravel yang memiliki proteksi bawaan. Penelitian ini bertujuan untuk merancang, mengimplementasikan, dan menganalisis efektivitas algoritma HMAC-SHA256 sebagai lapisan keamanan tambahan untuk memperkuat pertahanan terhadap serangan CSRF pada *framework* Laravel. Metode penelitian yang digunakan adalah penelitian terapan dengan pendekatan kuantitatif. Pengujian dilakukan menggunakan metode *Black Box Testing* melalui empat skenario berbeda untuk mengevaluasi sistem tanpa proteksi, fungsionalitas normal, serta efektivitas pertahanan berlapis dan lapisan HMAC secara mandiri. Hasil pengujian menunjukkan bahwa sistem tanpa proteksi sepenuhnya rentan terhadap serangan. Sebaliknya, sistem dengan pertahanan berlapis berhasil menolak serangan, di mana lapisan pertama (*token* CSRF Laravel) memblokir permintaan dengan *respons error* 419. Puncak pengujian membuktikan bahwa lapisan HMAC-SHA256 mampu berfungsi sebagai benteng pertahanan mandiri yang efektif, dengan berhasil memblokir serangan (*respons error* 403) bahkan ketika proteksi bawaan dinonaktifkan, tanpa mengganggu fungsionalitas normal aplikasi. Penelitian ini menyimpulkan bahwa implementasi strategi pertahanan berlapis (*Defense-in-Depth*) menggunakan HMAC-SHA256 secara signifikan meningkatkan ketahanan aplikasi *web* terhadap serangan CSRF dan terbukti menjadi mekanisme pertahanan independen yang andal.

Kata Kunci: *Cross-Site Request Forgery*, CSRF, HMAC-SHA256, Keamanan *Web*, Laravel, *Defense-in-Depth*.

IMPLEMENTATION OF *WEB*SITE *SECURITY* AGAINST CROSS-SITE REQUEST FORGERY ATTACKS USING THE HMAC-SHA256 ALGORITHM IN THE LARAVEL *FRAMEWORK*

ABSTRACT

Security threats such as Cross-Site Request Forgery (CSRF) pose a serious challenge to web applications, even those built with modern frameworks like Laravel that have built-in protections. This research aims to design, implement, and analyze the effectiveness of the HMAC-SHA256 algorithm as an additional Security layer to strengthen defenses against CSRF attacks on the Laravel framework. The methodology employed is applied research with a quantitative approach. Testing was conducted using the Black Box Testing method across four distinct scenarios to evaluate the system without protection, its normal functionality, and the effectiveness of both layered defense and the standalone HMAC layer. The results demonstrated that the unprotected system was completely vulnerable to attacks. Conversely, the system with layered defense successfully rejected attacks, where the first layer (Laravel's CSRF token) blocked requests with a 419 error response. Crucially, the HMAC-SHA256 layer proved to be an effective standalone defense, successfully blocking attacks (with a 403 error response) even when the default protection was disabled, all without disrupting the application's normal functionality. This study concludes that implementing a Defense-in-Depth strategy using HMAC-SHA256 significantly enhances web application resilience against CSRF attacks and proves to be a reliable independent defense mechanism.

Keywords: Cross-Site Request Forgery, CSRF, HMAC-SHA256, Web Security, Laravel, Defense-in-Depth.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I

PENDAHULUAN

1.1. Latar Belakang Masalah

Perkembangan teknologi in*form*asi mendorong lahirnya berbagai sistem berbasis *web* yang digunakan secara luas oleh instansi pemerintah, perusahaan, maupun masyarakat umum. Penggunaan *Website* untuk mendukung aktivitas administratif dan transaksi daring telah menjadi kebutuhan utama dalam digitalisasi layanan (Rizki & Ferico, 2021). Namun, kemajuan ini juga diiringi dengan meningkatnya ancaman terhadap keamanan sistem, salah satunya adalah serangan *Cross-Site Request Forgery* (CSRF).

CSRF merupakan salah satu jenis serangan yang mengeksploitasi kepercayaan *web* terhadap pengguna yang telah terautentikasi. Dengan memanfaatkan identitas korban, penyerang dapat mengirimkan permintaan yang tampak sah ke *server* tanpa sepengetahuan pengguna (Kour, 2020; Rankothge & Randeniya, 2020). Serangan ini memungkinkan penyerang melakukan tindakan seperti mengubah sandi, memindahkan dana, atau memodifikasi pengaturan pengguna dengan menyisipkan skrip berbahaya melalui media sosial, *email*, atau situs yang tidak terpercaya (Rankothge & Randeniya, 2020; Sajjad et al., 2024).

Framework Laravel telah menyediakan sistem proteksi dasar terhadap serangan CSRF melalui penggunaan *middleware* dan *token* keamanan. Namun, penelitian menunjukkan bahwa penggunaan *token* ini belum sepenuhnya efektif, terutama jika tidak didukung oleh metode otentikasi tambahan yang kuat. Bahkan dengan *middleware* aktif sekalipun, sistem masih dapat menjadi target eksploitasi apabila *token* dapat ditebak atau disalahgunakan (Calzavara et al., 2020).

Salah satu pendekatan yang menjanjikan dalam meningkatkan keamanan terhadap serangan semacam ini adalah dengan menerapkan algoritma HMAC-SHA256 (*Hash-based Message Authentication Code*). Algoritma ini menggabungkan fungsi *hash* SHA-256 dengan kunci rahasia untuk menghasilkan tanda tangan digital (Angkasa et al., 2023). HMAC telah terbukti andal dalam menjamin integritas data dan autentikasi pesan dalam berbagai skenario sistem keamanan, termasuk *cloud computing* dan sistem terdistribusi (Herzberg, 2025; Ranganathan & Srinivasan, 2025).

Dalam penelitian oleh (Suhaili et al., 2024), HMAC-SHA256 dapat diimplementasikan secara efisien bahkan pada sistem berbasis perangkat keras dengan per*form*a tinggi. Dalam konteks *web*, algoritma ini dapat digunakan untuk menghasilkan *token* yang unik dan tidak dapat dipalsukan, serta mampu memperkuat validasi setiap permintaan yang dikirimkan oleh klien ke *server* (Ramdani et al., 2023; Rana et al., 2023).

Studi sebelumnya yang mengimplementasikan pendekatan *token* sinkronisasi (Kour, 2020) memberikan kontribusi penting terhadap pengembangan sistem keamanan, namun masih menyisakan ruang untuk eksplorasi pendekatan yang lebih kriptografis dan dinamis seperti HMAC-SHA256. Oleh karena itu, penelitian ini mengusulkan penerapan algoritma HMAC-SHA256 sebagai lapisan tambahan keamanan terhadap serangan CSRF pada *Framework* Laravel.

1.2. Rumusan Masalah

Berdasarkan latar belakang yang telah dijelaskan, maka permasalahan dalam penelitian ini dapat dirumuskan sebagai berikut:

1. Perlunya implementasi algoritma HMAC-SHA256 untuk memperkuat sistem keamanan dari serangan *Cross-Site Request Forgery* (CSRF) pada *Framework* Laravel.
2. Pentingnya menganalisis perbandingan efektivitas antara sistem keamanan bawaan Laravel dengan sistem yang telah menerapkan algoritma HMAC-SHA256.

1.3. Batasan Masalah

Dalam penelitian ini, terdapat beberapa aspek yang perlu dibatasi agar ruang lingkup pembahasan tetap fokus, terarah, dan terukur. Pembatasan ini bertujuan untuk menghindari pembahasan yang terlalu luas serta memastikan bahwa penelitian hanya mencakup permasalahan yang sesuai dengan tujuan yang telah ditetapkan. Adapun batasan masalah dalam penelitian ini adalah sebagai berikut:

1. Penelitian ini hanya berfokus pada implementasi algoritma HMAC-SHA256 dalam *web* berbasis Laravel. Pembahasan tidak mencakup penerapan algoritma tersebut pada *platform* atau *Framework* selain Laravel.
2. Serangan CSRF yang ditangani dalam penelitian ini terbatas pada *web* yang menggunakan sesi pengguna (*user session*) sebagai metode autentikasi.

3. Pengujian dalam penelitian ini hanya dilakukan pada *web* dengan model permintaan berbasis *form*, serta tidak mencakup skenario komunikasi *real-time* atau *websocket*.
4. Penelitian ini tidak membahas atau menguji kombinasi algoritma *hash* lainnya seperti SHA-1 atau SHA-3, serta tidak mengevaluasi metode pencegahan CSRF lain di luar HMAC-SHA256.
5. Evaluasi keamanan dilakukan dalam konteks *web* yang menggunakan *token* CSRF yang disediakan oleh *server*. Penelitian ini tidak mencakup pengujian terhadap sistem keamanan lain seperti autentikasi dua faktor (2FA) atau sistem keamanan berbasis perangkat keras.

1.4. Tujuan Penelitian

Untuk menjawab rumusan masalah yang telah dirumuskan, serta dalam upaya memberikan solusi terhadap ancaman keamanan *web* yang disebabkan oleh serangan CSRF, penelitian ini memiliki tujuan sebagai berikut:

1. Mengimplementasikan algoritma HMAC-SHA256 pada *Framework* Laravel untuk memperkuat perlindungan *web*site terhadap serangan *Cross-Site Request Forgery* (CSRF).
2. Menganalisis hasil pengujian untuk membandingkan efektivitas keamanan antara sistem yang menggunakan HMAC-SHA256 dengan sistem keamanan bawaan Laravel.

1.5. Manfaat Penelitian

Penelitian ini diharapkan dapat memberikan manfaat bagi berbagai pihak, baik di kalangan akademisi, pengembang *web*, maupun masyarakat umum. Adapun manfaat yang diharapkan dari penelitian ini adalah sebagai berikut:

1. Penelitian ini memberikan kontribusi dalam pengembangan ilmu pengetahuan, khususnya di bidang keamanan *web*. Dengan penerapan HMAC-SHA256, penelitian ini memperkaya literatur mengenai pencegahan serangan CSRF dan memberikan panduan praktis untuk meningkatkan keamanan *web*, khususnya pada *Framework* Laravel.
2. Penelitian ini memberikan wawasan baru bagi pengembang *web* dalam memahami risiko CSRF serta menawarkan solusi teknis dengan menggunakan HMAC-SHA256. Diharapkan implementasi ini dapat memudahkan pengembang untuk mengamankan *web* dan mengurangi kerentanannya terhadap serangan siber.
3. Dalam konteks industri, penelitian ini memberikan alternatif solusi keamanan yang lebih kuat untuk melindungi data dan sistem *web*. HMAC-SHA256 dapat memberikan perlindungan tambahan terhadap serangan CSRF tanpa mempengaruhi per*form*a sistem, yang penting bagi perusahaan yang mengelola data sensitif.
4. Penelitian ini dapat menjadi referensi untuk penelitian selanjutnya dalam keamanan *web*. Hasil penelitian ini juga dapat digunakan untuk mengembangkan metode kriptografi lain dan teknik pencegahan serangan di masa mendatang.

BAB II

LANDASAN TEORI

2.1. Website

2.1.1. Pengertian Website

Website adalah sekumpulan halaman yang saling terhubung dan dapat diakses melalui jaringan internet. *Website* digunakan untuk menyajikan in*form*asi dalam bentuk teks, gambar, animasi, suara, atau gabungan dari semua elemen tersebut. *Website* berfungsi sebagai sarana komunikasi yang memungkinkan pengguna untuk berbagi in*form*asi secara *global* melalui jaringan internet (Limbong & Sriadhi, 2021; Rizki & Ferico, 2021).

Secara teknis, *Website* adalah sebuah aplikasi berbasis *web* yang dapat diakses melalui URL (*Uniform Resource Locator*) menggunakan *browser*. *Website* dapat bersifat statis atau dinamis, tergantung pada bagaimana konten dikelola dan ditampilkan. *Website* statis umumnya tidak memerlukan interaksi pengguna, sementara *Website* dinamis memungkinkan pembaruan konten secara otomatis berdasarkan interaksi pengguna (Limbong & Sriadhi, 2021; Rizki & Ferico, 2021).

2.1.2. Jenis-jenis Website

Website dapat dibagi menjadi beberapa jenis berdasarkan fungsinya, antara lain:

a. *Website* Statis: *Website* yang kontennya tidak berubah kecuali ada pembaruan manual dari pengelola *Website*. Jenis *Website* ini sering digunakan untuk in*form*asi yang tetap, seperti halaman profil atau portofolio (Limbong & Sriadhi, 2021).

b. *Website* Dinamis: *Website* yang memungkinkan pembaruan kontennya secara otomatis berdasarkan interaksi pengguna. *Website* ini digunakan untuk aplikasi *e-commerce*, media sosial, dan lain sebagainya (Rizki & Ferico, 2021).

2.1.3. Keamanan Website

Seiring dengan perkembangan teknologi in*form*asi, *Website* kini menjadi lebih rentan terhadap serangan siber. Keamanan *Website* menjadi aspek penting untuk melindungi data sensitif dan menjaga integritas sistem. Jenis serangan yang umum terjadi pada *Website* meliputi *Cross-Site Scripting* (XSS), *SQL Injection*, dan *Cross-Site Request Forgery* (CSRF), yang dapat mengeksploitasi celah keamanan untuk merusak atau mencuri data pengguna (Budiman et al., 2021).

Untuk mengatasi masalah ini, pengembang *Website* perlu memastikan bahwa aplikasi *web* yang mereka buat sudah dilengkapi dengan mekanisme keamanan yang memadai, seperti enkripsi, penggunaan *token* keamanan, dan validasi *input* yang ketat. Salah satu teknik yang digunakan untuk mencegah serangan CSRF adalah dengan menambahkan *token* CSRF yang unik pada setiap permintaan pengguna yang harus diverifikasi oleh *server* sebelum diproses (Limbong & Sriadhi, 2021; Rizki & Ferico, 2021). Selain itu, penggunaan protokol HTTPS juga sangat dianjurkan untuk memastikan bahwa komunikasi antara pengguna dan *server* terlindungi dari potensi peretasan.

2.2. Cross Site Request Forgery (CSRF)

2.2.1. Pengertian CSRF

Cross-Site Request Forgery (CSRF) adalah jenis serangan di mana penyerang memanfaatkan kepercayaan yang diberikan oleh aplikasi *web* terhadap

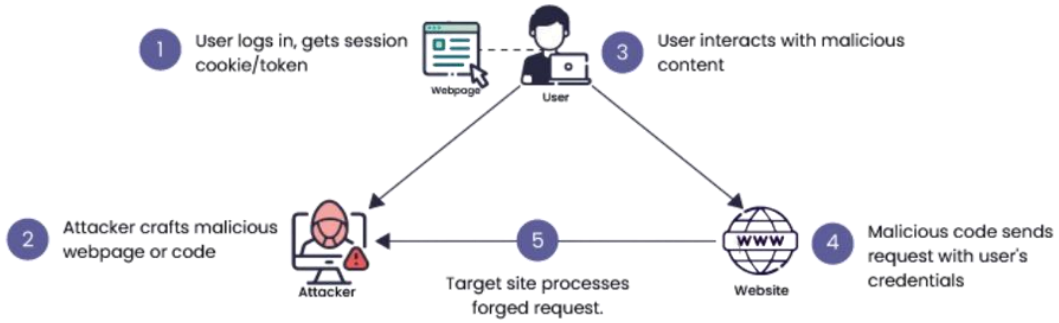
browser pengguna untuk mengirimkan permintaan yang tidak sah ke situs yang diserang. Serangan ini biasanya dilakukan dengan menyuntikkan *link* atau *script* berbahaya ke dalam halaman yang sah yang diakses oleh korban, yang kemudian mengirimkan permintaan yang tidak diinginkan atas nama korban ke situs *web* yang menjadi target serangan. Tujuan utama dari serangan ini adalah untuk melakukan tindakan yang merugikan pada situs *web* yang diserang, seperti mengubah data atau mengeksekusi tindakan yang tidak sah (Ashari et al., 2022; Kour, 2020).

Serangan CSRF dapat dilakukan dengan berbagai cara, tergantung pada jenis serangannya. CSRF umumnya dapat diklasifikasikan berdasarkan cara penyerang mengelabui korban untuk mengirimkan permintaan HTTP palsu dan berdasarkan keadaan autentikasi korban pada aplikasi *web* tepercaya. Dua jenis serangan CSRF yang umum ditemukan adalah *Stored* CSRF, di mana kode berbahaya disimpan pada *server* dan dieksekusi ketika korban mengakses halaman tertentu, dan Reflected CSRF, di mana penyerang mengirimkan *link* berbahaya yang dieksekusi segera setelah korban mengkliknya (Rankothge & Randeniya, 2020).

Dalam situasi di mana serangan CSRF berhasil, pelaku berhasil menciptakan keadaan di mana korban tanpa sadar melakukan tindakan yang tidak diinginkan, seperti mengganti alamat *email* atau memasukkan data yang tidak sah. Dampak dari serangan ini dapat sangat beragam, tergantung pada hak akses yang dimiliki korban. Dalam beberapa kasus, penyerang dapat memperoleh akses penuh ke akun pengguna, bahkan jika korban memiliki hak istimewa tertentu

dalam aplikasi, yang memungkinkan penyerang untuk mengambil alih kontrol atas seluruh data dan fitur aplikasi tersebut (Sajjad et al., 2024).

2.2.2. Mekanisme Serangan CSRF



Gambar 2.1 Cara Kerja CSRF

Serangan CSRF (*Cross Site Request Forgery*) adalah jenis serangan keamanan yang mengeksploitasi kepercayaan aplikasi *web* terhadap pengguna yang sudah terautentikasi. Berikut adalah cara kerja serangan CSRF:

1. Ketika seorang pengguna *login* ke sebuah situs *web*, mereka biasanya menerima *session cookie* atau *token* autentikasi yang digunakan situs *web* tersebut untuk mengenali mereka sebagai pengguna yang sudah *login*.
2. Seorang penyerang membuat halaman *web* jahat atau menyuntikkan kode berbahaya ke dalam halaman *web* yang sah. Kode ini berisi permintaan ke situs *web* target, seperti untuk mentransfer dana atau mengubah pengaturan akun.
3. Penyerang menipu pengguna agar mengunjungi halaman *web* jahat tersebut atau berinteraksi dengan halaman *web* yang telah disusupi, seringkali melalui teknik rekayasa sosial (*social Engineering*) seperti *email phishing* atau tautan berbahaya.

4. Kode berbahaya di halaman *web* tersebut mengirimkan permintaan ke situs *web* target. Permintaan ini menyertakan *token* autentikasi atau *session cookie* milik pengguna, yang dilampirkan secara otomatis oleh *browser* karena berasal dari origin yang sama.
5. Situs *web* target menerima permintaan tersebut dan memprosesnya seolah-olah permintaan itu sah karena berisi kredensial autentikasi yang valid. Hal ini dapat menyebabkan dilakukannya tindakan atas nama pengguna tanpa persetujuan atau sepengetahuan mereka.
6. Penyerang mendapatkan akses tidak sah ke akun pengguna dan dapat melakukan tindakan seperti mentransfer dana, mengubah pengaturan akun, atau mengakses in*form*asi sensitif.

2.2.3. Mekanisme Token CSRF

Gambar 2.2 Cara Kerja Token CSRF

Token Anti-CSRF adalah sebuah *token* yang digunakan untuk melindungi aplikasi *web* dari serangan *Cross Site Request Forgery* (CSRF). Cara kerjanya sebagai berikut:

1. Saat pengguna meminta halaman dengan *form*, *server* membuat *token* unik dan rahasia.

2. *Server* mengirimkan halaman tersebut ke pengguna, dengan *token* disisipkan di dalam *form* (biasanya sebagai *hidden field*) dan juga dikaitkan dengan sesi pengguna (misalnya *via cookie*).
3. Ketika pengguna mengirimkan (*submit*) *form*, *browser* mengirimkan kembali data *form* beserta *token* dari *hidden field* (dan *cookie*, jika digunakan).
4. *Server* menerima data, lalu memastikan *token* dari *form* cocok dengan *token* yang diharapkan untuk sesi pengguna tersebut.
5. Jika cocok, permintaan dianggap sah dan diproses. Jika tidak cocok, permintaan ditolak karena kemungkinan besar merupakan permintaan palsu (serangan CSRF).

Dengan mekanisme *token* Anti-CSRF, serangan CSRF dapat dicegah, karena *server* hanya akan memproses permintaan (*request*) jika disertai *token* yang valid dan sesuai dengan sesi pengguna yang sah.

2.3. Algoritma HMAC-SHA256

Dalam dunia keamanan digital, fungsi *hash* kriptografi merupakan sebuah prosedur matematis yang krusial, berfungsi untuk mengubah data masukan dengan ukuran acak menjadi sebuah string keluaran berukuran tetap yang dikenal sebagai nilai *hash*. Hasil keluaran ini sering diibaratkan sebagai "sidik jari digital" yang unik untuk data asli, menjadikannya alat yang sangat efektif untuk memverifikasi integritas data (Uriawan et al., 2024). Salah satu karakteristik utamanya adalah sensitivitasnya yang tinggi, di mana perubahan sekecil apa pun pada data asli akan menghasilkan nilai *hash* yang sama sekali berbeda (Uriawan et al., 2024).

Agar dapat diandalkan untuk tujuan keamanan, fungsi *hash* kriptografi yang baik harus memiliki beberapa sifat esensial. Sifat pertama adalah satu arah (Preimage Resistance), yang berarti secara komputasi sangat sulit atau bahkan tidak mungkin untuk mendapatkan kembali data masukan asli hanya dengan mengetahui nilai *hash*-nya, membuat proses ini tidak dapat dibalik (Dewi, 2023). Sifat kedua adalah *Second Preimage Resistance*, di mana jika sudah diketahui suatu data masukan dan nilai *hash*-nya, sulit secara komputasi untuk menemukan data masukan lain yang berbeda namun memiliki nilai *hash* yang sama (Dewi, 2023; Suhaili et al., 2024).

Sifat ketiga, yang sangat penting untuk menjaga integritas data, adalah tahan tumbukan (Collision Resistance), yang berarti harus sulit untuk menemukan dua data masukan yang berbeda ($m_1 \neq m_2$) yang menghasilkan nilai *hash* yang sama persis ($H(m_1) = H(m_2)$) (Dewi, 2023; Suhaili et al., 2024). Kegagalan dalam menahan tumbukan inilah yang membuat algoritma lama seperti MD5 dan SHA-1 kini dianggap tidak aman untuk aplikasi *modern* (Angkasa et al., 2023). Oleh karena itu, standar yang lebih baru dan aman seperti keluarga algoritma SHA-2 lebih direkomendasikan (Angkasa et al., 2023; Uriawan et al., 2024).

SHA-256 (Secure *Hash* Algorithm 256-bit) adalah salah satu implementasi dari keluarga SHA-2 yang dikembangkan oleh *National Security Agency* (NSA) dan diterbitkan sebagai standar oleh *National Institute of Standards and Technology* (NIST) (Angkasa et al., 2023; Suhaili et al., 2024). Sesuai namanya, algoritma ini dirancang untuk menghasilkan nilai *hash* dengan panjang tetap 256 bit dari data masukan dengan ukuran berapa pun. Namun, penting untuk dicatat bahwa fungsi *hash* kriptografi dalam bentuk dasarnya, termasuk SHA-256, tidak

melibatkan penggunaan kunci rahasia dalam operasinya, sehingga hanya dapat menjamin integritas data, bukan autentikasi sumber (Suhaili et al., 2024).

Untuk menambahkan lapisan autentikasi, digunakanlah sebuah mekanisme yang disebut MAC (*Message Authentication Code*), yang secara spesifik dapat diimplementasikan sebagai HMAC (*Hash-based Message Authentication Code*) bila digunakan bersama fungsi *hash* dan sebuah kunci rahasia (*secret key*) (Angkasa et al., 2023; Dewi, 2023). Tujuan utama HMAC adalah untuk memverifikasi bahwa sebuah pesan tidak hanya utuh (integritas), tetapi juga benar-benar berasal dari pihak yang memegang kunci rahasia yang sah (autentikasi) (Dewi, 2023; Uriawan et al., 2024).

Konstruksi HMAC secara cerdas melibatkan penggunaan fungsi *hash* (dilambangkan dengan H) sebanyak dua kali secara bertingkat (*nested*) dengan mengolah kunci rahasia (K) dan pesan (m) (Angkasa et al., 2023; Suhaili et al., 2024). *Form*ula umum untuk konstruksi HMAC adalah sebagai berikut:

$$HMAC(K, m) = ((K_0 \oplus opad \,||\, H((K_0 \oplus ipad)||m)) \quad \text{.....(2.1)}$$

Penerapan spesifik dari *form*ula di atas dengan menggunakan fungsi *hash* SHA-256 dikenal sebagai HMAC-SHA256 (Dewi, 2023; Rana et al., 2023). Dalam implementasi ini, fungsi H pada rumus tersebut digantikan dengan algoritma SHA-256 (Suhaili et al., 2024). Proses ini secara efektif menggabungkan struktur HMAC yang aman dengan kekuatan dan ketahanan fungsi *hash* SHA-256. Kombinasi ini menghasilkan sebuah mekanisme keamanan yang sangat populer dan andal untuk berbagai aplikasi keamanan, termasuk dalam protokol standar seperti TLS, penandatanganan JSON *Web Token* (JWT), dan pengamanan API, karena mampu memberikan keseimbangan yang baik antara

tingkat keamanan yang kuat dan efisiensi komputasi yang memadai untuk sebagian besar aplikasi *modern* (Angkasa et al., 2023; Dewi, 2023).

2.4. Framework Laravel

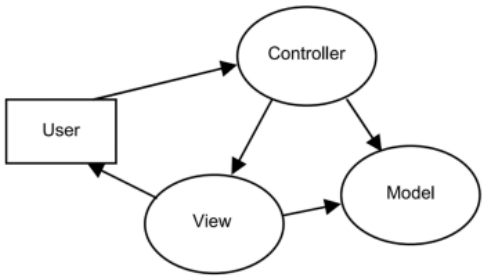
2.4.1. Pengertian Framework Laravel

Framework dalam pengembangan aplikasi *web* adalah kerangka kerja yang menyediakan fungsi, pustaka, dan alat bantu untuk mempercepat proses pengembangan. Penggunaan *framework* memungkinkan pengembang untuk memanfaatkan struktur dan fungsi yang telah disediakan, tanpa perlu membangun komponen dari awal (Lubis et al., 2022).

Salah satu bahasa pemrograman yang populer untuk pengembangan aplikasi *web* sisi server adalah PHP. Untuk mempermudah pengembangan, berbagai *framework* PHP telah diciptakan, dan salah satu yang paling terkenal adalah Laravel (Jalis et al., 2025; Lubis et al., 2022). Laravel adalah *framework open-source* yang pertama kali dirilis pada tahun 2011 dan dirancang dengan pola arsitektur *Model-View-Controller* (MVC), yang memisahkan kode aplikasi menjadi Model (logika data), *View* (antarmuka pengguna), dan *Controller* (pengatur alur permintaan) (Jalis et al., 2025; Lubis et al., 2022).

Laravel populer karena sintaksis yang elegan, kemudahan pengelolaan *database*, fitur keamanan terintegrasi, dan fleksibilitas dalam pengembangan. *Framework* ini dilengkapi dengan berbagai fitur seperti *Eloquent* ORM, *Blade Templating Engine*, *routing*, *middleware*, *Artisan* CLI, dan sistem migrasi *database*, yang membantu pengembang membangun aplikasi *web* yang fungsional dan aman (Jalis et al., 2025; Lubis et al., 2022).

2.4.2. Arsitektur Laravel



Gambar 2.3 Konsep MVC Laravel

Laravel secara fundamental mengadopsi pola arsitektur *Model-View-Controller* (MVC) dalam struktur pengembangannya. Pola MVC ini merupakan pendekatan desain perangkat lunak yang memisahkan representasi in*form*asi dari interaksi pengguna, dengan tujuan utama untuk mengorganisasi kode aplikasi menjadi komp*onen*-komp*onen* yang lebih terstruktur, modular, dan mudah dikelola (Lubis et al., 2022). Dalam konteks Laravel, komp*onen* MVC ini memiliki peran sebagai berikut:

1. Model: Bertanggung jawab untuk merepresentasikan data dan logika bisnis aplikasi. Komp*onen* Model berinteraksi langsung dengan *database* untuk mengambil, menyimpan, atau memanipulasi data. Di Laravel, ini sering diimplementasikan menggunakan *Eloquent ORM*.
2. *View*: Bertugas untuk menampilkan data atau in*form*asi kepada pengguna dalam bentuk antarmuka (*user Interface*). Komp*onen* *View* mengambil data yang telah diolah oleh *Controller* dan menampilkannya dalam *form*at yang sesuai (biasanya HTML). Laravel menggunakan *Blade Templating Engine* untuk memudahkan pembuatan *View*.
3. *Controller*: Berperan sebagai perantara antara Model dan *View*. *Controller* menerima permintaan (*request*) dari pengguna, memprosesnya (misalnya

dengan mengambil atau memanipulasi data melalui Model), dan kemudian menentukan *View* mana yang akan ditampilkan kepada pengguna beserta data yang diperlukan.

Dengan pemisahan tugas ini, pengembangan aplikasi menjadi lebih terorganisir, memungkinkan pengembang untuk bekerja pada bagian yang berbeda secara independen dan mempermudah proses pemeliharaan serta pengembangan fitur di kemudian hari (Lubis et al., 2022).

2.4.3. Fitur Utama Laravel

Popularitas Laravel sebagai *Framework* PHP didukung oleh ekosistem fitur yang kaya dan *modern*, dirancang untuk meningkatkan produktivitas pengembang dan kualitas aplikasi. Beberapa fitur utama yang menjadi unggulan Laravel antara lain:

1. *Eloquent* ORM (*Object-Relational Mapping*): Menyediakan implementasi *ActiveRecord* yang memudahkan pengembang berinteraksi dengan *database*. Operasi *database* seperti membuat, membaca, mengubah, dan menghapus data (CRUD) dapat dilakukan melalui objek PHP tanpa perlu menulis *Query SQL* secara manual.
2. *Blade Templating Engine*: Sistem *template* bawaan Laravel yang sederhana namun kuat. *Blade* memungkinkan penulisan kode PHP langsung di dalam *view* dengan sintaksis yang bersih dan menyediakan fitur seperti pewarisan *template* (*template inheritance*) dan *sections* untuk membuat tata letak halaman yang konsisten.
3. *Artisan Console*: Sebuah antarmuka baris perintah (CLI) yang menyediakan berbagai perintah bantu untuk mempercepat tugas-tugas

pengembangan rutin, seperti membuat *Controller*, *model*, *migration*, *seeder*, menjalankan *job* antrian, dan banyak lagi.

4. *Routing*: Laravel menyediakan sistem *routing* yang sangat fleksibel dan ekspresif untuk mendefinisikan bagaimana aplikasi me*respons* permintaan URL dari pengguna, baik untuk aplikasi *web* maupun API.
5. *Middleware*: Menyediakan mekanisme yang nyaman untuk memfilter permintaan HTTP yang masuk ke aplikasi. *Middleware* dapat digunakan untuk berbagai keperluan seperti autentikasi pengguna, verifikasi *token* CSRF, *logging*, atau modifikasi *request/response*.
6. Keamanan Terintegrasi: Laravel memiliki fokus kuat pada keamanan dan menyediakan fitur bawaan untuk melindungi aplikasi dari kerentanan umum seperti *Cross-Site Scripting* (XSS), injeksi *SQL* (melalui *Eloquent*/*Query Builder*), dan *Cross-Site Request Forgery* (CSRF). Fitur autentikasi dan otorisasi juga terintegrasi dengan baik.
7. Manajemen Dependensi dengan *Composer*: Laravel memanfaatkan *Composer*, manajer paket standar PHP, untuk mengelola semua pustaka atau dependensi pihak ketiga yang dibutuhkan oleh aplikasi.
8. Migrasi *Database* dan *Seeding*: Memudahkan pengelolaan skema *database* melalui *migrations* (kontrol versi untuk *database*) dan pengisian data awal atau *dummy* melalui *seeders*.

Fitur-fitur ini, beserta aspek lain seperti *Class Auto Loading*, *IoC Containers*, dan dukungan *Unit Testing*, menjadikan Laravel sebagai *Framework* yang komprehensif untuk membangun aplikasi *web* yang *modern*, aman, dan mudah dipelihara (Jalis et al., 2025).

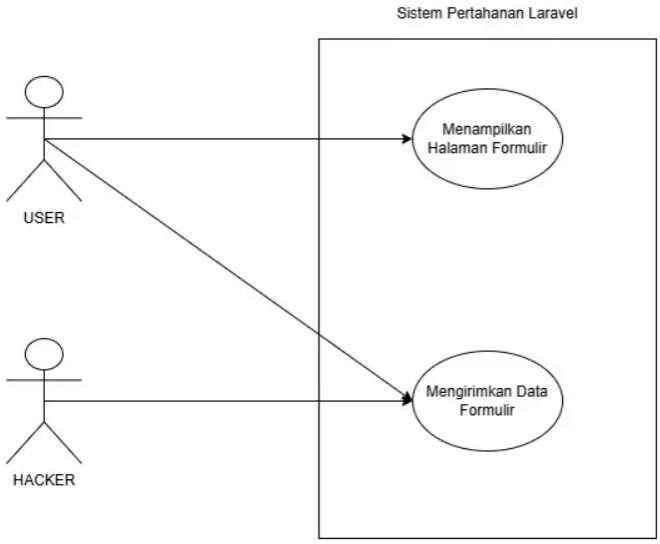
2.5. Unified Modelling Language (UML)

Unified Modelling Language (UML) adalah bahasa pemodelan standar yang digunakan untuk merancang dan mendokumentasikan sistem perangkat lunak, terutama dalam pengembangan berorientasi objek. UML bukan metode pengembangan, melainkan bahasa visual yang digunakan untuk mendefinisikan kebutuhan, melakukan analisis dan desain, serta menggambarkan arsitektur sistem (Hasanah & Untari, 2020).

UML muncul untuk memenuhi kebutuhan pemodelan visual guna menspesifikasikan, menggambarkan, membangun, dan mendokumentasikan berbagai artefak sistem perangkat lunak. Dengan serangkaian notasi grafis standar, UML memungkinkan pengembang, analis, dan pemangku kepentingan lainnya untuk berkomunikasi secara efektif mengenai rancangan sistem. UML juga dapat dihubungkan langsung dengan bahasa pemrograman (seperti *Java*, C++, *Visual Basic*) dan digunakan untuk menghasilkan dokumentasi proyek yang komprehensif. Standarisasi UML dilakukan oleh *Object Management Group* (OMG), yang mengeluarkan standar teknologi berorientasi objek (Hasanah & Untari, 2020).

Di antara berbagai diagram UML, *Use Case* Diagram memiliki peran penting dalam menggambarkan fungsionalitas yang diharapkan dari sistem dan interaksi antara pengguna (aktor) dengan sistem tersebut.

2.5.1. Use Case Diagram



Gambar 2.4 Use Case Diagram

Berdasarkan perancangan sistem, *Use Case* Diagram pada Gambar 2.4 di atas menggambarkan interaksi fungsional antara aktor dengan sistem *web*site yang dikembangkan. Terdapat dua aktor utama yang diidentifikasi dalam lingkup penelitian ini:

1. Pengguna Sah: Merupakan pengguna yang telah terautentikasi dan memiliki hak untuk berinteraksi dengan sistem secara normal.
2. Penyerang (*Attacker*): Merupakan pihak eksternal yang bertujuan untuk mengeksploitasi sistem dengan cara yang tidak sah.

Interaksi antara aktor dan sistem terfokus pada dua fungsi utama (*Use Case*):

a. Menampilkan Halaman *Form*ulir: Fungsi ini digunakan oleh Pengguna Sah untuk meminta dan menampilkan halaman antarmuka yang berisi *form*ulir *input* data.
b. Mengirimkan Data *Form*ulir: Fungsi ini digunakan oleh Pengguna Sah untuk mengirimkan data yang telah diisi pada *form*ulir ke server. Fungsi

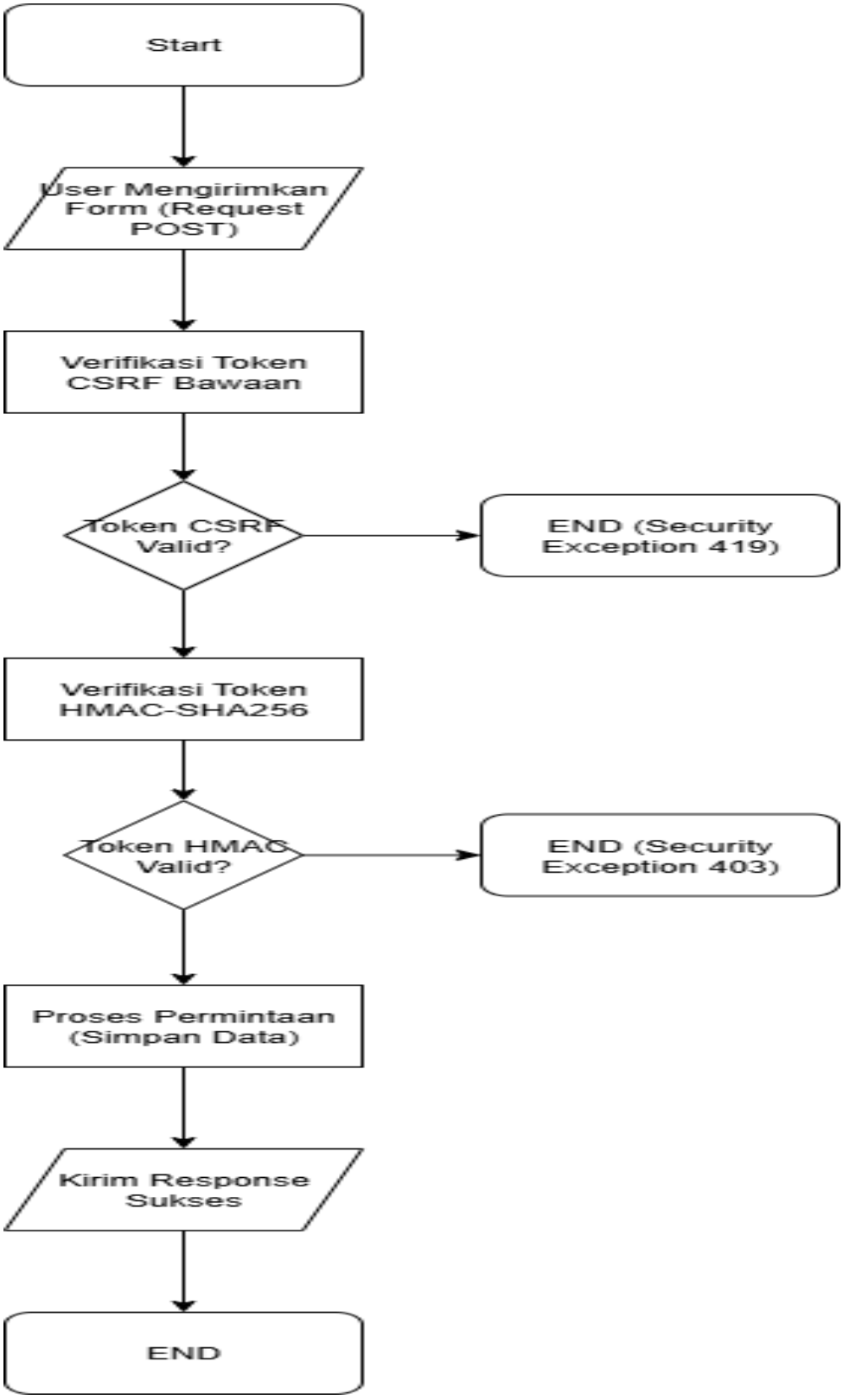
inilah yang menjadi target utama dalam skenario serangan. Penyerang juga menargetkan *Use Case* ini, namun dengan cara memicunya secara tidak sah melalui peramban milik Pengguna Sah untuk mengirimkan permintaan yang dipalsukan (*forged request*).

Diagram ini secara efektif memvisualisasikan masalah keamanan *Cross-Site Request Forgery* (CSRF), di mana fungsionalitas yang sah dapat dieksploitasi oleh pihak yang tidak berwenang.

2.6. Flowchart

Flowchart adalah diagram yang menggambarkan langkah-langkah dan urutan prosedur dalam suatu program atau sistem secara visual. Menggunakan simbol standar, *Flowchart* merepresentasikan elemen-elemen seperti operasi, alur logika, pengambilan keputusan, dan aktivitas lainnya. Dengan kemampuannya memvisualisasikan proses yang kompleks, *Flowchart* mempermudah pemahaman alur kerja, menganalisis alternatif, dan memecahkan masalah (Zalukhu et al., 2023).

Sebagai alat visualisasi, *Flowchart* mengilustrasikan langkah-langkah untuk menyelesaikan masalah menggunakan simbol yang sederhana dan mudah dipahami, yang memudahkan pemahaman bagi analis, pemrogram, dan pihak terkait. *Flowchart* berfungsi untuk menggambarkan urutan proses dan hubungan antar langkah dalam suatu sistem (Zalukhu et al., 2023).



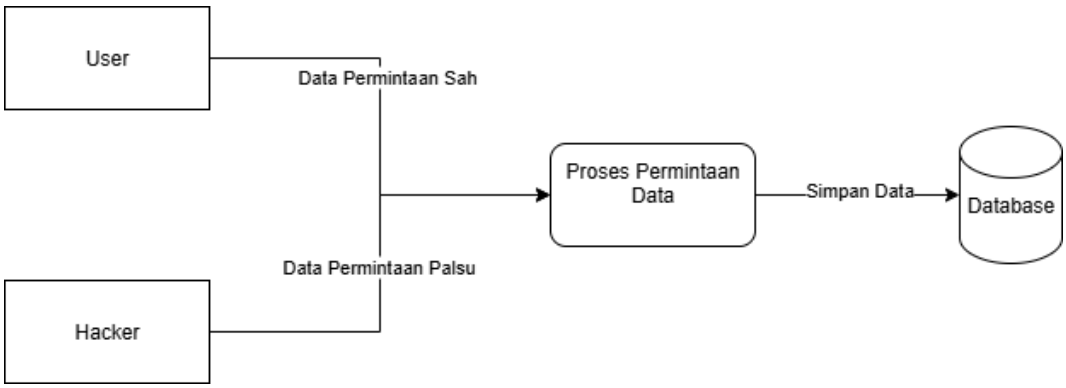
Gambar 2.5 Flowchart Perlindungan CSRF

Berdasarkan *flowchart* di atas, alur proses keamanan dimulai ketika server menerima sebuah *Request POST*. Pertama, sistem akan melakukan validasi lapisan pertama menggunakan mekanisme pertahanan CSRF bawaan Laravel, yaitu membandingkan *token* yang dikirim melalui *form*ulir dengan *token* yang tersimpan di sesi server. Jika *token* CSRF ini tidak valid, permintaan akan segera dihentikan dengan *Security exception*. Namun, jika valid, proses dilanjutkan ke lapisan keamanan kedua. Pada tahap ini, server akan memvalidasi *token* kustom HMAC-SHA256. Sistem akan membuat ulang *token* HMAC pembanding di sisi server dan mencocokkannya dengan *token* HMAC yang diterima dari *form*ulir. Hanya jika kedua *token* (CSRF dan HMAC) terbukti valid, maka permintaan akan dianggap sah dan diproses lebih lanjut untuk dieksekusi oleh aplikasi.

2.7. Data Flow Diagram (DFD)

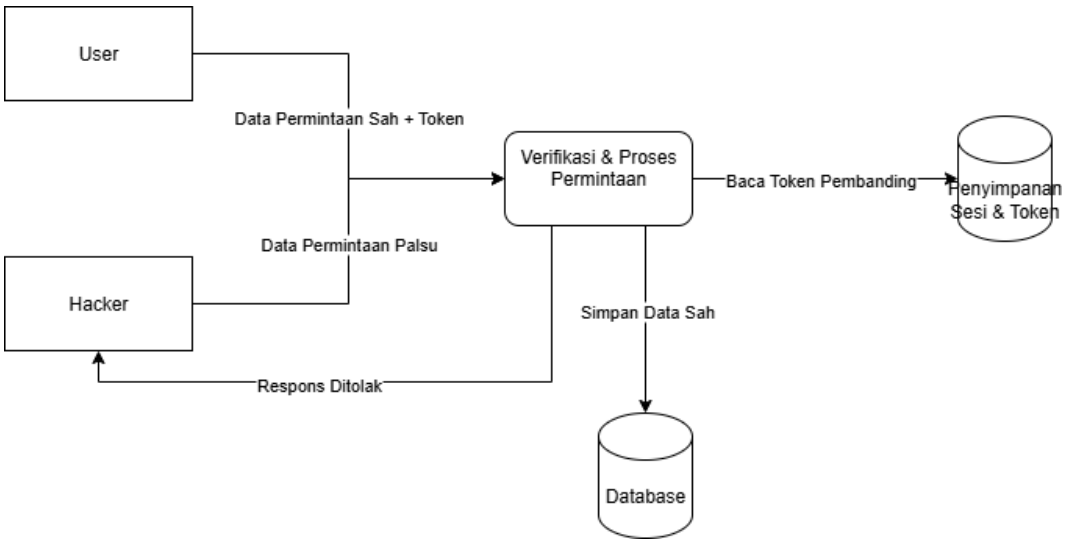
Data Flow Diagram (DFD), atau dikenal juga sebagai Diagram Alir Data, merupakan sebuah representasi grafik yang secara khusus menggambarkan aliran atau pergerakan in*form*asi di dalam sebuah sistem. Menurut (Hasanah & Untari, 2020), DFD mengilustrasikan bagaimana in*form*asi tersebut ditrans*form*asi atau diolah oleh sistem seiring data mengalir dari titik masukan (*input*) hingga menjadi keluaran (*output*). DFD juga dapat dipandang sebagai suatu metode untuk merancang sistem yang orientasinya terpusat pada alur data yang bergerak di dalam sistem tersebut. Lebih lanjut, DFD berfungsi sebagai teknik analisis untuk memvisualisasikan aliran masukan dalam sistem, bagaimana masukan tersebut diolah melalui berbagai proses, hingga akhirnya menghasilkan suatu keluaran; secara esensial, diagram ini memodelkan "apa yang terjadi" pada data di dalam sebuah sistem. DFD juga disebut sebagai alat perancangan sistem yang

berorientasi pada alur data dengan menggunakan konsep dekomposisi atau penguraian (Hasanah & Untari, 2020).



Gambar 2.6 DFD Alur Serangan CSRF (Tanpa Proteksi)

Pada Gambar 2.6, digambarkan alur data pada sistem yang rentan terhadap serangan CSRF. Baik *USER* maupun *HACKER* dapat mengirimkan aliran data ke Proses Permintaan Data. Tanpa adanya mekanisme verifikasi, sistem tidak dapat membedakan permintaan yang sah dan yang palsu, sehingga semua data akan langsung disimpan ke dalam *Database*. Ini menunjukkan bagaimana data dari pihak tidak berwenang dapat masuk ke sistem.



Gambar 2.7 DFD Alur Sistem dengan Proteksi HMAC-SHA256

Sementara itu, Gambar 2.7 menunjukkan alur data pada sistem yang telah diimplementasikan proteksi. Setiap permintaan yang masuk akan melewati proses Verifikasi & Proses Permintaan. Proses ini akan mengambil data *token* dari Penyimpanan Sesi & *Token* untuk memvalidasi permintaan yang masuk. Permintaan sah dari USER yang menyertakan *token* valid akan lolos verifikasi dan datanya disimpan ke Database. Sebaliknya, permintaan palsu dari *HACKER* yang tidak memiliki *token* valid akan gagal diverifikasi dan sistem akan mengirimkan *Respons* Ditolak, sehingga serangan berhasil dicegah.

2.8. Black Box Testing

Pengujian *Black Box* (Kotak Hitam) adalah metode pengujian perangkat lunak yang berfokus pada spesifikasi fungsional, yaitu mengevaluasi kesesuaian fungsi masukan dan keluaran (*input*-output) berdasarkan kebutuhan yang diharapkan, tanpa memperhatikan struktur kode internal atau cara kerja program di dalamnya. Tujuan utama metode ini adalah untuk menemukan kesalahan fungsional dan memastikan perangkat lunak berjalan sesuai harapan dari sudut pandang pengguna. Dalam pengujian *Black Box*, perangkat lunak diperlakukan sebagai 'kotak hitam', di mana penguji hanya perlu mengetahui apa yang seharusnya dilakukan oleh sistem, bukan bagaimana sistem melakukannya (Hasanah & Untari, 2020; Shaleh et al., 2021).

BAB III

METODOLOGI PENELITIAN

3.1. Jenis dan Pendekatan Penelitian

Penelitian ini merupakan penelitian terapan (*Applied Research*) yang berfokus pada pengembangan dan implementasi solusi teknis untuk mengatasi masalah keamanan siber. Secara spesifik, penelitian ini melakukan perancangan, pembangunan, dan pengujian implementasi keamanan *Website* dari serangan *Cross-Site Request Forgery* (CSRF) menggunakan algoritma HMAC-SHA256 pada *Framework* Laravel. Tujuan dari penelitian terapan ini adalah untuk mendapatkan pemahaman mendalam mengenai proses implementasi serta mengukur efektivitas solusi yang dibangun dalam skenario dunia nyata.

Pendekatan penelitian yang digunakan adalah pendekatan kuantitatif. Pendekatan kuantitatif dipilih karena penelitian ini bertujuan untuk mengukur dan mengevaluasi keberhasilan implementasi sistem keamanan melalui data terukur yang diperoleh dari serangkaian pengujian. Sebagaimana penelitian terkait implementasi keamanan *Website* lainnya yang menggunakan metode pengujian terukur, data kuantitatif akan dikumpulkan melalui *Black Box Testing* yang tersistematis. Hasil dari pengujian ini, berupa data tentang keberhasilan atau kegagalan serangan CSRF sebelum dan sesudah implementasi, akan dianalisis secara kuantitatif untuk menarik kesimpulan mengenai efektivitas implementasi algoritma HMAC-SHA256 dalam menangkal serangan tersebut. Pengukuran kuantitatif ini penting untuk memberikan bukti empiris terhadap klaim keamanan yang dihasilkan dari implementasi.

3.2. Teknik Pengumpulan Data

Pengumpulan data dalam penelitian ini dilakukan untuk memperoleh in*form*asi yang relevan dan akurat guna mendukung proses perancangan, implementasi, pengujian, dan analisis hasil penelitian. Teknik pengumpulan data yang digunakan meliputi:

1. Studi Pustaka (*Literature Review*): Teknik ini dilakukan dengan mengumpulkan dan menelaah berbagai sumber literatur, termasuk buku, jurnal ilmiah, artikel, dokumentasi resmi *Framework* Laravel, serta sumber *onLine* terpercaya yang berkaitan dengan keamanan *Website*, serangan CSRF, algoritma kriptografi HMAC-SHA256, dan *Framework* Laravel. Data dari studi pustaka digunakan untuk membangun landasan teori yang kuat (seperti yang tersaji di Bab II), memahami konsep-konsep kunci, mengidentifikasi masalah penelitian, serta memilih metode perancangan, implementasi, dan pengujian yang sesuai.
2. Observasi: dilakukan dengan mengamati secara langsung proses pengembangan dan implementasi sistem keamanan *Website* pada *Framework* Laravel. Observasi mencakup pengamatan terhadap konfigurasi lingkungan pengembangan, penulisan kode implementasi algoritma HMAC-SHA256, serta integrasi fitur keamanan tersebut ke dalam alur kerja Laravel. Data hasil observasi digunakan untuk mendokumentasikan langkah-langkah teknis implementasi.
3. Pengujian (*Testing*): Pengujian merupakan teknik utama untuk mengumpulkan data mengenai efektivitas implementasi keamanan yang dibangun. Metode pengujian yang spesifik digunakan adalah *Black Box*

Testing. Data dikumpulkan dari hasil simulasi serangan CSRF terhadap *Website* yang telah diimplementasikan keamanannya. Data kuantitatif yang dicatat meliputi indikator keberhasilan atau kegagalan serangan (misalnya, apakah permintaan berbahaya berhasil dieksekusi atau diblokir) dan detail teknis lainnya yang relevan dari log pengujian. Data hasil pengujian ini krusial untuk evaluasi dan analisis di Bab IV.

Teknik pengumpulan data ini dipilih untuk memastikan kelengkapan data yang dibutuhkan dalam menganalisis, merancang, mengimplementasikan, dan menguji solusi keamanan yang diusulkan secara sistematis.

3.3. Teknik Analisis Data

Teknik analisis data yang digunakan dalam penelitian ini adalah analisis deskriptif kuantitatif. Setelah data terkumpul dari proses pengujian keamanan menggunakan metode *Black Box Testing* (seperti yang dijelaskan pada sub-bab 3.2), data tersebut akan diolah dan dianalisis secara kuantitatif.

Analisis data akan meliputi tahapan berikut:

1. Pengumpulan Data Pengujian: Data hasil dari setiap skenario pengujian *Black Box Testing* dicatat secara sistematis. Data ini mencakup apakah simulasi serangan CSRF berhasil mengeksekusi aksi berbahaya pada *Website* atau apakah serangan berhasil diblokir oleh sistem keamanan yang diimplementasikan.
2. Klasifikasi Hasil Pengujian: Setiap hasil pengujian diklasifikasikan ke dalam kategori keberhasilan serangan atau kegagalan serangan (diblokir).
3. Perhitungan Kuantitatif: Dilakukan perhitungan untuk mendapatkan angka-angka atau persentase, misalnya:

a. Jumlah total skenario serangan yang diuji.
b. Jumlah skenario serangan yang berhasil (sebelum implementasi, jika diuji).
c. Jumlah skenario serangan yang berhasil setelah implementasi keamanan HMAC-SHA256.
d. Jumlah skenario serangan yang gagal (diblokir) setelah implementasi.
e. Persentase keberhasilan atau kegagalan serangan setelah implementasi.

4. Interpretasi Hasil: Angka-angka atau persentase yang diperoleh diinterpretasikan untuk mengevaluasi seberapa efektif implementasi keamanan menggunakan HMAC-SHA256 pada *Framework* Laravel dalam mencegah serangan CSRF. Analisis ini bertujuan untuk menjawab pertanyaan penelitian mengenai efektivitas solusi yang diusulkan dan tantangan teknis implementasinya.

Teknik analisis data ini dipilih karena sesuai dengan pendekatan kuantitatif penelitian dan memungkinkan pengukuran yang jelas terhadap dampak implementasi keamanan berdasarkan hasil pengujian yang terukur.

3.4. Spesifikasi Sistem dan Lingkungan Pengembangan

Penelitian ini dilakukan dalam sebuah lingkungan pengembangan yang terdiri dari spesifikasi perangkat keras dan perangkat lunak yang diuraikan di bawah ini.

3.4.1. Perangkat Keras (Hardware)

Spesifikasi perangkat keras minimum yang digunakan untuk membangun dan menguji aplikasi adalah sebagai berikut:

a. Prosesor: 12th Gen Intel Core i5-12450HX
b. RAM: 20 GB
c. Penyimpanan: SSD 512 GB
d. VGA: RTX 3050

3.4.2. Perangkat Lunak (Software)

Adapun perangkat lunak yang digunakan untuk implementasi dan pengujian sistem adalah sebagai berikut:

a. Sistem Operasi: Windows 11
b. *Web* Server: Server Pengembangan Lokal PHP (dijalankan melalui perintah php *artisan serve*).
c. Bahasa Pemrograman: PHP 8.2.12
d. *Framework*: Laravel *Framework* 12.21.0
e. Manajemen *Database*: MySQL/MariaDB (melalui XAMPP).
f. Editor Kode: *Visual Studio Code*

3.5. Studi Literatur

Bagian ini menyajikan tinjauan literatur dalam bentuk tabel yang merangkum beberapa penelitian terdahulu yang relevan dengan penelitian ini, khususnya terkait topik implementasi keamanan *Website* dari serangan CSRF, penggunaan algoritma HMAC-SHA256, dan pengujian terkait. Ringkasan ini berfokus pada hasil atau kontribusi utama dari studi-studi tersebut.

Tabel 3.1 Studi Literatur

No	Nama Peneliti	Judul Penelitian	Hasil
1	Wisnu Uriawan, Ray Ramadita, Rizky Dwi Putra, Rizqi Ilham, Risyad Addiva (2024)	*Authenticate and Verification Source Files using SHA256 and HMAC Algorithms*	Penelitian ini membahas tentang penerapan algoritma HMAC-SHA256 untuk verifikasi integritas data dan otentikasi file dalam sistem digital. Algoritma HMAC-SHA256 digunakan untuk memberikan perlindungan lebih pada aplikasi *web*, terutama dalam memastikan bahwa data yang diterima atau dikirimkan tetap terjaga keasliannya dan tidak mengalami perubahan yang tidak sah. Penerapan HMAC ini sangat relevan dalam konteks pengamanan aplikasi berbasis Laravel.
2	Puneet Kour (2020)	*A Study on Cross-Site Request Forgery Attack and its Prevention Measures*	Penelitian ini secara khusus mengkaji serangan *Cross-Site Request Forgery* (CSRF), bagaimana serangan ini bekerja, dan dampaknya pada aplikasi *web*. Solusi yang diusulkan termasuk penerapan CSRF *Token*s sebagai salah satu metode pencegahan yang efektif. Menggunakan CSRF *token*s untuk setiap permintaan yang dibuat oleh pengguna terbukti mengurangi risiko eksploitasi oleh penyerang yang memanfaatkan celah CSRF. Penelitian ini sangat mendalam dan relevan untuk penanggulangan CSRF pada aplikasi berbasis Laravel.
3	Dewa Ayu Mutiara Kirana Praba Dewi (2022)	Analisis Penggunaan HMAC-SHA256	Penelitian ini menganalisis penggunaan HMAC-SHA256 dalam menjaga

		pada Keamanan Aplikasi Chatting	integritas dan otentikasi pesan pada aplikasi chatting. Hasil penelitian menunjukkan bahwa penerapan algoritma ini sangat efektif dalam memastikan pesan yang dikirim melalui aplikasi tidak dimanipulasi oleh pihak ketiga. Metode HMAC-SHA256 terbukti lebih aman dalam melindungi data pribadi pengguna. Meskipun penelitian ini berfokus pada aplikasi chatting, konsep yang dibahas dapat diterapkan dalam konteks pengamanan *web* berbasis Laravel.
4	Ficry Cahya Ramdani, Alam Rahmatulloh, Rahmi Nur Shofa (2023)	Implementasi JSON *Web Token* pada *Authentication* dengan Algoritma HMAC SHA-256	Penelitian ini mengkaji penggunaan JWT (JSON *Web Token*) yang digabungkan dengan algoritma HMAC-SHA256 untuk meningkatkan keamanan sistem autentikasi dalam aplikasi berbasis *web*. Hasil penelitian ini menunjukkan bahwa implementasi HMAC-SHA256 dalam JWT dapat memperkuat proses otentikasi dan memberikan perlindungan yang lebih baik terhadap manipulasi *token*. Penerapan konsep ini relevan untuk membangun sistem autentikasi yang lebih aman pada aplikasi Laravel.
5	Chitra Sabapathy, Chelliah Srinivasan (2024)	*Enhanced Cloud Data Security by Employing HMAC for Advanced Cryptographic Protection*	Penelitian ini berfokus pada pengamanan data di *cloud* dengan menggabungkan algoritma SHA-256 dengan HMAC untuk meningkatkan perlindungan

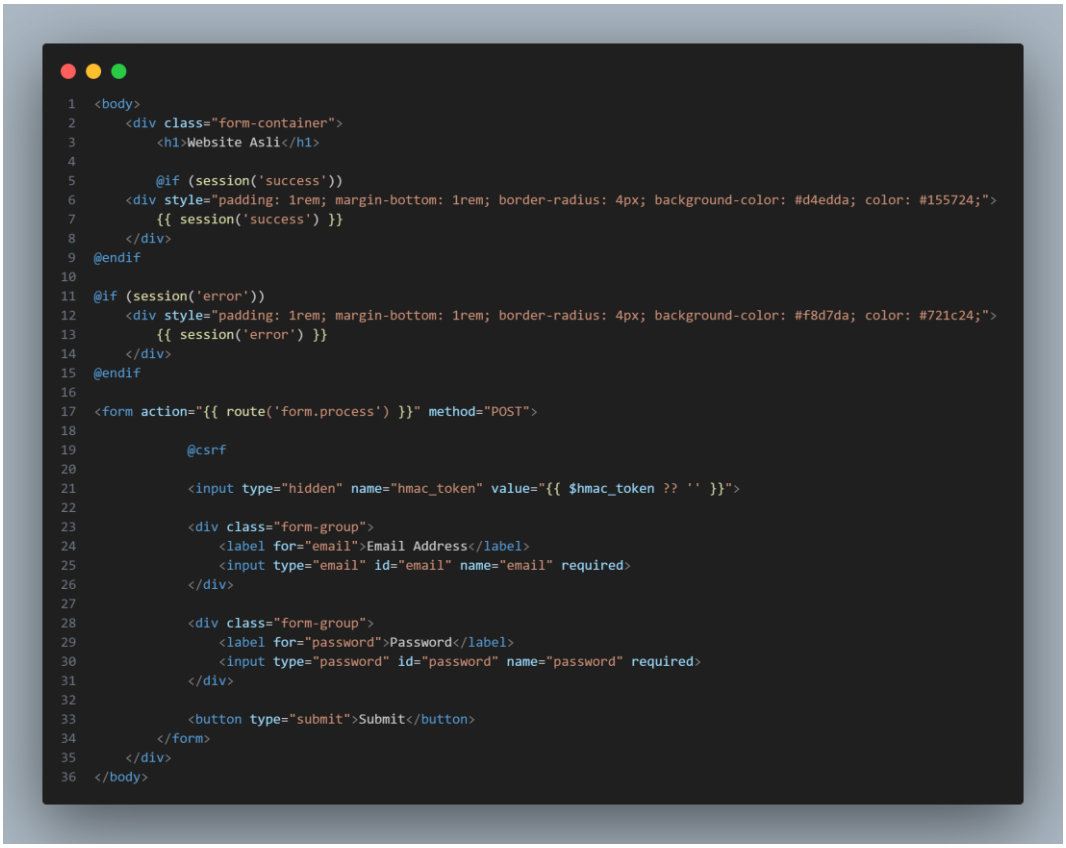
			terhadap data sensitif yang disimpan di *cloud*. Hasil dari penelitian ini menunjukkan bahwa penerapan HMAC bersama dengan algoritma *hash* lainnya memberikan lapisan tambahan dalam melindungi data dari serangan. Meskipun lebih berfokus pada *cloud*, konsep perlindungan data ini sangat relevan untuk aplikasi berbasis *web*, terutama dalam menjaga integritas dan kerahasiaan data yang ditransmisikan.

3.6. Perancangan

Bagian ini menjelaskan perancangan sistem *Website* yang dikembangkan untuk penelitian ini. Sistem dirancang untuk mendemonstrasikan dan menguji implementasi keamanan terhadap serangan *Cross-Site Request Forgery* (CSRF) menggunakan algoritma HMAC-SHA256 pada *Framework* Laravel. Perancangan sistem ini mengikuti pola arsitektur *Model-View-Controller* (MVC) dan mencakup desain antarmuka pengguna, logika pemrosesan data, serta struktur penyimpanan data. Detail perancangan untuk setiap komp*one*n sistem akan diuraikan dalam sub-sub-bab berikut.

3.6.1. Halaman Input

Halaman *input* ini merupakan antarmuka utama bagi pengguna untuk mengirimkan data. Halaman ini dirancang tidak hanya untuk menerima *input*, tetapi juga untuk menyertakan *token* keamanan yang diperlukan dan menampilkan umpan balik kepada pengguna setelah aksi dilakukan. Rancangan kode lengkap untuk antarmuka ini ditunjukkan pada Gambar 3.1.



```
<body>
    <div class="form-container">
        <h1>Website Asli</h1>

        @if (session('success'))
    <div style="padding: 1rem; margin-bottom: 1rem; border-radius: 4px; background-color: #d4edda; color: #155724;">
        {{ session('success') }}
    </div>
@endif

@if (session('error'))
    <div style="padding: 1rem; margin-bottom: 1rem; border-radius: 4px; background-color: #f8d7da; color: #721c24;">
        {{ session('error') }}
    </div>
@endif

<form action="{{ route('form.process') }}" method="POST">

            @csrf

            <input type="hidden" name="hmac_token" value="{{ $hmac_token ?? '' }}">

            <div class="form-group">
                <label for="email">Email Address</label>
                <input type="email" id="email" name="email" required>
            </div>

            <div class="form-group">
                <label for="password">Password</label>
                <input type="password" id="password" name="password" required>
            </div>

            <button type="submit">Submit</button>
        </form>
    </div>
</body>
```

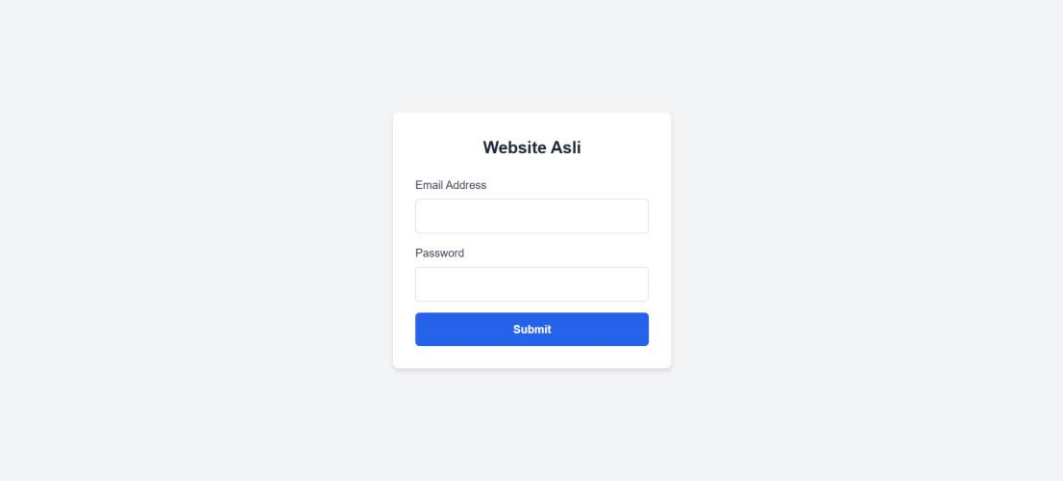
Gambar 3.1 Struktur Halaman Input

Berdasarkan Gambar 3.1, komponen-komponen utama dari halaman *input* ini adalah sebagai berikut:

1. Elemen *Form*: Menggunakan tag <*form*> dengan *method*="*POST*" untuk mengirimkan data. Atribut *action* diarahkan ke *route* bernama *form.process* yang akan ditangani oleh *FormController*.
2. *Token* Keamanan Berlapis: Terdapat dua *token* keamanan yang disisipkan. Pertama adalah *token* CSRF bawaan Laravel melalui direktif @csrf. Kedua adalah *token* kustom hmac_*token* yang nilainya dikirim dari *controller* dan disisipkan dalam sebuah *input* tersembunyi.
3. Field *Input* Pengguna: Terdiri dari kolom *input* standar untuk *email* dan *password*, beserta label yang sesuai.

4. Notifikasi Umpan Balik: Terdapat blok kondisional *@if*(*session*('*success*')) dan *@if*(*session*('*error*')) yang dirancang untuk menampilkan pesan kepada pengguna setelah mereka melakukan *submit form*.

Berdasarkan rancangan kode tersebut, antarmuka pengguna yang dihasilkan memiliki tampilan minimalis dan fungsional seperti yang ditunjukkan pada Gambar 3.2.



Gambar 3.2 Tampilan Halaman Input

3.6.2. Halaman Serangan CSRF

Untuk menguji sistem keamanan, dirancang sebuah halaman *web* statis yang berfungsi sebagai simulasi situs berbahaya. Halaman ini dirancang untuk menipu pengguna dan mencoba mengirimkan permintaan palsu ke aplikasi *web* asli. Rancangan kode untuk halaman simulasi serangan ini ditunjukkan pada Gambar 3.3.

```
<body>
    <div class="form-container">
        <h1>Website Tiruan/Palsu</h1>

        <form action="http://127.0.0.1:8000/form" method="POST">
            <div class="form-group">
                <label for="email">Email Address</label>
                <input type="email" id="email" name="email" required />
            </div>

            <div class="form-group">
                <label for="password">Password</label>
                <input
                    type="password"
                    id="password"
                    name="password"
                    required
                />
            </div>

            <button type="submit">Submit</button>
        </form>
    </div>
</body>
```

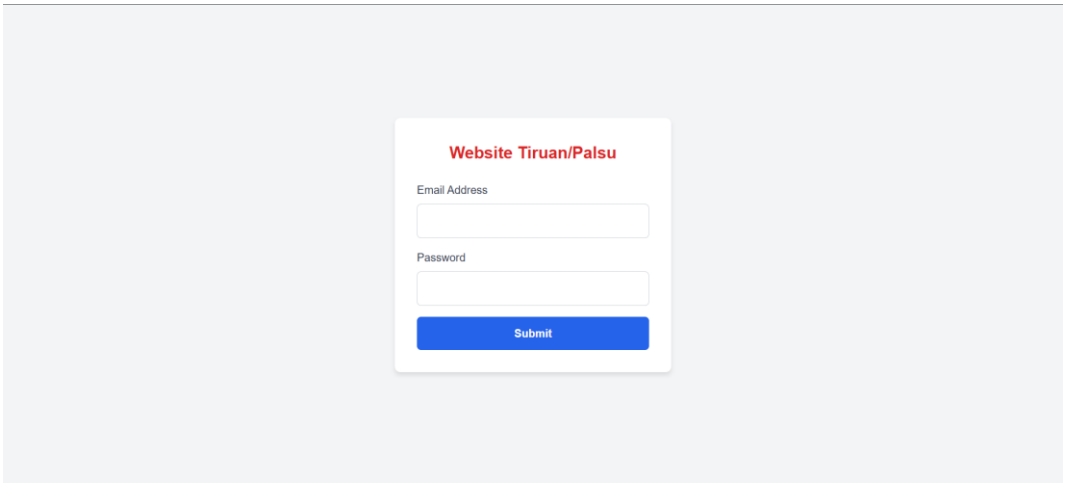
Gambar 3.3 Struktur Halaman Serangan CSRF

Berdasarkan Gambar 3.3, terdapat beberapa elemen kunci yang sengaja dirancang untuk melancarkan serangan CSRF:

1. Atribut *action*: Atribut *action* pada *form* diarahkan secara langsung ke URL *endpoint* pemrosesan data pada aplikasi *web* asli (http://127.0.0.1:8000/*form*). Hal ini bertujuan untuk mengirimkan permintaan lintas-situs (*cross-site*).
2. Ketiadaan *Token* Keamanan: Perbedaan paling krusial adalah halaman ini tidak menyertakan *token* CSRF bawaan Laravel maupun *token* kustom HMAC-SHA256. Penyerang tidak memiliki akses ke sesi pengguna yang valid, sehingga tidak dapat menghasilkan *token-token* tersebut.

3. Antarmuka Penipuan: Tampilan halaman dibuat semirip mungkin dengan halaman *input* asli untuk menipu pengguna agar secara sukarela memasukkan dan mengirimkan data.

Berdasarkan rancangan kode tersebut, dihasilkan antarmuka pengguna untuk halaman serangan seperti yang ditunjukkan pada Gambar 3.4.



Gambar 3.4 Tampilan Halaman Website Tiruan

3.6.3. Controller dan Function Input Data

Sub-bab ini menjelaskan perancangan kompo*nen* *Controller* pada *Framework* Laravel yang bertanggung jawab untuk menerima dan memproses permintaan yang dikirimkan dari *form input*, serta mengimplementasikan logika verifikasi keamanan berlapis. *Controller* berperan sebagai perantara antara *View* (Halaman *Input*) dan Model (untuk interaksi *database*).

Perancangan struktur dasar kode *Controller* yang menunjukkan alur logika penanganan permintaan ini dapat dilihat pada Gambar 3.5.

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
// Tambahkan 3 baris import di bawah ini
use Illuminate\Support\Facades\Session;
use Illuminate\Support\Facades\Log;
use App\Models\DataInputModel;

class FormController extends Controller
{
    /**
     * Metode untuk menampilkan halaman form.
     */
    public function showForm()
    {
        // 1. Ambil ID Sesi unik pengguna saat ini
        $sessionId = Session::getId();

        // 2. Ambil kunci rahasia dari file config
        $secretKey = config('app.hmac_secret_key');

        // 3. Buat token HMAC-SHA256
        $hmacToken = hash_hmac('sha256', $sessionId, $secretKey);

        // 4. Kirim token tersebut ke view 'form'
        return view('form', ['hmac_token' => $hmacToken]);
    }

    /**
     * Metode untuk memproses data yang dikirim dari form.
     */
    public function processForm(Request $request)
    {
        // --- TAHAP VALIDASI KEAMANAN HMAC-SHA256 ---
        $receivedHmacToken = $request->input('hmac_token');

        // Buat ulang token yang seharusnya di sisi server
        $sessionId = Session::getId();
        $secretKey = config('app.hmac_secret_key');
        $expectedHmacToken = hash_hmac('sha256', $sessionId, $secretKey);

        // Bandingkan token dari form dengan token yang seharusnya
        if (!$receivedHmacToken || !hash_equals($expectedHmacToken, $receivedHmacToken)) {
            // Jika tidak cocok, ini adalah percobaan serangan CSRF. Hentikan!
            Log::warning('CSRF Attack Detected (Invalid HMAC Token)'); // Catat di log
            abort(403, 'Invalid Security Token.');
        }

        // --- JIKA LOLOS, SIMPAN DATA KE DATABASE ---
        try {
            DataInputModel::create([
                'email' => $request->input('email'),
                'password' => bcrypt($request->input('password')) // Selalu hash password
            ]);

            return redirect()->route('form.show')->with('success', 'Data berhasil disimpan dengan aman!');
        } catch (\Exception $e) {
            Log::error('Gagal menyimpan data: ' . $e->getMessage());
            return redirect()->route('form.show')->with('error', 'Terjadi kesalahan pada server.');
        }
    }
}
```

Gambar 3.5 Struktur Kode Controller

Berdasarkan Gambar 3.5, *FormController* memiliki dua metode utama dengan fungsi sebagai berikut:

1. Metode *showForm*(): Metode ini bertanggung jawab untuk menangani permintaan *GET* dari pengguna untuk menampilkan halaman *form*. Di dalam metode ini, sistem secara dinamis menghasilkan *token* HMAC-SHA256 yang unik berdasarkan ID Sesi pengguna. *Token* tersebut kemudian dikirimkan ke *view* untuk disisipkan ke dalam *form*.

2. Metode *processForm*(): Metode ini menangani permintaan *POST* saat pengguna menekan tombol *submit*. Logika di dalamnya dirancang untuk melakukan validasi keamanan berlapis. Pertama, ia akan memvalidasi *token* HMAC-SHA256 yang diterima. Jika *token* valid, proses akan dilanjutkan ke blok try untuk menyimpan data ke *database*. Jika *token* tidak ada atau tidak valid, permintaan akan dihentikan dengan *respons error* 403, yang menandakan adanya percobaan serangan CSRF.

3.6.4. Models

Model dalam Laravel bertanggung jawab untuk berinteraksi dengan tabel *database* yang spesifik. Dalam penelitian ini, Data*Input*Model dirancang untuk mengelola semua operasi data (seperti penyimpanan) pada tabel data_*input_models*. Rancangan kode untuk Data*Input*Model ditunjukkan pada Gambar 3.6.

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class DataInputModel extends Model
{
    use HasFactory;

    protected $fillable = [
        'email',
        'password',
    ];
}
```

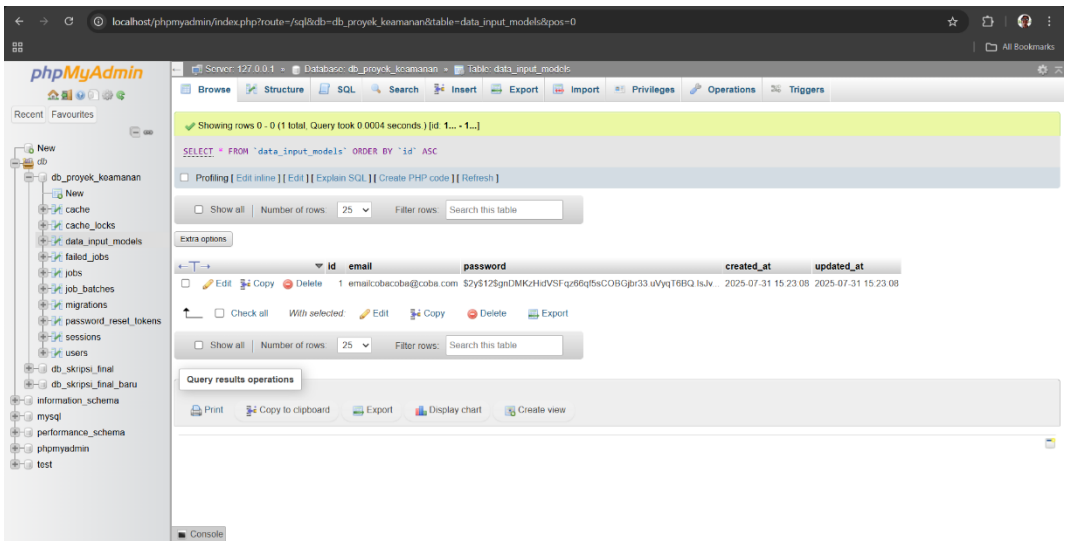
Gambar 3.6 Struktur Kode Models

Berdasarkan Gambar 3.6, Data*Input*Model mewarisi (*extends*) kelas Model bawaan Laravel, sehingga ia memiliki semua kemampuan untuk berinteraksi dengan *database*. Properti penting yang didefinisikan adalah:

a. *protected* $*fillable* = ['*email*', '*password*'];: Properti ini merupakan fitur keamanan Laravel yang disebut *mass assignment protection*. Dengan mendefinisikan *array* ini, kita hanya mengizinkan kolom *email* dan *password* untuk diisi secara massal melalui metode *create*(), sehingga melindungi tabel dari potensi penyisipan data yang tidak diinginkan pada kolom lain.

3.6.5. Database

Database berfungsi sebagai media penyimpanan persisten untuk data yang dikirimkan oleh pengguna setelah berhasil melewati validasi keamanan. Untuk penelitian ini, dirancang sebuah tabel bernama data_*input*_models yang bertugas untuk menyimpan data email dan password pengguna. Struktur dari tabel ini dapat dilihat pada Gambar 3.7.



Gambar 3.7 Field Database

Berdasarkan Gambar 3.7, rancangan tabel data_*input*_models memiliki struktur kolom sebagai berikut:

a. *id*: Kolom ini berfungsi sebagai kunci utama (*Primary Key*) untuk setiap baris data. Tipe datanya adalah BIGINT dan nilainya akan bertambah secara otomatis (*auto-increment*).
b. *email*: Kolom dengan tipe data VARCHAR(255) yang dirancang untuk menyimpan data email yang di*input* oleh pengguna.
c. *password*: Kolom dengan tipe data VARCHAR(255) yang dirancang untuk menyimpan kata sandi pengguna yang sudah dienkripsi (*hashed*).
d. *created_at* dan *updated_at*: Kolom *timestamps* yang secara otomatis dikelola oleh Laravel untuk mencatat waktu kapan sebuah data dibuat dan terakhir kali diperbarui.

BAB IV

HASIL DAN PEMBAHASAN

4.1. Implementasi Sistem

Pada bagian ini, akan dipaparkan secara teknis tahapan-tahapan dalam membangun sistem keamanan untuk menangani serangan *Cross-Site Request Forgery* (CSRF). Sistem ini diimplementasikan menggunakan *framework* PHP Laravel, yang kemudian akan diperkuat dengan mekanisme keamanan tambahan menggunakan algoritma HMAC-SHA256 sesuai dengan tujuan penelitian.

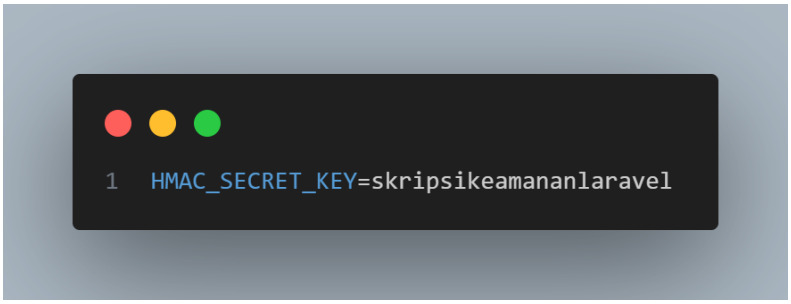
4.2. Implementasi Mekanisme Keamanan HMAC-SHA256

Tahapan ini menjelaskan implementasi teknis dari algoritma HMAC-SHA256 sebagai lapisan keamanan tambahan di dalam aplikasi Laravel. Implementasi ini mencakup empat langkah utama: konfigurasi kunci rahasia, pembangkitan *token*, penyisipan *token* ke dalam *form*, dan validasi *token* di sisi server.

4.2.1. Konfigurasi Kunci Rahasia

Langkah pertama dalam implementasi HMAC adalah mendefinisikan sebuah kunci rahasia (*secret key*) yang hanya diketahui oleh server. Kunci ini merupakan komponen krusial yang digunakan untuk menghasilkan dan memvalidasi *token*. Sesuai dengan praktik keamanan *modern*, kunci ini tidak disimpan langsung di dalam kode, melainkan didefinisikan sebagai variabel lingkungan di dalam *file* .env untuk mencegah kebocoran pada *source code repository*.

Baris berikut ditambahkan pada *file* .env:



Gambar 4.1 Konfigurasi Kunci Rahasia

Selanjutnya, untuk memudahkan pemanggilan di dalam aplikasi, kunci ini didaftarkan pada *file* konfigurasi config/app.php.

4.2.2. Pembangkitan Token HMAC

Token HMAC harus dibuat secara dinamis untuk setiap sesi pengguna yang unik. Proses ini dilakukan di dalam *FormController* pada metode *showForm*(), yang dieksekusi setiap kali halaman *form* akan ditampilkan kepada pengguna. Logika pembangkitan *token* adalah sebagai berikut: pertama, sistem mengambil ID Sesi unik milik pengguna. Kedua, ID Sesi tersebut digabungkan dengan kunci rahasia menggunakan fungsi *hash*_hmac() dengan algoritma sha256 untuk menghasilkan sebuah *token* yang unik untuk sesi tersebut. *Token* yang telah dihasilkan kemudian dikirimkan ke *view* untuk ditampilkan.

Kode implementasi pada app/Http/*Controllers*/*FormController*.php:

```php
public function showForm()
    {
        // 1. Ambil ID Sesi unik pengguna saat ini
        $sessionId = Session::getId();

        // 2. Ambil kunci rahasia dari file config
        $secretKey = config('app.hmac_secret_key');

        // 3. Buat token HMAC-SHA256
        $hmacToken = hash_hmac('sha256', $sessionId, $secretKey);

        // 4. Kirim token tersebut ke view 'form'
        return view('form', ['hmac_token' => $hmacToken]);
    }
```

Gambar 4.2 Pembangkitan Token HMAC

4.2.3. Penyisipan Token ke Dalam Form

Agar *token* dapat dikirimkan kembali ke server saat pengguna melakukan *submit*, *token* yang telah dibuat pada langkah sebelumnya disisipkan ke dalam *form* HTML sebagai sebuah *input* tersembunyi (*hidden input*). Hal ini memastikan *token* menjadi bagian dari data yang dikirimkan oleh pengguna tanpa terlihat di antarmuka.

Kode implementasi pada re*sources*/*views*/*form.blade*.php:

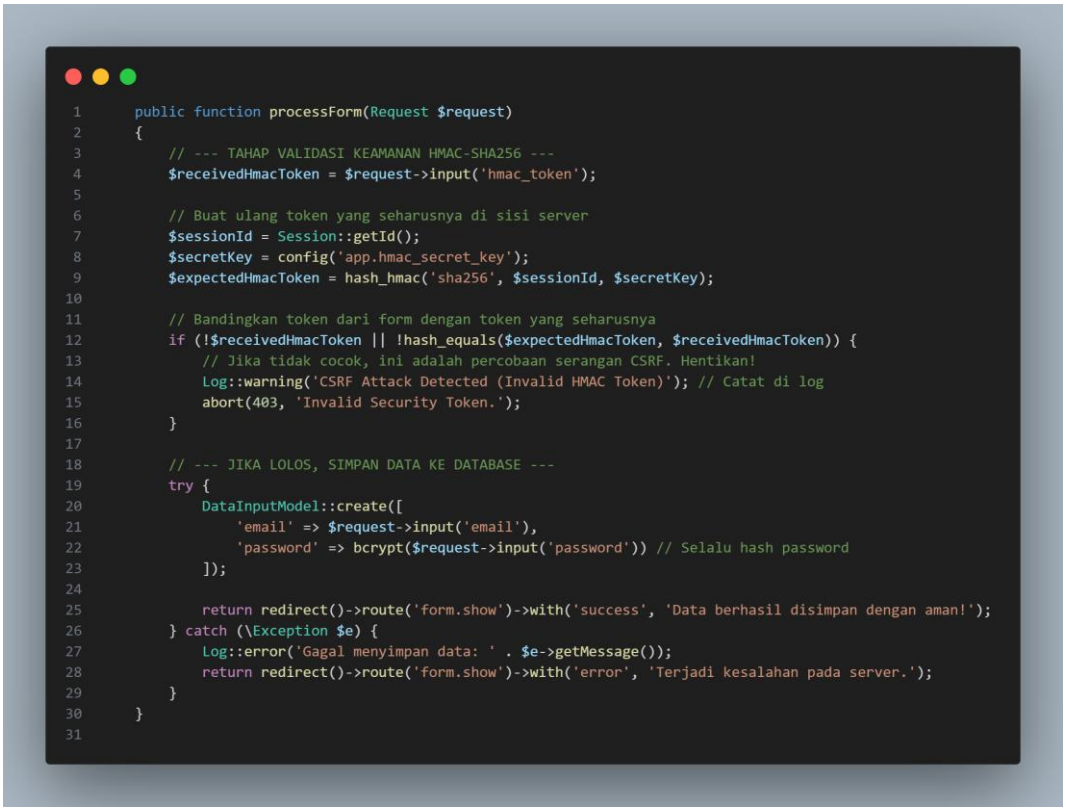
```
<form action="{{ route('form.process') }}" method="POST">

    @csrf

    <input type="hidden" name="hmac_token" value="{{ $hmac_token ?? '' }}">

    <div class="form-group">
        <label for="email">Email Address</label>
        <input type="email" id="email" name="email" required>
    </div>

    <div class="form-group">
        <label for="password">Password</label>
        <input type="password" id="password" name="password" required>
    </div>

    <button type="submit">Submit</button>
</form>

```

Gambar 4.3 Penyisipan Token

4.2.4. Validasi Token HMAC

Tahap validasi merupakan inti dari mekanisme pertahanan ini. Proses ini dijalankan di dalam *Form*Controller pada metode *processForm*() setiap kali server menerima permintaan *POST* dari *form*. Server akan melakukan validasi dengan cara membuat ulang *token* pembanding menggunakan ID Sesi dan kunci rahasia yang sama. Kemudian, *token* yang diterima dari *form* akan dibandingkan dengan *token* pembanding tersebut menggunakan fungsi *hash_equals*() yang aman dari serangan *timing attack*. Jika *token* yang diterima tidak ada atau tidak cocok, server akan menganggapnya sebagai percobaan serangan CSRF dan akan menghentikan permintaan dengan memberikan *respons error* 403 (*Forbidden*). Jika cocok, permintaan dianggap sah dan proses dilanjutkan.

Kode implementasi pada app/Http/*Controllers*/*Form*Controller.php:



```php
public function processForm(Request $request)
{
    // --- TAHAP VALIDASI KEAMANAN HMAC-SHA256 ---
    $receivedHmacToken = $request->input('hmac_token');

    // Buat ulang token yang seharusnya di sisi server
    $sessionId = Session::getId();
    $secretKey = config('app.hmac_secret_key');
    $expectedHmacToken = hash_hmac('sha256', $sessionId, $secretKey);

    // Bandingkan token dari form dengan token yang seharusnya
    if (!$receivedHmacToken || !hash_equals($expectedHmacToken, $receivedHmacToken)) {
        // Jika tidak cocok, ini adalah percobaan serangan CSRF. Hentikan!
        Log::warning('CSRF Attack Detected (Invalid HMAC Token)'); // Catat di log
        abort(403, 'Invalid Security Token.');
    }

    // --- JIKA LOLOS, SIMPAN DATA KE DATABASE ---
    try {
        DataInputModel::create([
            'email' => $request->input('email'),
            'password' => bcrypt($request->input('password')) // Selalu hash password
        ]);

        return redirect()->route('form.show')->with('success', 'Data berhasil disimpan dengan aman!');
    } catch (\Exception $e) {
        Log::error('Gagal menyimpan data: ' . $e->getMessage());
        return redirect()->route('form.show')->with('error', 'Terjadi kesalahan pada server.');
    }
}
```

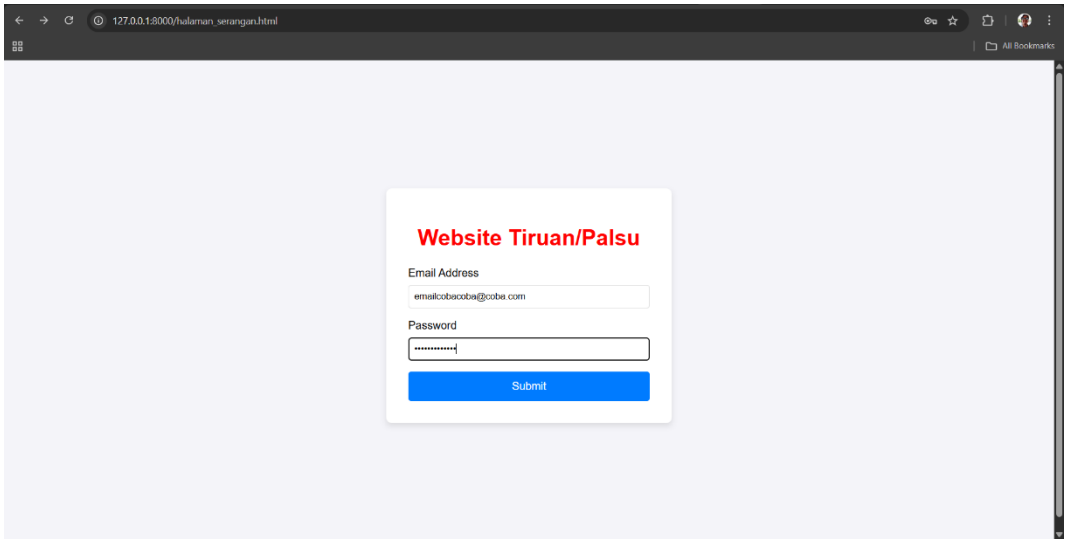
Gambar 4.4 Validasi Token HMAC

4.3. Skenario dan Prosedur Pengujian

Pengujian sistem dilakukan untuk mengukur efektivitas implementasi keamanan HMAC-SHA256 dalam menangkal serangan *Cross-Site Request Forgery*. Metode pengujian yang digunakan adalah *Black Box Testing*, di mana pengujian berfokus pada evaluasi fungsionalitas sistem (*input*-output) tanpa memperhatikan struktur kode internalnya. Prosedur pengujian dibagi menjadi empat skenario utama untuk mengevaluasi sistem secara komprehensif, mulai dari kondisi tanpa proteksi hingga kondisi dengan proteksi keamanan berlapis penuh.

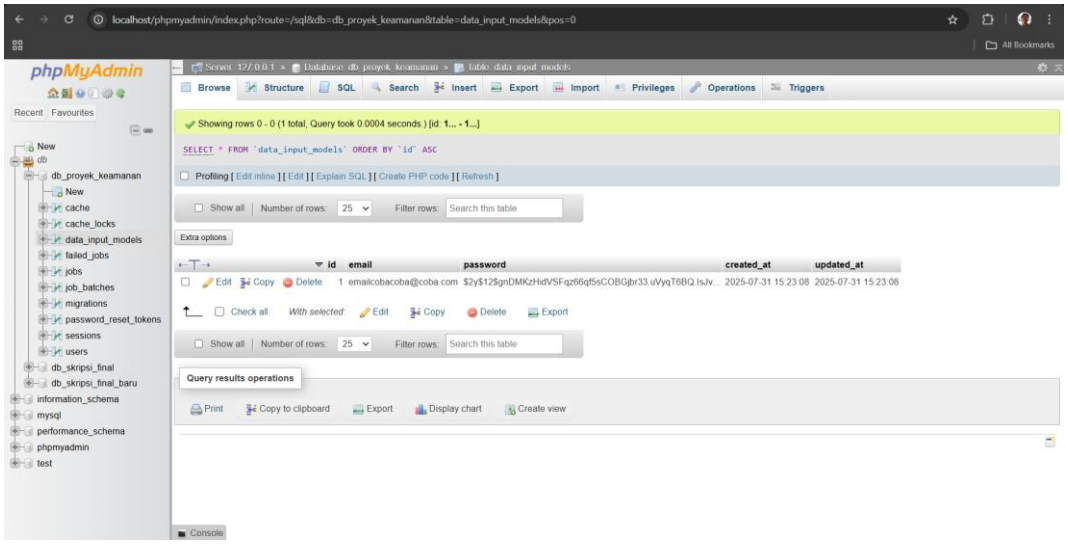
4.3.1. Skenario Uji 1: Serangan pada Sistem Tanpa Proteksi (Baseline)

Skenario pertama ini bertujuan untuk menetapkan kondisi dasar (*baseline*) dengan membuktikan bahwa aplikasi *web* rentan terhadap serangan CSRF jika tidak ada mekanisme keamanan yang aktif. Pada pengujian ini, proteksi CSRF bawaan Laravel dan lapisan keamanan HMAC-SHA256 dinonaktifkan. Serangan kemudian dilancarkan dengan melakukan *submit form* dari *web*site tiruan, seperti yang ditunjukkan pada Gambar 4.5.



Gambar 4.5 Tampilan Input pada Website Tiruan

Setelah *form* tersebut di-*submit*, permintaan berhasil diproses oleh server tanpa ada penolakan. Hasilnya, data dari *web*site tiruan tersebut berhasil tersimpan di dalam *database*, seperti yang dibuktikan pada Gambar 4.6.

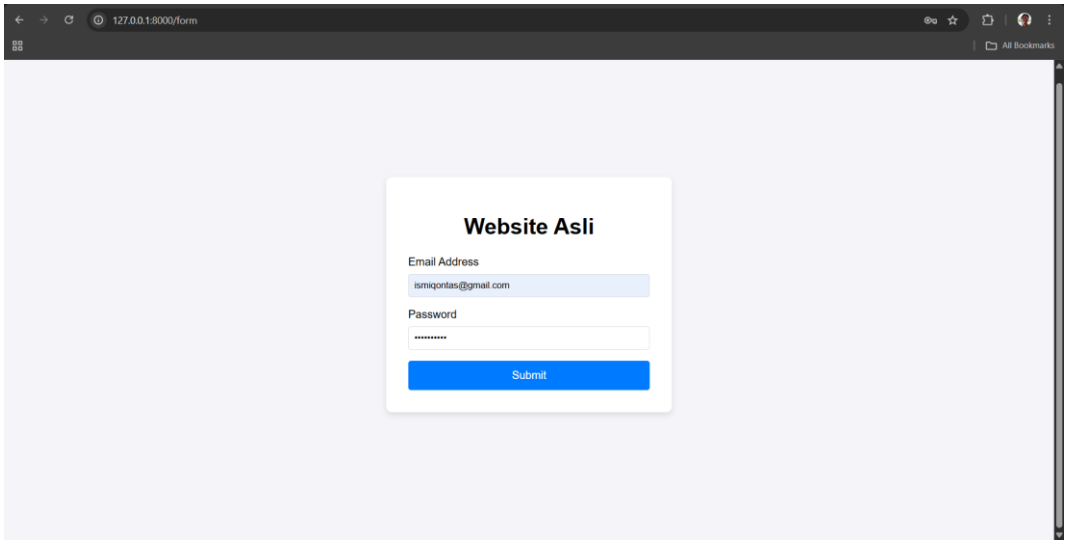


Gambar 4.6 Bukti Data dari Website Tiruan Masuk ke Database

Keberhasilan serangan ini mengonfirmasi bahwa tanpa adanya proteksi, permintaan berbahaya yang dikirimkan dari situs eksternal dapat dieksekusi oleh server, sehingga membuktikan adanya kerentanan CSRF pada sistem.

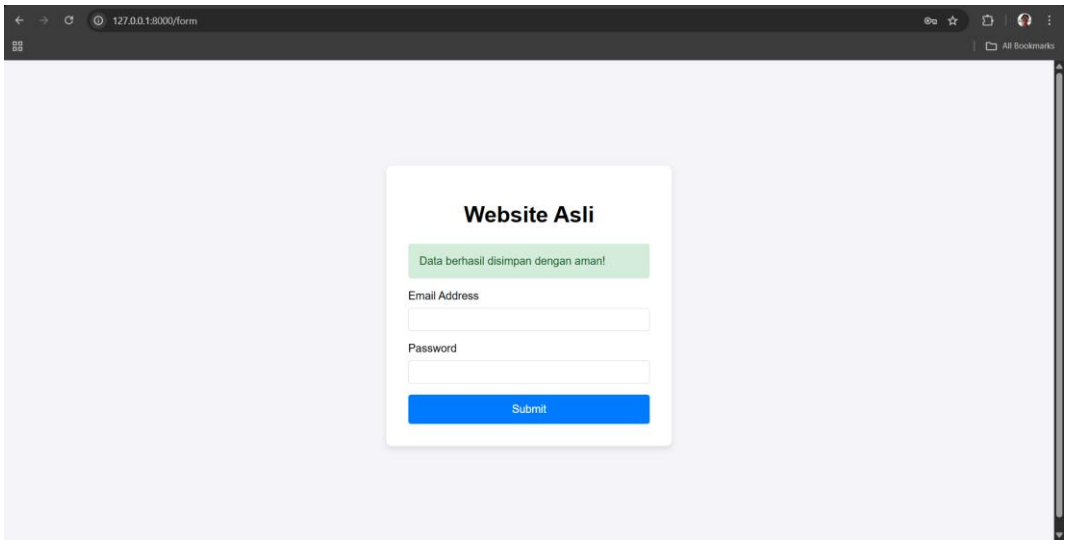
4.3.2. Skenario Uji 2: Fungsionalitas Normal dengan Proteksi Penuh

Skenario kedua bertujuan untuk memastikan bahwa implementasi keamanan yang ditambahkan tidak mengganggu fungsionalitas normal bagi pengguna yang sah. Pada pengujian ini, seluruh lapisan keamanan, yaitu proteksi CSRF bawaan Laravel dan validasi *token* HMAC-SHA256, diaktifkan. Pengujian dilakukan dengan mengirimkan data melalui *form input* pada aplikasi *web* asli, seperti yang terlihat pada Gambar 4.7.



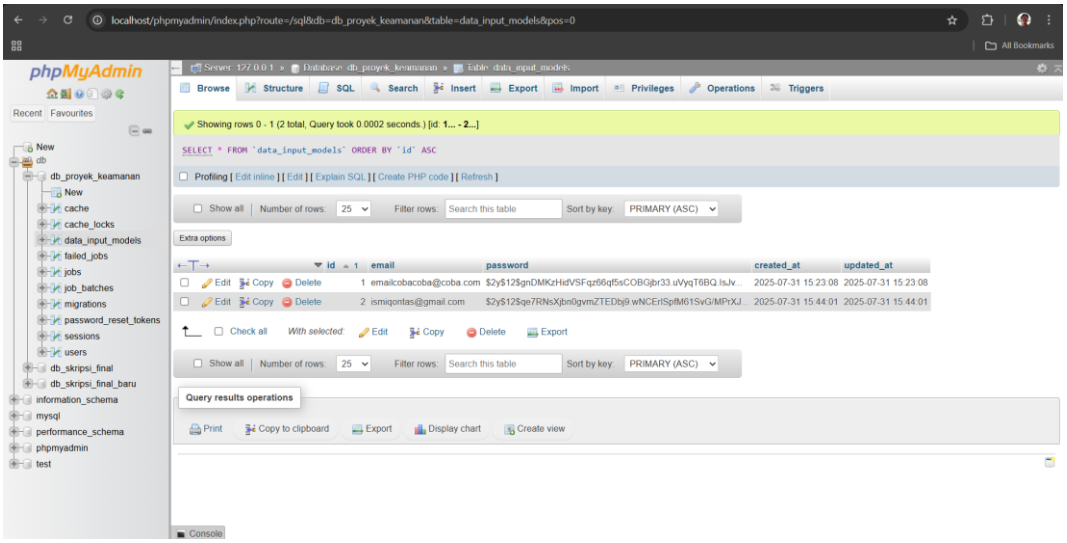
Gambar 4.7 Tampilan Input pada Aplikasi Web Asli

Setelah *form* di-*submit* dengan data yang valid, permintaan berhasil diproses oleh server dan sistem menampilkan pesan keberhasilan. Hasil ini ditunjukkan pada Gambar 4.8.



Gambar 4.8 Notifikasi Keberhasilan Penyimpanan Data

Untuk memverifikasi lebih lanjut, dilakukan pengecekan pada *database*. Terlihat bahwa data yang dikirimkan dari aplikasi *web* asli berhasil tersimpan, seperti yang dibuktikan pada Gambar 4.9.

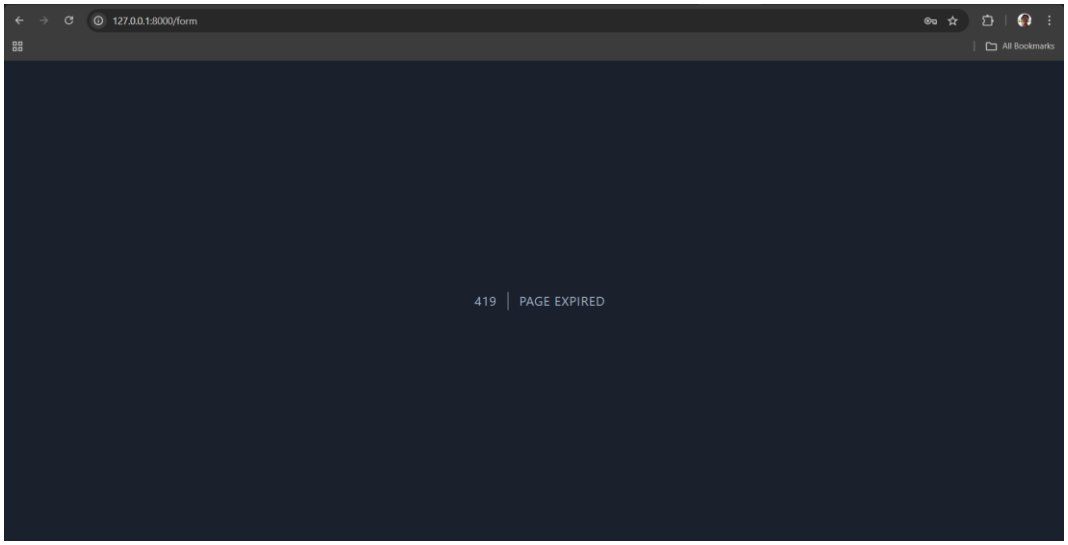


Gambar 4.9 Bukti Data dari Website Asli Masuk ke Database

Hasil pengujian pada skenario ini menunjukkan bahwa implementasi lapisan keamanan tambahan tidak memberikan dampak negatif pada alur kerja normal aplikasi. Pengguna yang sah tetap dapat mengirimkan data dengan lancar, yang menandakan bahwa sistem keamanan berjalan sesuai yang diharapkan tanpa mengorbankan fungsionalitas utama.

4.3.3. Skenario Uji 3: Serangan dengan Proteksi Penuh (Defense-in-Depth)

Skenario ketiga bertujuan untuk menguji efektivitas dari strategi pertahanan berlapis (*Defense-in-Depth*) saat semua mekanisme keamanan aktif. Pada pengujian ini, proteksi CSRF bawaan Laravel dan validasi *token* HMAC-SHA256 keduanya diaktifkan. Serangan dilancarkan dengan melakukan *submit form* dari *web*site tiruan. Hasil dari pengujian ini ditunjukkan pada Gambar 4.10

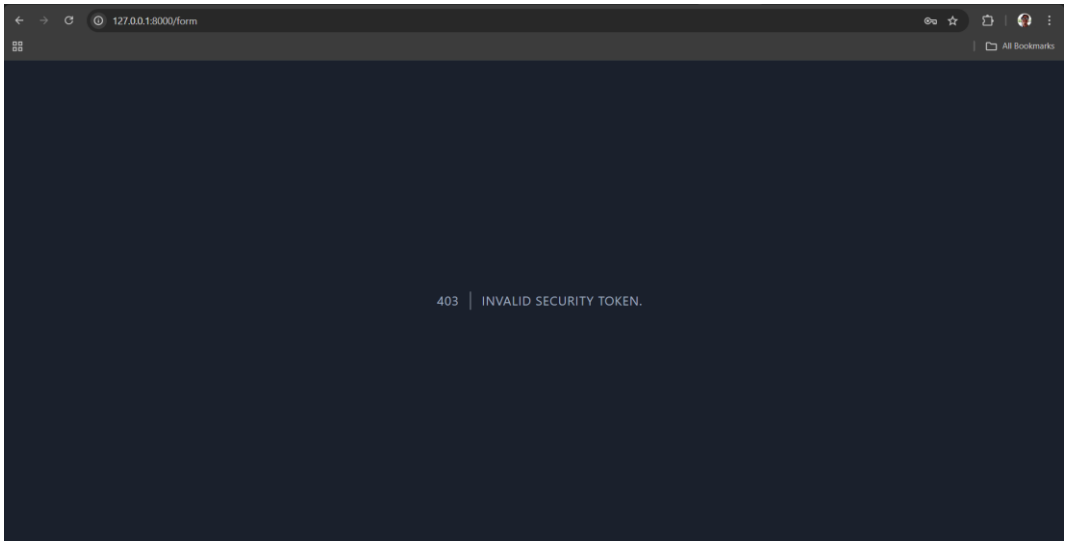


Gambar 4.10 Hasil Pengujian Skenario 3

Gambar 4.10 menunjukkan bahwa permintaan dari *web*site tiruan gagal dan diblokir oleh sistem. Halaman *error* 419 (*Page Expired*) yang ditampilkan merupakan *respons* standar dari *framework* Laravel ketika *token* CSRF bawaannya tidak valid atau tidak ada. Hal ini membuktikan bahwa lapisan keamanan pertama dari sistem berhasil mengidentifikasi dan menolak permintaan yang tidak sah tersebut sebelum sempat diproses lebih lanjut oleh lapisan keamanan kedua.

4.3.4. Skenario Uji 4: Serangan dengan Proteksi Lapisan Kedua (HMAC)

Skenario keempat bertujuan untuk mengisolasi dan membuktikan efektivitas lapisan keamanan HMAC-SHA256 secara mandiri. Prosedur pada skenario ini adalah dengan menonaktifkan proteksi CSRF bawaan Laravel, namun tetap mengaktifkan validasi *token* HMAC yang telah diimplementasikan. Untuk menjalankan Skenario Uji 4, proteksi CSRF bawaan Laravel dinonaktifkan terlebih dahulu melalui konfigurasi middleware pada file bootstrap/app.php. Serangan kembali dilancarkan dengan melakukan submit *form* dari website tiruan. Hasil dari pengujian ini ditunjukkan pada Gambar 4.11.



Gambar 4.11 Hasil Pengujian Skenario 4

Gambar 4.11 menunjukkan bahwa meskipun lapisan keamanan pertama dinonaktifkan, permintaan dari *web*site tiruan tetap gagal dan berhasil diblokir. Halaman *error* 403 dengan pesan "*Invalid Security Token*" yang ditampilkan merupakan *respons* yang telah didefinisikan secara kustom di dalam *Form*Controller. Hal ini membuktikan bahwa lapisan keamanan kedua, yaitu validasi *token* HMAC-SHA256, mampu berfungsi sebagai pertahanan yang independen dan efektif dalam mengidentifikasi serta menolak permintaan yang dipalsukan.

4.4. Hasil Pengujian

Berdasarkan empat skenario pengujian yang telah dilakukan pada sub-bab 4.2, hasil dari setiap pengujian dirangkum dalam tabel berikut. Penyajian hasil dalam *form*at tabel ini bertujuan untuk memberikan gambaran yang jelas mengenai efektivitas sistem keamanan yang diimplementasikan dalam berbagai kondisi. Pendekatan ini serupa dengan metode penyajian hasil pengujian yang dilakukan pada penelitian relevan sebelumnya.

Tabel 4.1 Ringkasan Hasil Pengujian Keamanan

No	Skenario Uji	Keamanan Laravel	Keamanan HMAC	Hasil	Keterangan
1	Serangan (Baseline)	Nonaktif	Nonaktif	Berhasil	Data dari situs tiruan berhasil masuk ke *database*.
2	Penggunaan Normal	Aktif	Aktif	Berhasil	Data dari situs asli berhasil disimpan.
3	Serangan (Lengkap)	Aktif	Aktif	Gagal	Permintaan diblokir oleh Laravel (*Error* 419).
4	Serangan (Lapis 1 Gagal)	Nonaktif	Aktif	Gagal	Permintaan diblokir oleh validasi HMAC (*Error* 403).

Tabel 4.1 di atas secara ringkas menunjukkan bahwa implementasi keamanan HMAC-SHA256 berhasil memenuhi tujuannya, yaitu memblokir serangan CSRF tanpa mengganggu fungsionalitas normal aplikasi *web*, dan mampu berfungsi sebagai lapisan pertahanan mandiri.

4.5. Pembahasan

Berdasarkan hasil pengujian yang telah disajikan, dapat dilakukan analisis mendalam mengenai efektivitas implementasi keamanan HMAC-SHA256. Setiap skenario pengujian memberikan wawasan spesifik terhadap postur keamanan aplikasi *web* yang dibangun.

Hasil dari Skenario Uji 1 secara empiris membuktikan adanya kerentanan kritis pada aplikasi *web* jika tidak dilengkapi dengan mekanisme proteksi CSRF. Keberhasilan serangan dalam skenario ini menunjukkan bahwa server secara

inheren akan memproses permintaan apa pun yang membawa *cookie* sesi yang valid, terlepas dari mana permintaan itu berasal. Hal ini sejalan dengan konsep dasar serangan CSRF yang telah dipaparkan pada Bab 2, di mana penyerang mengeksploitasi kepercayaan implisit antara server dan *browser* pengguna.

Skenario Uji 2 mengonfirmasi bahwa implementasi lapisan keamanan tambahan tidak mengganggu fungsionalitas normal aplikasi. Data yang dikirim dari pengguna sah melalui *form* asli berhasil diproses, menandakan bahwa mekanisme keamanan yang dibangun telah terintegrasi dengan baik tanpa menimbulkan dampak negatif pada pengalaman pengguna.

Selanjutnya, Skenario Uji 3 menunjukkan keberhasilan dari strategi pertahanan berlapis (*Defense-in-Depth*). Permintaan dari situs tiruan berhasil digagalkan oleh lapisan keamanan pertama, yaitu proteksi CSRF bawaan Laravel. Munculnya *error* 419 (*Page Expired*) adalah bukti bahwa mekanisme *Synchronizer Token Pattern* yang digunakan oleh Laravel efektif dalam mengidentifikasi permintaan tanpa *token* yang valid.

Puncak dari penelitian ini dibuktikan pada Skenario Uji 4. Dengan menonaktifkan proteksi bawaan Laravel, serangan berhasil melewati lapisan pertahanan pertama. Namun, permintaan tersebut tetap gagal karena berhasil diidentifikasi dan diblokir oleh lapisan keamanan kedua, yaitu validasi *token* HMAC-SHA256, yang ditandai dengan munculnya *error* 403. Hasil ini secara langsung menjawab rumusan masalah penelitian, membuktikan bahwa implementasi HMAC-SHA256 tidak hanya berfungsi sebagai lapisan tambahan, tetapi juga sebagai benteng pertahanan mandiri yang efektif. Keberhasilan ini

sejalan dengan teori HMAC yang menjamin integritas dan autentikasi permintaan melalui penggunaan kunci rahasia yang tidak dapat dipalsukan oleh penyerang.

Hasil dari Skenario 3 dan 4 secara kuantitatif menunjukkan tingkat keberhasilan 100% dalam memblokir serangan, yang dibuktikan dengan gagalnya permintaan dari situs tiruan pada kedua skenario pengujian tersebut. Secara keseluruhan, hasil pengujian ini menegaskan bahwa penerapan strategi pertahanan berlapis dengan mengombinasikan proteksi CSRF bawaan *framework* dan mekanisme *token* HMAC-SHA256 kustom secara signifikan meningkatkan ketahanan aplikasi *web* terhadap serangan *Cross-Site Request Forgery*.

BAB V

PENUTUP

5.1. Kesimpulan

Berdasarkan hasil implementasi dan pengujian yang telah dilakukan, maka dapat ditarik beberapa kesimpulan sebagai berikut:

1. Implementasi algoritma HMAC-SHA256 sebagai lapisan keamanan tambahan pada aplikasi *web* berbasis Laravel dapat dilakukan secara efektif dengan memanfaatkan ID Sesi pengguna dan kunci rahasia yang tersimpan di server.
2. Hasil pengujian membuktikan bahwa aplikasi *web* tanpa mekanisme proteksi CSRF yang memadai berada dalam kondisi rentan dan dapat dieksploitasi oleh permintaan yang berasal dari situs eksternal.
3. Penerapan *token* HMAC-SHA256 terbukti berhasil menjadi lapisan pertahanan mandiri yang mampu mengidentifikasi dan memblokir serangan CSRF, bahkan ketika proteksi bawaan *framework* dinonaktifkan.
4. Secara keseluruhan, penerapan strategi pertahanan berlapis (*Defense-in-Depth*) dengan mengombinasikan proteksi CSRF bawaan Laravel dan mekanisme HMAC-SHA256 secara signifikan meningkatkan ketahanan aplikasi *web* terhadap serangan *Cross-Site Request Forgery*.

5.2. Saran

Berdasarkan penelitian yang telah dilakukan, terdapat beberapa saran yang dapat diberikan untuk pengembangan selanjutnya:

1. Bagi Pengembang Aplikasi *Web*: Disarankan untuk tidak hanya mengandalkan fitur keamanan bawaan dari sebuah *framework*.

Menerapkan lapisan keamanan tambahan yang spesifik seperti *token* HMAC merupakan praktik "*Defense-in-Depth*" yang baik untuk meningkatkan keamanan sistem secara keseluruhan.

2. Bagi Peneliti Selanjutnya: Penelitian ini dapat dikembangkan lebih lanjut dengan menganalisis dampak implementasi keamanan ini terhadap per*forma* aplikasi pada skala lalu lintas yang tinggi. Selain itu, penelitian selanjutnya juga dapat membandingkan efektivitas metode *token* HMAC ini dengan metode mitigasi CSRF lainnya, seperti *SameSite Cookies* atau *Double Submit Cookies*.

DAFTAR PUSTAKA

Angkasa, B., Asriyanik, & Pambudi, A. (2023). *Implementasi Algoritma Hmac-Sha-256 Untuk Keamanan Kemasan Produk Implementation of Hmac-Sha-256 Algorithm for Product Packaging Security*. *20*(2), 1693–9166.

Ashari, I. F., Oktarina, V., Sadewo, R. G., & Damanhuri, S. (2022). Analysis of Cross Site Request Forgery (CSRF) Attacks on West Lampung Regency *Web*sites Using OWASP ZAP Tools. *Jurnal Sisfokom (Sistem Informasi Dan Komputer)*, *11*(2), 276–281. https://doi.org/10.32736/sisfokom.v11i2.1393

Budiman, A., Ahdan, S., & Aziz, M. (2021). Analisis Celah Keamanan Aplikasi *Web* E-Learning Universitas Abc Dengan Vulnerability Assesment. *Jurnal Komputasi*, *9*(2), 1–10. https://jurnal.fmipa.unila.ac.id/komputasi/article/view/2800

Calzavara, S., Conti, M., Rabitti, R. F. and A., & Tolomei, G. (2020). Machine Learning for *Web* Vulnerability Detection: The Case of Cross-Site Request Forgery. *IEEE Security and Privacy*, *18*(3), 8–16. https://doi.org/10.1109/MSEC.2019.2961649

Dewi, D. A. M. K. P. (2023). *Analisis Penggunaan HMAC-SHA256 pada Keamanan*
Aplikasi Chatting. *18220084*. https://in*form*atika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-dan-

Hasanah, F. N., & Untari, R. S. (2020). Buku Ajar Rekayasa Perangkat Lunak. In *Buku Ajar Rekayasa Perangkat Lunak*. https://doi.org/10.21070/2020/978-623-6833-89-6

Herzberg, A. (2025). Applied Introduction to Cryptography and Cyber*Security*. *Applied*
Introduction to Cryptography and CyberSecurity, *November 2023*. https://doi.org/10.1142/14014

Jalis, Auliana, S., & Permana, B. R. S. (2025). *Penerapan Framework Laravel Pada Aplikasi Nilai Siswa Berbasis Web Di Sdn Terumbu Kota Serang*. *9*(2), 3338–3342.

Kour, P. (2020). A Study On Cross-Site Request Forgery Attack And Its Prevention Measures. *InterNational Journal of Advanced Networking and Applications*, *12*(02), 4561–4566. https://doi.org/10.35444/ijana.2020.12204

Limbong, T., & Sriadhi. (2021). *Pemrograman Web Dasar*. https://books.google.co.id/books/about/Pemrograman_*Web*_Dasar.html?id=0pxLDwAAQBAJ&redir_esc=y

Lubis, M. M. M., Tommy, Handoko, D., & Wulan, N. (2022). Analisis Implementasi Laravel 9 Pada *Web*site E-Book Dalam Mengatasi N+1

Problem Serta Penyerangan Csrf dan Xss. *Jurnal Ilmu Komputer Dan Sistem Informasi (JIRSI)*, *2023*(2), 173–187. https://jurnal.unityacademy.sch.id/index.php/jirsi/index%0Ahttp://creativecommons.org/licenses/by-sa/4.0/

Ramdani, F. C., Rahmatulloh, A., & Shofa, R. N. (2023). Implementasi JSON *Web Token* pada Authentication dengan Algoritma HMAC SHA-256. *SISTEMASI: Jurnal Sistem Informasi*, *12*(1), 194–205. http://sistemasi.ftik.unisi.ac.id

Rana, M., Pandey, A., Mishra, A., & Kandu, V. (2023). Enhancing Data *Security*: A Comprehensive Study on the Efficacy of JSON *Web Token* (JWT) and HMAC SHA-256 Algorithm for *Web* Application *Security*. *InterNational Journal on Recent and Innovation Trends in Computing and Communication*, *11*(9), 4409–4416. https://doi.org/10.17762/ijritcc.v11i9.9930

Ranganathan, C. S., & Srinivasan, C. (2025). *Enhanced Cloud Data Security by Employing HMAC for Advanced Cryptographic Protection*. *1*(1), 1–10.

Rankothge, W. H., & Randeniya, S. M. N. (2020). *Identification and Mitigation tool for SQLIA*. 1–5.https://ieeexplore-ieee-org.recursos.biblioteca.upc.edu/stamp/stamp.jsp?tp=&arnumber=9342703&tag=1

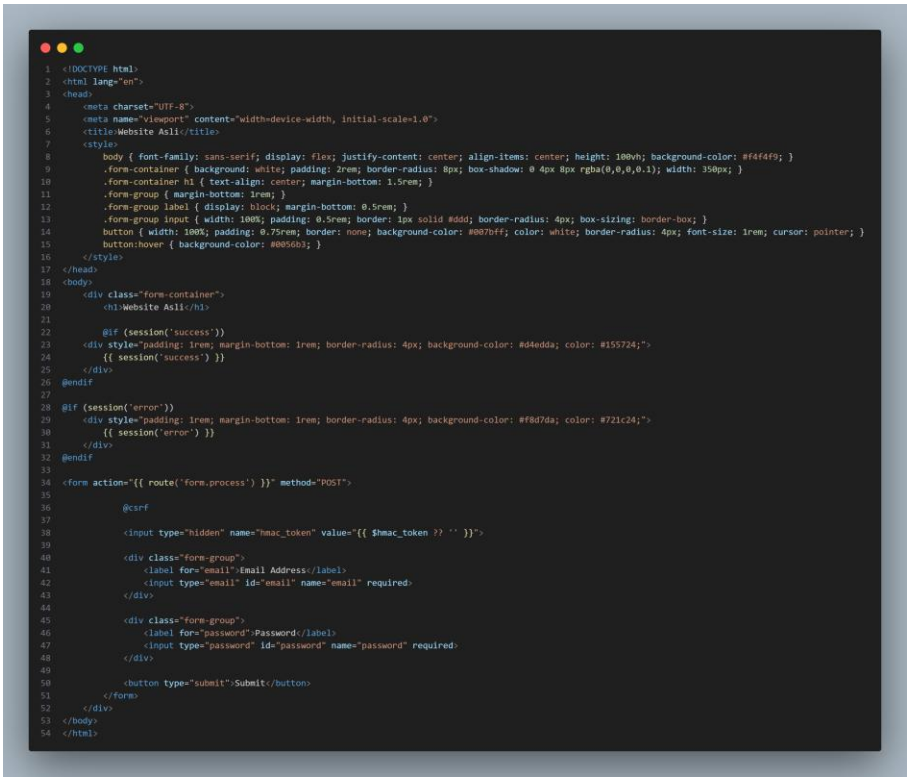
Rizki, M. A. K., & Ferico, A. (2021). Rancang Bangun Aplikasi E-Cuti Pegawai Berbasis *Web*site (Studi Kasus : Pengadilan Tata Usaha Negara). *Jurnal Teknologi Dan Sistem Informasi (JTSI)*, *2*(3), 1–13. http://jim.teknokrat.ac.id/index.php/JTSI

Sajjad, M., Mehmood, A., Saifullah, D. M., & Baig, J. I. (2024). Cross-Site Request Forgery Attacks. *Journal of Computers and Intelligent Systems*, *2*(2), 200–226. https://doi.org/10.1201/9781003373568-5

Suhaili, S., Julai, N., Sapawi, R., & Rajaee, N. (2024). Towards Maximising Hardware Re*source*s and Design Efficiency via High-Speed Implementation of HMAC based on SHA-256 Design. *Pertanika Journal of Science and Technology*, *32*(1), 31–44. https://doi.org/10.47836/pjst.32.1.02

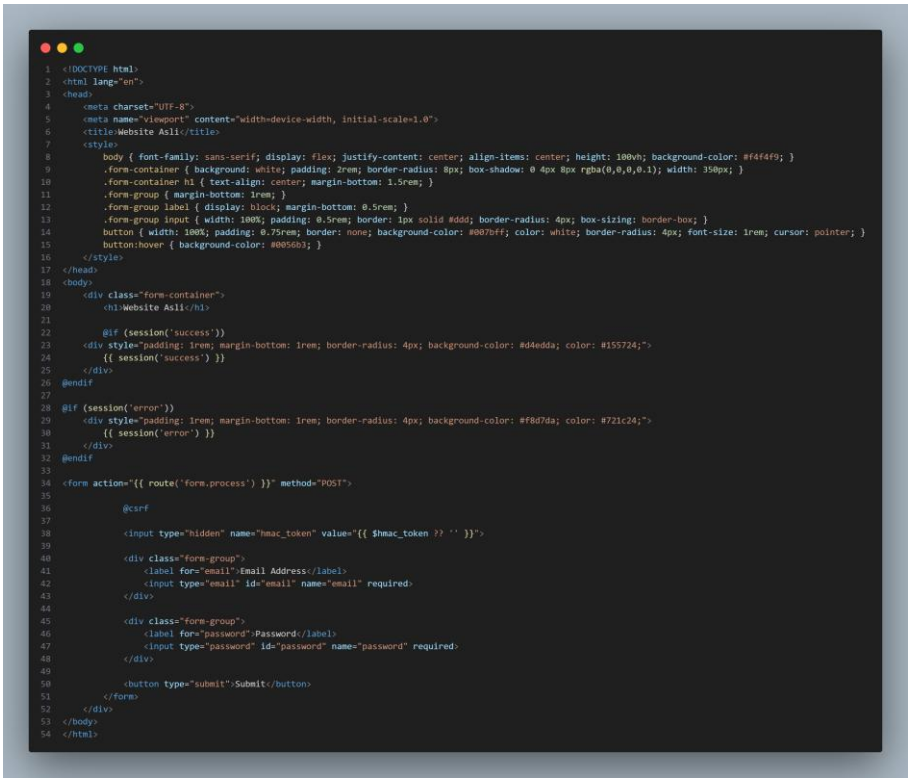
Uriawan, W., Ramadita, R., Putra, R. D., Siregar, R. I., & Addiva, R. (2024). *Authenticate and Verification Source Files using SHA256 and HMAC Algorithms*. https://doi.org/10.20944/preprints202407.0075.v1

LAMPIRAN



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Website Asli</title>
    <style>
        body { font-family: sans-serif; display: flex; justify-content: center; align-items: center; height: 100vh; background-color: #f4f4f9; }
        .form-container { background: white; padding: 2rem; border-radius: 8px; box-shadow: 0 4px 8px rgba(0,0,0,0.1); width: 350px; }
        .form-container h1 { text-align: center; margin-bottom: 1.5rem; }
        .form-group { margin-bottom: 1rem; }
        .form-group label { display: block; margin-bottom: 0.5rem; }
        .form-group input { width: 100%; padding: 0.5rem; border: 1px solid #ddd; border-radius: 4px; box-sizing: border-box; }
        button { width: 100%; padding: 0.75rem; border: none; background-color: #007bff; color: white; border-radius: 4px; font-size: 1rem; cursor: pointer; }
        button:hover { background-color: #0056b3; }
    </style>
</head>
<body>
    <div class="form-container">
        <h1>Website Asli</h1>

        @if (session('success'))
    <div style="padding: 1rem; margin-bottom: 1rem; border-radius: 4px; background-color: #d4edda; color: #155724;">
        {{ session('success') }}
    </div>
@endif

@if (session('error'))
    <div style="padding: 1rem; margin-bottom: 1rem; border-radius: 4px; background-color: #f8d7da; color: #721c24;">
        {{ session('error') }}
    </div>
@endif

<form action="{{ route('form.process') }}" method="POST">

            @csrf

            <input type="hidden" name="hmac_token" value="{{ $hmac_token ?? '' }}">

            <div class="form-group">
                <label for="email">Email Address</label>
                <input type="email" id="email" name="email" required>
            </div>

            <div class="form-group">
                <label for="password">Password</label>
                <input type="password" id="password" name="password" required>
            </div>

            <button type="submit">Submit</button>
        </form>
    </div>
</body>
</html>
```

***Source* Code Website Asli**

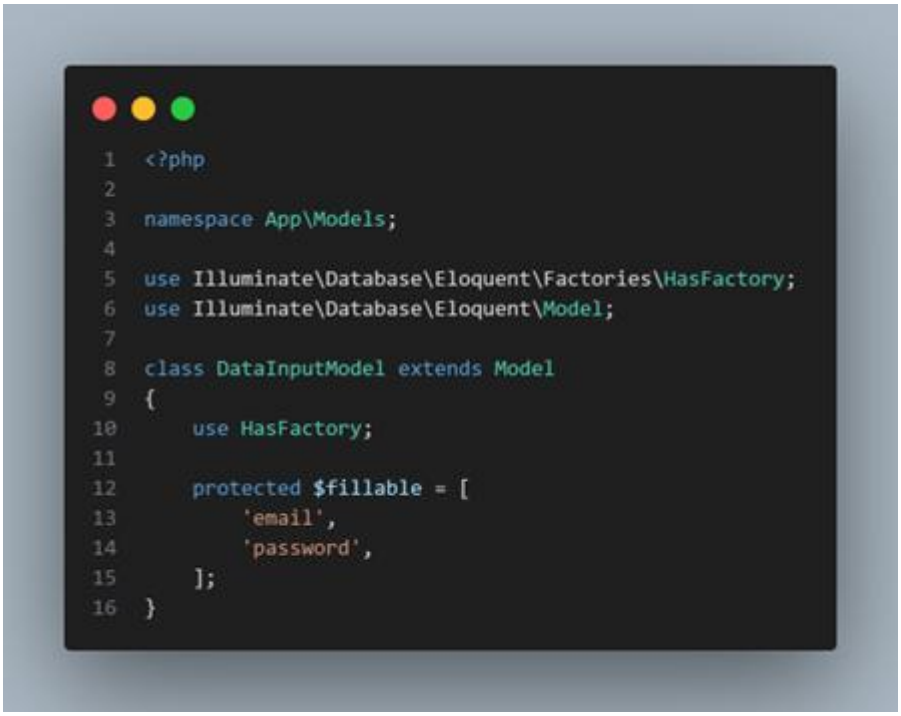


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Website Asli</title>
    <style>
        body { font-family: sans-serif; display: flex; justify-content: center; align-items: center; height: 100vh; background-color: #f4f4f9; }
        .form-container { background: white; padding: 2rem; border-radius: 8px; box-shadow: 0 4px 8px rgba(0,0,0,0.1); width: 350px; }
        .form-container h1 { text-align: center; margin-bottom: 1.5rem; }
        .form-group { margin-bottom: 1rem; }
        .form-group label { display: block; margin-bottom: 0.5rem; }
        .form-group input { width: 100%; padding: 0.5rem; border: 1px solid #ddd; border-radius: 4px; box-sizing: border-box; }
        button { width: 100%; padding: 0.75rem; border: none; background-color: #007bff; color: white; border-radius: 4px; font-size: 1rem; cursor: pointer; }
        button:hover { background-color: #0056b3; }
    </style>
</head>
<body>
    <div class="form-container">
        <h1>Website Asli</h1>

        @if (session('success'))
    <div style="padding: 1rem; margin-bottom: 1rem; border-radius: 4px; background-color: #d4edda; color: #155724;">
        {{ session('success') }}
    </div>
@endif

@if (session('error'))
    <div style="padding: 1rem; margin-bottom: 1rem; border-radius: 4px; background-color: #f8d7da; color: #721c24;">
        {{ session('error') }}
    </div>
@endif

<form action="{{ route('form.process') }}" method="POST">

            @csrf

            <input type="hidden" name="hmac_token" value="{{ $hmac_token ?? '' }}">

            <div class="form-group">
                <label for="email">Email Address</label>
                <input type="email" id="email" name="email" required>
            </div>

            <div class="form-group">
                <label for="password">Password</label>
                <input type="password" id="password" name="password" required>
            </div>

            <button type="submit">Submit</button>
        </form>
    </div>
</body>
</html>
```

***Source* Code Website Tiruan**

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Session;
use Illuminate\Support\Facades\Log;
use App\Models\DataInputModel;

class FormController extends Controller
{
    /**
     * Metode untuk menampilkan halaman form.
     */
    public function showForm()
    {
        // 1. Ambil ID Sesi unik pengguna saat ini
        $sessionId = Session::getId();

        // 2. Ambil kunci rahasia dari file config
        $secretKey = config('app.hmac_secret_key');

        // 3. Buat token HMAC-SHA256
        $hmacToken = hash_hmac('sha256', $sessionId, $secretKey);

        // 4. Kirim token tersebut ke view 'form'
        return view('form', ['hmac_token' => $hmacToken]);
    }

    /**
     * Metode untuk memproses data yang dikirim dari form.
     */
    public function processForm(Request $request)
    {
        // --- TAHAP VALIDASI KEAMANAN HMAC-SHA256 ---
        $receivedHmacToken = $request->input('hmac_token');

        // Buat ulang token yang seharusnya di sisi server
        $sessionId = Session::getId();
        $secretKey = config('app.hmac_secret_key');
        $expectedHmacToken = hash_hmac('sha256', $sessionId, $secretKey);

        // Bandingkan token dari form dengan token yang seharusnya
        if (!$receivedHmacToken || !hash_equals($expectedHmacToken, $receivedHmacToken)) {
            // Jika tidak cocok, ini adalah percobaan serangan CSRF. Hentikan!
            Log::warning('CSRF Attack Detected (Invalid HMAC Token)'); // Catat di log
            abort(403, 'Invalid Security Token.');
        }

        // --- JIKA LOLOS, SIMPAN DATA KE DATABASE ---
        try {
            DataInputModel::create([
                'email' => $request->input('email'),
                'password' => bcrypt($request->input('password')) // Selalu hash password
            ]);

            return redirect()->route('form.show')->with('success', 'Data berhasil disimpan dengan aman!');
        } catch (\Exception $e) {
            Log::error('Gagal menyimpan data: ' . $e->getMessage());
            return redirect()->route('form.show')->with('error', 'Terjadi kesalahan pada server.');
        }
    }
}

```

Kode Controller



```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class DataInputModel extends Model
{
    use HasFactory;

    protected $fillable = [
        'email',
        'password',
    ];
}
```

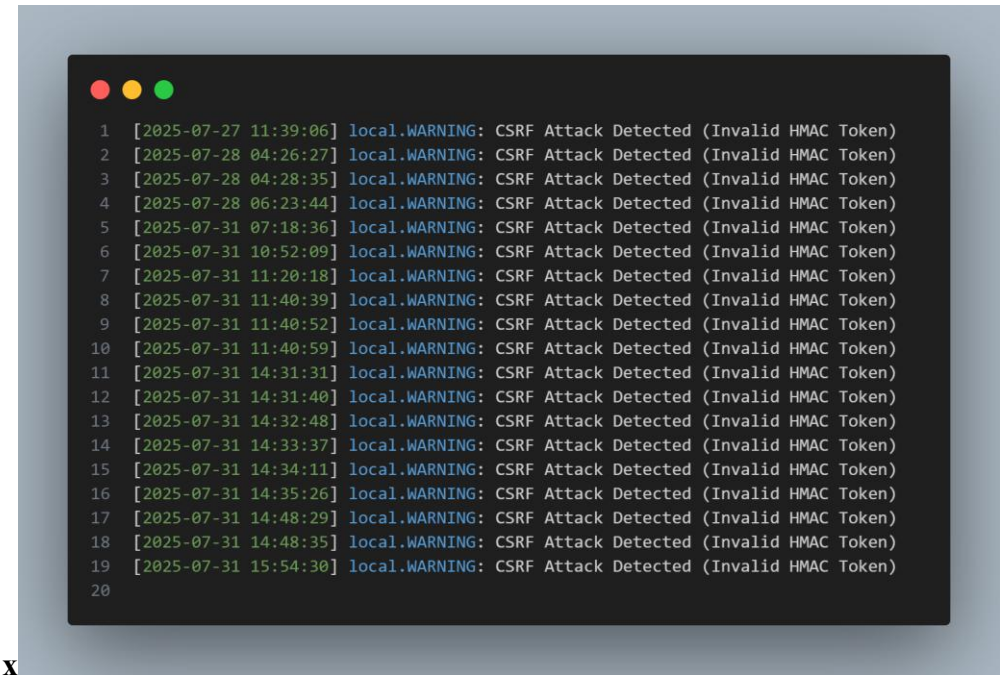
Kode Model

```php
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\FormController;

Route::get('/', function () {
    return view('welcome');
});

// Rute untuk menampilkan form input
Route::get('/form', [FormController::class, 'showForm'])->name('form.show');

// Rute untuk memproses data dari form
Route::post('/form', [FormController::class, 'processForm'])->name('form.process');

```

Kode Route

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('data_input_models', function (Blueprint $table) {
            $table->id();
            $table->string('email');
            $table->string('password');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('data_input_models');
    }
};

```

Kode Migration



```
[2025-07-27 11:39:06] local.WARNING: CSRF Attack Detected (Invalid HMAC Token)
[2025-07-28 04:26:27] local.WARNING: CSRF Attack Detected (Invalid HMAC Token)
[2025-07-28 04:28:35] local.WARNING: CSRF Attack Detected (Invalid HMAC Token)
[2025-07-28 06:23:44] local.WARNING: CSRF Attack Detected (Invalid HMAC Token)
[2025-07-31 07:18:36] local.WARNING: CSRF Attack Detected (Invalid HMAC Token)
[2025-07-31 10:52:09] local.WARNING: CSRF Attack Detected (Invalid HMAC Token)
[2025-07-31 11:20:18] local.WARNING: CSRF Attack Detected (Invalid HMAC Token)
[2025-07-31 11:40:39] local.WARNING: CSRF Attack Detected (Invalid HMAC Token)
[2025-07-31 11:40:52] local.WARNING: CSRF Attack Detected (Invalid HMAC Token)
[2025-07-31 11:40:59] local.WARNING: CSRF Attack Detected (Invalid HMAC Token)
[2025-07-31 14:31:31] local.WARNING: CSRF Attack Detected (Invalid HMAC Token)
[2025-07-31 14:31:40] local.WARNING: CSRF Attack Detected (Invalid HMAC Token)
[2025-07-31 14:32:48] local.WARNING: CSRF Attack Detected (Invalid HMAC Token)
[2025-07-31 14:33:37] local.WARNING: CSRF Attack Detected (Invalid HMAC Token)
[2025-07-31 14:34:11] local.WARNING: CSRF Attack Detected (Invalid HMAC Token)
[2025-07-31 14:35:26] local.WARNING: CSRF Attack Detected (Invalid HMAC Token)
[2025-07-31 14:48:29] local.WARNING: CSRF Attack Detected (Invalid HMAC Token)
[2025-07-31 14:48:35] local.WARNING: CSRF Attack Detected (Invalid HMAC Token)
[2025-07-31 15:54:30] local.WARNING: CSRF Attack Detected (Invalid HMAC Token)

```

x

Log Deteksi Serangan

```php
<?php

use Illuminate\Foundation\Application;
use Illuminate\Foundation\Configuration\Exceptions;
use Illuminate\Foundation\Configuration\Middleware;

return Application::configure(basePath: dirname(__DIR__))
    ->withRouting(
        web: __DIR__ . '/../routes/web.php',
        commands: __DIR__ . '/../routes/console.php',
        health: '/up',
    )
    ->withMiddleware(function (Middleware $middleware): void {
        $middleware->validateCsrfTokens(except: [
            '*'
        ]);
    })
    ->withExceptions(function (Exceptions $exceptions): void {
        //
    })->create();

```

Konfigurasi Aplikasi Inti

qontasturnitin-1757910085916

ORIGINALITY REPORT

21%	18%	8%	10%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	repository.umsu.ac.id Internet Source	4%
2	docplayer.info Internet Source	1%
3	id.123dok.com Internet Source	1%
4	eprints.universitasputrabangsa.ac.id Internet Source	1%
5	flexadia.com Internet Source	<1%
6	Submitted to Strategic Education Student Paper	<1%
7	ojs.stmik-banjarbaru.ac.id Internet Source	<1%
8	teknologiterkini.org Internet Source	<1%
9	Submitted to Liberty University Student Paper	<1%
10	www.theassignmenthelpline.com Internet Source	<1%
11	digilibadmin.unismuh.ac.id Internet Source	<1%
	repository.upnjatim.ac.id	

12	Internet Source	<1%
13	Taufik Ramadan Firdaus. "Keamanan Aplikasi Web Melalui Penerapan Cross Site Request Forgery(CSRF)", ITEJ (Information Technology Engineering Journals), 2016 Publication	<1%
14	doku.pub Internet Source	<1%
15	journal.aptii.or.id Internet Source	<1%
16	text-id.123dok.com Internet Source	<1%
17	www.termpaperwarehouse.com Internet Source	<1%
18	Submitted to Curtin University of Technology Student Paper	<1%
19	digilib.uns.ac.id Internet Source	<1%
20	dspace.uii.ac.id Internet Source	<1%
21	Muh Abu Nasher Aldikdar Aldi, Moh. Anshori Aris Widya Aris, Agus Sifaunajah Agus. "Aplikasi Metrologi Legal Berbasis Pelanggan Terintegrasi", Jurnal Pengembangan Teknologi Informasi dan Komunikasi (JUPTIK), 2024 Publication	<1%
22	etd.umy.ac.id Internet Source	<1%

23	Submitted to Universitas Muhammadiyah Purwokerto Student Paper	<1 %
24	repository.upi.edu Internet Source	<1 %
25	www.slideshare.net Internet Source	<1 %
26	Submitted to Universitas Raharja Student Paper	<1 %
27	adoc.pub Internet Source	<1 %
28	journal.budiluhur.ac.id Internet Source	<1 %
29	Submitted to Australasian Academy of Higher Education Student Paper	<1 %
30	Submitted to Forum Perpustakaan Perguruan Tinggi Indonesia Jawa Timur II Student Paper	<1 %
31	eprints.akakom.ac.id Internet Source	<1 %
32	jidt.org Internet Source	<1 %
33	jurnal.unprimdn.ac.id Internet Source	<1 %
34	repository.upnvj.ac.id Internet Source	<1 %
35	Submitted to Mountain Lakes High School Student Paper	<1 %

36	Submitted to Georgia Institute of Technology Main Campus Student Paper	<1 %
37	Submitted to Universitas Sebelas Maret Student Paper	<1 %
38	repository.unibi.ac.id Internet Source	<1 %
39	Mohamad Saiful Anwar Sri Rumini, Endang Sulistiyani. "Pengembangan Sistem Informasi LSP Indohusada Berbasis Web Menggunakan Design Science Research (DSR)", Journal of Information System,Graphics, Hospitality and Technology, 2025 Publication	<1 %
40	repo.itera.ac.id Internet Source	<1 %
41	repository.ung.ac.id Internet Source	<1 %
42	upcommons.upc.edu Internet Source	<1 %
43	Dhia Suhaila, Muhammad Karim Bachtiar, Tedi Kurniawan. "Ananlisis Vulnerabilitas dan Pengujian Terhadap Google Gruyere", Journal of Internet and Software Engineering, 2024 Publication	<1 %
44	Muhammad Amirul Mu'min, Zumhur Alamin, Fathir, Sahrul Ramadhan. "Uji Penetrasi Injeksi SQL terhadap Celah Keamanan Website XYZ menggunakan Tools SQLMap", Scientific : Journal of Computer Science and Informatics, 2024	<1 %

Publication

45	Muhammad Rizqy Ananda, Riya Widayanti. "Pengembagan dan Implementasi Modul Custom Odoo 18 Pada Cloud VPS", Jurnal Minfo Polgan, 2025 Publication	<1 %
46	Submitted to Universitas Pancasila Student Paper	<1 %
47	Submitted to Universitas Putera Batam Student Paper	<1 %
48	Valian Yoga Pudya Ardhana, Muhammad Taufiq Hidayat, Miftahul Jannah, Sumiati Sumiati, Puspita Rini, Nila Sari. "Implementasi RESTful API Pada Laravel dan Simulator IoT Wokwi Untuk Pengukuran Suhu dan Kelembaban Menggunakan Metode Waterfall", Arcitech: Journal of Computer Science and Artificial Intelligence, 2023 Publication	<1 %
49	Zuhri Halim. "PENERAPAN SISTEM INFORMASI AKADEMIK DENGAN METODE EXTREME PROGRAMMING", JSiI (Jurnal Sistem Informasi), 2021 Publication	<1 %
50	repository.unpas.ac.id Internet Source	<1 %
51	Submitted to Adtalem Global Education Student Paper	<1 %
52	Submitted to Harrisburg University of Science and Technology Student Paper	<1 %

53	Submitted to Universitas Multimedia Nusantara Student Paper	<1 %
54	fourtrezz.co.id Internet Source	<1 %
55	tif.unusida.ac.id Internet Source	<1 %
56	widuri.raharja.info Internet Source	<1 %
57	Submitted to LL DIKTI IX Turnitin Consortium Part II Student Paper	<1 %
58	id.scribd.com Internet Source	<1 %
59	imanagerpublications.com Internet Source	<1 %
60	libres.uncg.edu Internet Source	<1 %
61	repository.unsri.ac.id Internet Source	<1 %
62	sir.stikom.edu Internet Source	<1 %
63	xlescience.org Internet Source	<1 %
64	Riska Kurniyanto Abdullah, Muhammad Thariq Fudhail, Syamsul Mujahidin. "Penggunaan Snort dan Fail2ban sebagai IDS untuk Mengatasi Brute Force Attack dengan Notifikasi Telegram: Studi Kasus pada Institusi	<1 %

XYZ", Jurnal Sistem dan Teknologi Informasi (JustIN), 2024
Publication

65	Royyan Pandu Muhammad, Gibran El Ibrahim. "RANCANG BANGUN SISTEM PPDB ONLINE STUDI KASUS SMK MUHAMMADIYAH GAMPING MENGGUNAKAN METODE EXTREME PROGRAMMING", Jurnal Informatika dan Teknik Elektro Terapan, 2024 Publication	<1 %
66	docobook.com Internet Source	<1 %
67	eprints.ummi.ac.id Internet Source	<1 %
68	repository.unism.ac.id Internet Source	<1 %
69	www.hostinger.co.id Internet Source	<1 %
70	Abdullah Azzam Robbani, Bambang Harie Wiyono, Imam Haromain. "Design and Construction of Web-Based Warehouse System Application Using Laravel Framework with Agile Development Case Study at PT XYZ", Jurnal Informatika Terpadu, 2025 Publication	<1 %
71	Submitted to Politeknik Negeri Bandung Student Paper	<1 %
72	Submitted to Universitas Muhammadiyah Palembang Student Paper	<1 %
73	digilib.unila.ac.id	

	Internet Source	<1%
74	ejurnal.methodist.ac.id Internet Source	<1%
75	eprints.umk.ac.id Internet Source	<1%
76	repository.dinamika.ac.id Internet Source	<1%
77	seminar.iaii.or.id Internet Source	<1%
78	Henri Septanto, David Harto. "PEMBUATAN APLIKASI POINT OF SALE (POS) UNTUK AGEN AAN MOTOR BOGOR", Jurnal Informatika dan Teknik Elektro Terapan, 2024 Publication	<1%
79	aqi.co.id Internet Source	<1%
80	blog.akhdani.co.id Internet Source	<1%
81	cp27.web.id Internet Source	<1%
82	e-journal.polsa.ac.id Internet Source	<1%
83	johannessimatupang.wordpress.com Internet Source	<1%
84	publisherqu.com Internet Source	<1%
85	repository.ub.ac.id Internet Source	<1%

86	repository.unja.ac.id Internet Source	<1 %
87	student.blog.dinus.ac.id Internet Source	<1 %
88	www.pandora-jewelryclearance.us Internet Source	<1 %
89	www.researchgate.net Internet Source	<1 %
90	www.scribd.com Internet Source	<1 %
91	www.synnexmetrodata.com Internet Source	<1 %
92	Delia-Elena Ungureanu, Irina Aflori, Cosmin-Iulian Irimia. "Document Identity Provider: an Anti-Tempering Solution", Procedia Computer Science, 2024 Publication	<1 %

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	Off		