PROSIDING Seminar Nasional SISFOTEK ISSN 2597-3584 (Media Online)
Sistem Informasi dan Teknologi
website : www.seminar.iaii.or.id | email : seminar@iaii.or.id

Analisis Dan Perancangan Keamanan Frontend Dalam Aplikasi Web: XSS dan CSRF

Vincent Laurensius[1], Qori Halimatul Hidayah[2], Nixon Erzed[3], Binastya Anggara Sekti[4]
[1,2,4]Sistem Informasi, Ilmu Komputer, Esa Unggul
[3]Teknik Informatika, Ilmu Komputer, Universitas Esa Unggul
vincenthambaya@student.esaunggul.ac.id

Abstract

Definitely Secure Bank (DSB) is a web application designed to model digital bank financial transactions. In the early stages of development, this DSB application has several security vulnerabilities, including Cross-Site Scripting (XSS) attacks with a non-persistent type on the web help page and Cross-Site Request Forgery (CSRF) attacks on the financial transaction process. In the DSB application and other modern web applications, the most common vulnerabilities encountered are vulnerabilities to XSS and CSRF attacks. XSS attacks occur when someone successfully injects malicious javascript scripts into a web page, which can be executed from the user's browser. While CSRF attacks are attacks to trick users into sending unwanted requests to trusted websites. This study aims to analyze frontend security vulnerabilities on DSB and implement solutions to prevent them. The analysis is carried out by identifying vulnerable points in the application and evaluating their potential for exploitation. The proposed solution to prevent XSS attacks is to apply input sanitation to all user-entered data on the help page. Input sanitation will clean data from malicious scripts before being processed by the system. To prevent CSRF attacks, the proposed solution is to use CSRF tokens when making transactions on DSB. A CSRF token is an encrypted random value that is added to each HTTP request and verified by the server. Implementing these solutions can improve DSB security and prevent exploitation of XSS and CSRF attack vulnerabilities.

Keywords: SDLC, XSS, CSRF, CSRF token, input sanitation

Abstrak

Definietly Secure Bank (DSB) adalah aplikasi web yang dirancang untuk memodelkan transaksi keuangan bank digital. Pada tahap pengembangan awal, aplikasi DSB ini memiliki beberapa kerentanan keamanan, termasuk serangan Cross-Site Scripting (XSS) dengan tipe non-persistent pada halaman bantuan web dan serangan CrossSite Request Forgery (CSRF) pada proses transaksi keuangan. Pada aplikasi DSB maupun aplikasi web modern lainnnya, kerentanan yang paling umum dijumpai adalah kerentanan terhadap serangan XSS dan CSRF tersebut. Serangan XSS terjadi ketika seseorang berhasil menyuntikkan script javascript berbahaya ke halaman web, yang dapat dieksekusi dari browser pengguna. Sedangkan serangan CSRF adalah serangan untuk menipu pengguna agar mengirimkan permintaan yang tidak diinginkan ke situs web terpercaya. Penelitian ini bertujuan untuk menganalisis kerentanan keamanan frontend pada DSB dan mengimplementasikan solusi untuk mencegahnya. Analisis dilakukan dengan mengidentifikasi titik-titik rentan pada aplikasi dan mengevaluasi potensinya untuk dieksploitasi. Solusi yang diusulkan untuk mencegah serangan XSS adalah dengan menerapkan sanitasi input pada semua data yang dimasukkan pengguna pada halaman bantuan. Sanitasi input akan membersihkan data dari skrip berbahaya sebelum diproses sistem. Untuk mencegah serangan CSRF, solusi yang diusulkan adalah dengan menggunakan token CSRF pada saat melakukan transaksi pada DSB. Token CSRF adalah nilai acak terenkripsi yang ditambahkan ke setiap permintaan HTTP dan diverifikasi oleh server. Implementasi solusi-solusi ini dapat meningkatkan keamanan DSB dan mencegah eksploitasi kerentanan serangan XSS dan CSRF.

Kata kunci: SDLC, XSS, CSRF, CSRF TOKEN, sanitasi input

1. Pendahuluan

Di era yang saling terhubung seperti sekarang ini, aplikasi web telah menjadi bagian yang tak terpisahkan dari kehidupan sehari-hari kita, mulai dari belanja online hingga jejaring sosial. Namun, ketergantungan yang semakin meningkat pada aplikasi web juga membuatnya menjadi target utama serangan dunia maya. Serangan ini dapat menimbulkan kerentanan pada bagian frontend aplikasi web yang memungkinkan para penyerang untuk mendapatkan akses tidak sah ke data sensitif, memanipulasi sesi pengguna, dan bahkan merebut akun pengguna.

Frontend adalah bagian dari aplikasi web yang dilihat dan berinteraksi dengan pengguna. Ini termasuk semua elemen visual, seperti tata letak, gambar, dan teks, serta elemen interaktif, seperti tombol, formulir, dan menu [1]. Dalam hal ini, kita akan melihat contoh kerentanan Frontend yang terdapat pada aplikasi web Definietly

Secure Bank (DSB). Meskipun DSB sudah memakai session untuk mengautorisasi transaksi uang antar pengguna, kita akan melihat bagaimana kerentanan masih bisa di eksploitasi penyerang dengan melakukan penyerangan CSRF dan juga XSS.

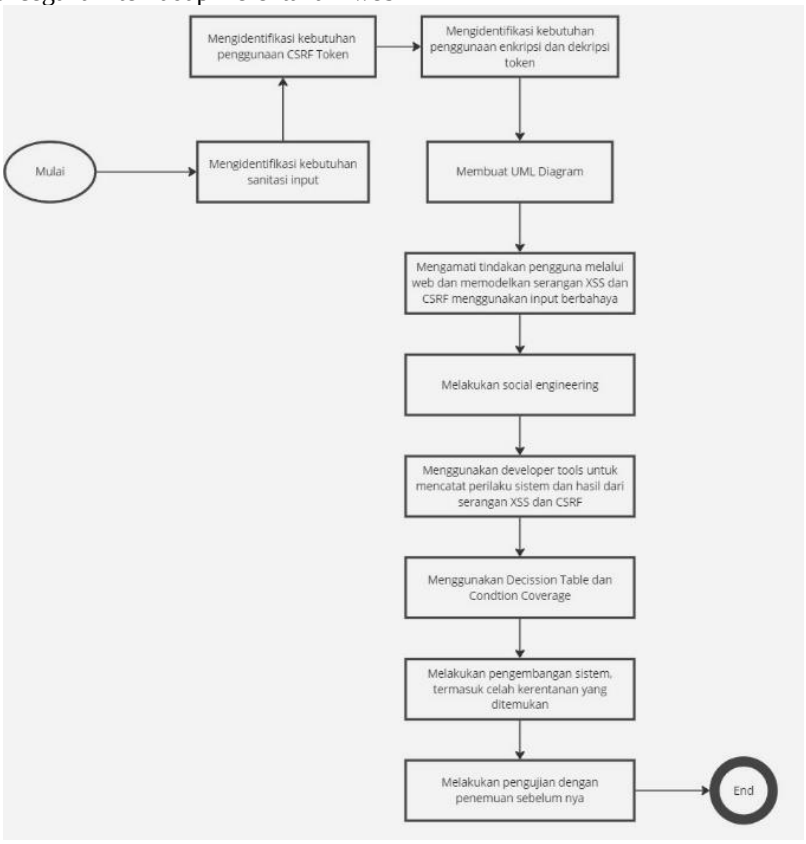
Cross-site scripting (XSS) XSS memungkinkan penyerang menyuntikkan skrip berbahaya ke halaman web. Skrip ini dapat mencuri data sensitif, mengarahkan pengguna ke situs web berbahaya, atau bahkan merebut sesi pengguna [2]. Cross-site request forgery (CSRF) menipu pengguna untuk mengirimkan permintaan yang tidak diinginkan ke situs web terpercaya. Permintaan ini dapat digunakan untuk melakukan tindakan atas nama pengguna, seperti mengganti kata sandi atau melakukan pembelian tanpa izin [3].

Terdapat beberapa langkah yang dapat diambil untuk mengurangi risiko keamanan frontend, antara lain: Sanitasi input, Token CSRF dan penggunaan framework modern yang secara default sudah mengimplementasikan keamanan-keamanan tersebut [3]. Tujuan dari penelitian ini adalah untuk menganalisis dan melakukan perancangan terhadap celah keamanan yang umum nya terdapat pada aplikasi frontend modern dalam kasus ini DSB, seperti umum nya XSS dan CSRF, serta melakukan penanggulangan keamanan dengan melakukan studi literatur. Melalui pengembangan ini, diharapkan dapat meningkatkan wawasan mengenai bagaimana cara mengidentifikasi dan melakukan pencegahan terhadap kerentanan web modern, yang nantinya akan memberikan pengalaman layanan yang lebih memuaskan bagi pengguna. Adapun identifikasi masalah nya, Aplikasi DSB juga mengalami kerentanan terhadap serangan CSRF, di mana penyerang dapat menggunakan social engineering untuk melakukan tindakan tanpa izin atas nama mereka dan berpotensi melakukan tindakan transaksional [4], serta Validasi input yang tidak memadai dapat menyebabkan masukan pengguna yang tidak aman, memungkinkan penyisipan kode berbahaya yang berdampak pada eksekusi skrip berbahaya di browser pengguna atau manipulasi tampilan halaman [4].

2. Metode Penelitian

Penelitian ini akan menggunakan pendekatan sistemik, di mana penulis akan secara langsung menguji situs web *Definitely Secure Bank (DSB)* dengan Teknik Observasi dan *Whitebox Testing*. Hasil dari pengujian tersebut akan dijadikan parameter masukan untuk melakukan *Blackbox Testing*, yang nantinya akan membantu penulis untuk merancang pengembangan sistem dengan sanitasi input dan token *CSRF* seperti pada Gambar 1.

Teknik Observasi, *Whitebox Testing*, dan *Blackbox Testing* akan memberikan gambaran lengkap dari proses *end-to-end* [5], bagaimana terjadinya kerentanan sistem, serta pencegahan terbaiknya. Adapun tahapan perencanaan penelitian sebagai berikut yaitu:



Gambar 1. Proses penelitian

Analisa kebutuhan: Kebutuhan dalam penelitian ini adalah penempatan code dan kondisi filter yang tepat dalam melakukan sanitasi input pada parameter *URL*, *input box*, tempat dimana pengguna memasukan input, juga *best practices rendering pattern* yang disarankan oleh *OWASP*. selain dari itu, penggunaan nilai dari *CSRF* Token pada *form* transaksi juga harus di atur sedemikian rupa, dalam hal ini penulis akan menggunakan *library crypto*, yang akan mengenerate token dengan gabungan huruf *capital* dan *lower case*, angka dan juga *symbol* secara acak, serta melakukan tambahan enkripsi dan dekripsi sehingga akan menyulitkan penyerang dalam mengeksploitasi sistem [6]. Penggantian token juga akan dilakukan dalam rentang 20 detik untuk menambah reliability dari keamanan token [6].

Pembuatan *UML Activity Diagram* untuk kerentanan *XSS* dan *CSRF* pada sistem berjalan, *Sequence Diagram* untuk kerentanan *XSS* pada sistem berjalan, Sequence Diagram untuk kerentanan *CSRF* pada sistem berjalan dan sistem usulan. Pembuatan *Activity Diagram* dan *Sequence Diagram* disesuaikan terhadap kerentanan *XSS* dan *CSRF* [7] [8], karena perancangan sistem untuk kerentanan *CSRF* mempunyai perubahan yang signifikan terhadap sistem, sedangkan perubahan perancangan sistem untuk kerentanan *XSS* adalah sama pada sistem berjalan.

Teknik Observasi dan Blackbox testing: Pengamatan penelitian ini dilakukan dengan cara memodelkan tindakan pengguna melalui sisi *UI* web dengan *input* yang berpotensi menimbulkan kerentanan pada web *DSB*. Teknik Blackbox testing digunakan penulis untuk bertindak sebagai penyerang dan melakukan social engineering melalui email kepada korban yang sedang login ke aplikasi web *DSB*. *Email* tersebut akan berisi *link* palsu yang, ketika diklik, akan mengeksekusi kode *JavaScript* untuk mentransfer uang secara acak. penulis akan menggunakan *developer tools* untuk mengamati dan mencatat perilaku web *DSB* dalam menerima input pada halaman bantuan dan transaksi, serta *HTTP request* yang terkait dengan transaksi tersebut. Teknik Observasi akan digunakan untuk mencatat perilaku sistem dalam menerima input tersebut dan output yang dihasilkan pada kerentanan *XSS* bertipe *non-persistent* dan *CSRF*. *Decision table* testing akan digunakan untuk mencatat kombinasi *input*, *output*, dan dampak kerentanan.

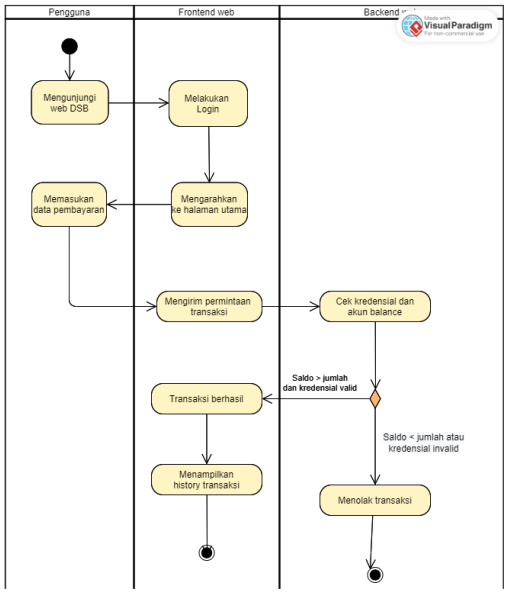
Teknik Whitebox testing: Pengamatan penelitian ini akan ditekankan pada sisi codebase dari web *DSB* mengguanakan metode *Condtion Coverage*. Metode *Condtion Coverage* merupakan salah satu teknik *Whitebox testing* yang memastikan bahwa setiap kondisi fungsionalitas program dieksekusi setidaknya sekali selama pengujian.untuk menguji seluruh kode pada aplikasi, agar menghasilkan nilai *TRUE* atau *FALSE* [9]. Hasil nilai tersebut akan ditentukan berdasarkan fungsionalitas sistem terhadap input dari pengguna, Dimana nilai *TRUE* dihasilkan jika aplikasi menghasilkan output yang sesuai, dan nilai *FALSE* jika aplikasi tidak menghasilkan output yang sesuai.

3. Hasil dan Pembahasan

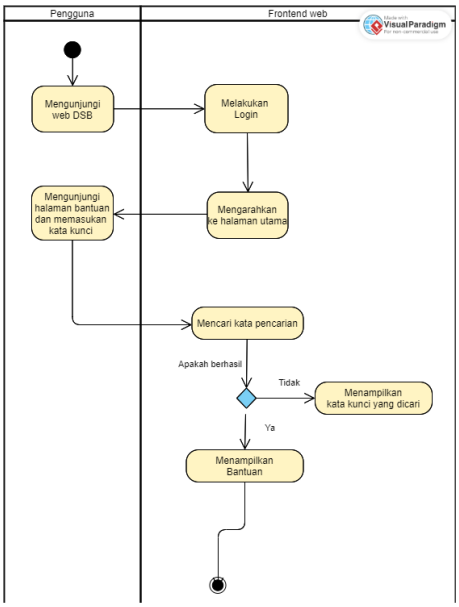
Setelah data penelitian dikumpulkan, hasil analisis ini akan menjelaskan lebih mendalam mengenai pemicu kerentanan dan respon sistem, penulis juga akan menyusun rumusan masalah untuk kemudian dilakukan perancangan sistem

3.1 Analisa Kebutuhan

Tahap ini merupakan hasil permodelan bagaimana sistem berjalan dan sistem usulan akan dirancang untuk kerentanan CSRF dan XSS. Gambar 2 menunjukkan bagaimana sistem berjalan merespon aktivitas transaksi yang dilakukan pengguna.

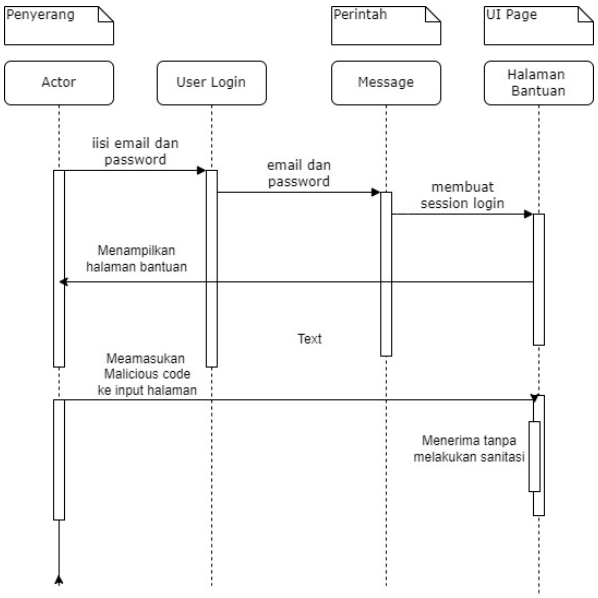


Gambar 2. Activity Diagram pengguna melakukan transaksi



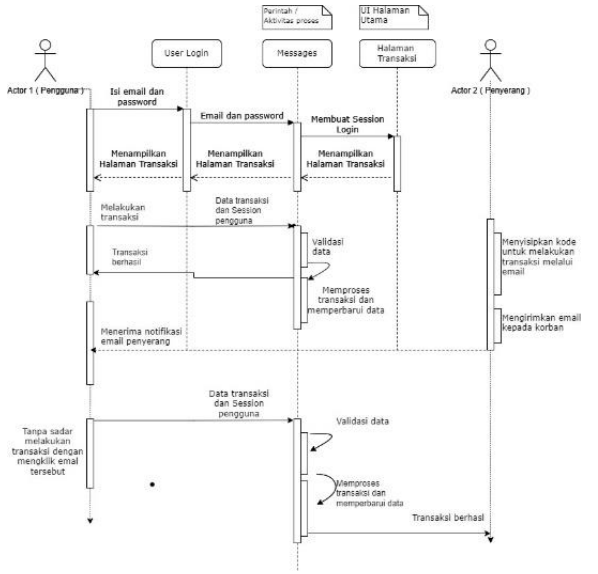
Gambar 3. Activity Diagram pengguna mengunjungi halaman bantuan

Gambar 3 memodelkan aktivitas antara sistem dan pengguna saat terjadi aktivitas pada halaman bantuan.



Gambar 4. Sequence Diagram saat penyerang menginjeksi malicious code

Pada Gambar 4 terdapat 2 titik kerentanan XSS tipe non persistent pada sistem. Pertama terdapat pada input box pencarian pada halaman bantuan, lalu selanjutnya pada parameter URL yang bisa mengeksekusi code javascript secara langsung di browser pengguna. Penyerang dapat memasukan script secara langsung pada input box web DSB untuk mengeksploitasi titik kerentanan pertama, lalu dengan melakukan social engineering penyerang dapat memanfaatkan untuk mencuri cookies pengguna terdaftar. Pada sistem usulan, sanitasi input akan dilakukan pada saat membaca parameter URL, maupun input pada box secara langsung.

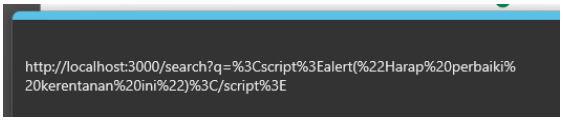


Gambar 5. Sequence Diagram pada sistem berjalan penyerang melakukan transaksi dengan meneruskan session pengguna terdaftar

Pada Gambar 5, kerentanan CSRF terbuka saat pengguna login kedalam web DSB, penyerang lalu mengirimkan email yang sudah dimodifikasi dengan code javascript untuk melakukan transaksi dan email template, untuk meyakinkan pengguna bahwa tautan pada email tersebut aman untuk dikunjungi, sehingga pengguna akan terpancing untuk mengunjungi tautan yang dikirimkan tersebut. Sistem sendiri tidak mempunyai kredensial unik untuk membedakan apakah request tersebut valid, maka transaksi pun dapat dilakukan.

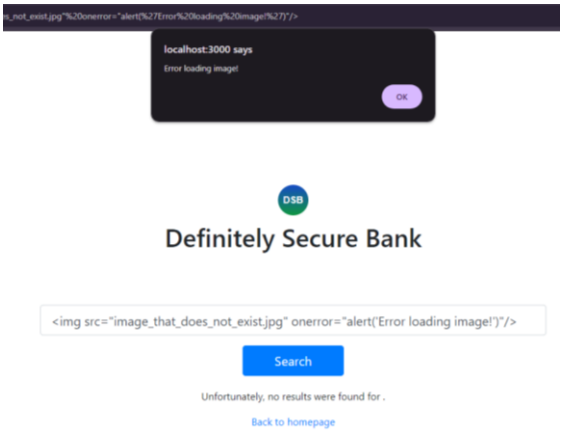
Berbeda dengan cara sistem memvalidasi request transaksi pada sistem berjalan, pada sistem usulan setiap kali pengguna terdaftar ingin melakukan transaksi, maka client akan melakukan request untuk meminta CSRF Token dari server. Nantinya token ini akan dikirimkan bersamaan dengan payload transaksi dan dijadikan satu satunya kredensial unik dalam membantu sistem mengenali transaksi yang valid. Jika penyerang mencoba meneruskan cookie pengguna dengan penyerangan sebelumnya, maka server akan menolak request tersebut karena ketidakhadiran CSRF Token [10] [11].

3.2 Blackbox testing



Gambar 6. copy URL pada notes

Pengujian XSS-non-persistent pada Gambar 6, terlihat jika Chrome browser telah melakukan encoding dengan menambah karakter acak pada url awal di notepad, namun untuk kerentanan kedua, dimana penyerang bisa membuat URL: *http://localhost:3000/search?q=%3Cimg%20src=%22image_that_does_not_exist.jpg%22%20onerror=%22alert(%27Error%20loading%20imag pe!%27)%22/%3*



Gambar 7. Eksekusi html element dengan javascript handle

Namun untuk eksekusi javascript terlihat masih dapat dieksekusi, terutama dalam bentuk format HTML tag

pada Gambar 7. Dengan demikian, Decision table akan disusun untuk perancangan sistem.

Tabel 1. Decission Table kerentanan XSS

Kondisi	Aturan		
	1	2	3
Script Tag	F	F	T
HTML Tag	F	F	T

Penjelasan Tabel 1: Aturan 1 (Sistem tidak akan merespon input pengguna jika kata kunci berisi string yang tidak valid seperti symbol dan lain lain; Aturan 2 (Sistem tidak akan menampilkan apapun jika kata kunci berisi string yang tidak valid seperti symbol dan lain lain); Aturan 3 (Sistem akan menampilkan respon dari kata kunci string yang valid).

3.3 Whitebox testing

Untuk kerentanan CSRF Pada Program validasi data (Frontend code) dan Program validasi data (Backend code) ditemukan bahwa sistem hanya melakukan validasi pada field input yang dikirimkan pada saat terjadi request. Sistem hanya akan menolak request jika ada field input yang hilang pada saat request terjadi [11].

Program validasi data (Frontend code)

```
const submittable =
!isNaN(floatAmount) && floatAmount > 0 && to.length > 0
&& description.length > 0
```

Program validasi data (Backend code)

```
if (
    isNaN(floatAmount) ||
    floatAmount <= 0 ||
    description == null ||
    description == '' ||
    to == null ||
    to == '' /* || (CSRFToken !== savedToken) */
  ) {
    return res.status(400).json({
      error: "Bad request: Invalid or missing data",
      success: false });
}
```

Melalui hasil tersebut kita bisa menyusun tabel Condition coverage seperti pada Tabel 2.

Tabel 2. Condition Coverage kerentanan CSRF

Gambar	Kondisi	Hasil
4,2,1,1,	*floatAmount = Not NULL*	TRUE
4,2,1,2	*floatAmount = 120* *to length* = Adi Kurniawan *description* = Transaksi	
4,2,1,1,	*floatAmount = NULL*	FALSE
4,2,1,2	*floatAmount = -1* *to = “”* *description = “”*	

3.4 Perancangan Sistem

Dari data dan pembahasan hasil penelitian bisa disimpulkan bahwa ada 2 kerentanan yang menyebabkan kerentanan *XSS* bertipe non persistent [12] dan *CSRF* pada aplikasi *DSB*, yaitu kurangnya sanitasi input dan identitas unik untuk memastikan request dibuat oleh pengguna. Dari kesimpulan tersebut penulis merancangkan sistem untuk membuat, menyimpan dan memvalidasi *CSRF* Token saat *request* transfer dibuat, juga melakukan sanitasi input pada simbol dan format input yang sering dijadikan *malicious injection code* pada kerentanan *XSS*.

Untuk menambahkan *CSRF* Token pada sistem, penulis menambahkan 4 fungsi baru pada sisi *Backend*, yaitu:

Fungsi *API* untuk mengirimkan *CSRF* token dalam format hexadecimal, merupakan hexadecimal (gabungan huruf dan angka) untuk divalidasi pada saat request transfer dibuat dari client.

Program Fungsi API CSRF Token (Backend code)

```
app.get('/generateCSRFToken', (req, res) => {
  const { session } = req.cookies;
  const user = db.getUser(session);

  if (!user) {
    res.json(null); // Use only res.json() here
  } else {
    const encryptedToken = encryptToken(savedToken,
secretKey);
    res.status(200).json({ csrfToken: encryptedToken }); //
Send status with JSON
  }
});
```

Fungsi dekripsi dan enkripsi *CSRF* token, fungsi ini digunakan untuk melindungi *CSRF* token yang dapat diakses secara langsung pada sisi client melalui *developer tools console*

Program Fungsi dekripsi dan enkripsi CSRF token

```
function encryptToken(token, secretKey) {
  const cipher = crypto.createCipheriv('aes-256-cbc',
secretKey, iv);
  let encrypted = cipher.update(token, 'utf8', 'hex');
  encrypted += cipher.final('hex');
  return encrypted;
}

function decryptToken(encryptedToken, secretKey) {
  const decipher = crypto.createDecipheriv('aes-256-cbc',
secretKey, iv);
  let decrypted = decipher.update(encryptedToken, 'hex',
'utf8');
  decrypted += decipher.final('utf8');
  return decrypted;
}
```

Fungsi *randomNumber* [13], sebagai fungsi *secretKey* untuk identitas unik setiap token *CSRF* pada saat melakukan enkripsi dan dekripsi.

Program Fungsi randomNumber

```
const secretKey = crypto.randomBytes(32);
const iv = crypto.randomBytes(16);

function generateRandomNumber(length) {
  return crypto.randomBytes(length)
    .toString('hex') // convert to hexadecimal format
    .slice(0, length); // return required number of characters
}
```

Fungsi *setInterval*, menggunakan *timer* dan memulai *interval* selama 20 detik untuk mengganti *secretKey* sebelumnya pada sisi *Backend* dan *Frontend*.

Program Fungsi setInterval

```
    let savedToken = generateRandomNumber(10);
    let refreshInterval;

    function startTokenRefresh() {
      refreshInterval = setInterval(() => {
        savedToken = generateRandomNumber(10);
      }, 20000); // Refresh token every 20 seconds
}
```

Untuk menambahkan sanitasi input pada sistem, penulis menambahkan 1 fungsi baru dan perubahan codebase pada sistem:

Fungsi Sanitasi input untuk menghilangkan berbagai macam tag unik di dalam penggunaan kode seperti *HTML*, *JS* ataupun *CSS* yang dicurigai dapat menimbulkan kerentanan *XSS*

Program Fungsi Sanitasi input

```
    const queryParams = new
URLSearchParams(location.search);
    const query = queryParams.get('q');
    function sanitizeSearchQuery(query) {
      return query?.replace(/[^a-zA-Z0-9\s\-_.!]+/g, ''); //
Remove all characters except allowed ones (more strict)
    }

    const sanitizedQuery = sanitizeSearchQuery(query);
    const onSubmit = e => {
      e.preventDefault();

history.push(`/search?q=${encodeURIComponent(sanitizeSear
chQuery(searchText))}`);
  };
```

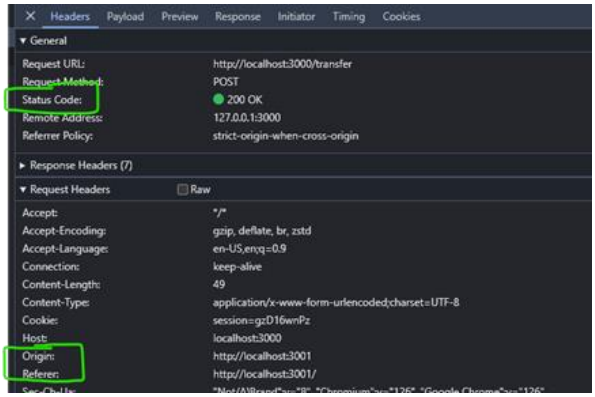
Pembaruan pada Client sesuai *best-practices* dari *OWASP* juga dilakukan pada saat merender kata pencarian, tidak lagi menggunakan *dangerouslySetInnerHTML*, melainkan *JSX*, cara yang direkomendasikan oleh *Framework Reactjs* yang mencegah *Client* secara langsung menetapkan input kedalam *HTML*

Program Pembaruan pada Client

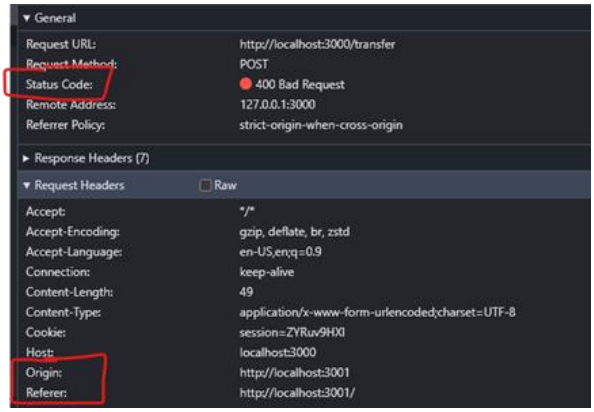
```
    {!!query && (
        <p>
          Unfortunately, no results were found for{' '}
          <span                    className="search-query-
name">{sanitizedQuery}</span>.
        </p>
      )}
      <p>
```

3.5 Pengujian Sistem

Dari hasil perancangan sistem dan pengembangan fitur keamanan pada Analisa kebutuhan, Sekarang penulis akan menguji kembali masing masing kerentanan yang ditemukan di awal, menggunakan metode *Blackbox testing*. Unuk kerentanan *CSRF* akan diuji apakah sistem masih bisa membuat *request* transfer jika dilakukan pada origin yang berbeda dengan *csrfToken* tanpa dekripsi dan dengan dekripsi.



Gambar 8 Request sebelum penerapan Enkripsi & Dekripsi CSRF Token



Gambar 9. Request sesudah penerapan CSRF Token

Pada Gambar 8 jika *CSRF* Token belum di enkripsi, maka *request* masih bisa dibuat, ditandai dengan kotak hijau pada origin yang berbeda. Hal ini dikarenakan *request* masih meneruskan cookie yang berisi informasi yang dibutuhkan sistem dalam memverifikasi identitas *client*, meskipun *CSRF* Token sudah di implementasikan. Namun jika *CSRF* Token sudah di implementasikan seperti pada Gambar 9 maka *request* tidak berhasil dibuat karena terdapat enkripsi dan dekripsi tambahan pada *Backend* yang sudah di implementasikan. Hal ini menunjukan jika fungsi dekripsi dan enkripsi telah berhasil menjamin kerahasiaan identitas *client* dengan fungsi *randomNumber* sebgai *secretKey*

4. Kesimpulan

Kesimpulan dari penelitian ini menunjukkan bahwa tujuan awal yang dijabarkan dalam bab “Pendahuluan” telah tercapai melalui hasil yang dipaparkan pada bab "Hasil dan Pembahasan." Fokus utama penelitian ini adalah menganalisis kerentanan *XSS* dan *CSRF* pada web *Definietly Secure Bank* [4], dengan hasil yang sesuai dengan ekspektasi awal. Analisis mendalam

menggunakan *Chrome Developer Tools* [14] berhasil mengidentifikasi celah keamanan pada *header request* dan pengaturan *cookies*, yang berperan dalam mengatasi kerentanan *CSRF* [14], serta menguji efektivitas teknik manual injection untuk mendeteksi *XSS*. Hasil penelitian ini menunjukkan bahwa kombinasi langkah-langkah keamanan seperti sanitasi input dan penggunaan token *CSRF* terbukti efektif dalam melindungi aplikasi web, sesuai dengan standar *OWASP* [10]. Namun, alternatif seperti *same-site cookies* masih membutuhkan kajian lebih lanjut karena adanya keterbatasan dalam kontrol *request* dan dukungan *browser* [15]. Adapun prospek pengembangan dari penelitian ini termasuk penerapan *framework modern* yang memiliki fungsi sanitasi input bawaan untuk meningkatkan keamanan aplikasi web terhadap serangan *XSS*. Studi lanjut dapat diarahkan pada eksplorasi penggunaan mekanisme keamanan tambahan, seperti implementasi *Content Security Policy (CSP)* dan peningkatan dukungan same-site cookies pada browser untuk mengatasi tantangan yang dihadapi pada skenario web frontend yang lebih kompleks [16]

Daftar Rujukan

[1] A. Alamsyah, "Pengantar javascript," [Online]. Available: https://scholar.google.de/citations?view_op=view_citation&hl=en&user=IZuCfzsAAAAJ&citation_for_view=IZuCfzsAAAAJ:u5HHmVD_uO8C.

[2] B. B. Gupta and P. Chaudhary, Cross-Site Scripting Attacks, Boca Raton, 2020.

[3] S. Azam, B. Shanmugam and K. Kannoorpatti, "Preventive Measures for Cross Site Request Forgery Attacks on Web-based Applications," 10 2018. [Online]. Available: https://www.researchgate.net/publication/328381749_Preventive_Measures_for_Cross_Site_Request_Forgery_Attacks_on_Web-based_Applications.

[4] V. Zhou, "Definietly Secure Bank," 2022. [Online]. Available: https://dsb.victorzhou.com/login.

[5] C. o. C. S. &. E. E. H. U. C. C. Independent Researcher, "Cross-Site Scripting Attacks and Defensive Techniques: A Comprehensive Survey*," Independent Researcher, College of Computer Science & Electronic Engineering, Hunan University, Changsha, China., 2022.

[6] S. Onofri and D. Onofri, "Utilization of CSRF Token," in Attacking and Exploiting Modern Web Applications: Discover the mindset, techniques, and tools to perform modern web attacks and exploitation, Packt Publishing, 2023, p. 253.

[7] b. M. Harwood and R. Price, Internet and Web Application Security 3rd Edition, Jones & Bartlett Learning; 3rd edition (December 12, 2022), 2022.

[8] D. Intern, "Dicoding," 21 Maret 2021. [Online]. Available: https://www.dicoding.com/blog/apa-itu-activity-diagram/.

[9] M. H. Y. &. N. N. Purnasari, "Jurnal Ilmu Siber dan Teknologi Digital," 2023. [Online]. Available: http://penerbitgoodwood.com/index.php/jisted/article/view/2298.

[10] Open Worldwide Application Security Project (OWASP), "Token Based Mitigation & Framework Security," in Cross Site Scripting & Cross Site Request Forgery Prevention, Open Worldwide Application Security Project (OWASP), 2024, p. Introduction.

[11] M. S. P. &. K. P. Singh, An Analytical Study on Cross-Site Scripting. 2020 International Conference on Computer Science, Engineering and Applications, India: IEE, 2020.

[12] S. Suroto dan A. Asman, "ANCAMAN TERHADAP KEAMANAN INFORMASI OLEH SERANGAN CROSS-SITE SCRIPTING (XSS) DAN METODE PENCEGAHANNYA," Zona Komputer: Program Studi Sistem Informasi Universitas Batam, Batam, 2021.

[13] S. J. Y. Weamie, "Cross-Site Scripting Attacks and Defensive Techniques: A Comprehensive Survey*," Independent Researcher, College of Computer Science & Electronic Engineering, Hunan University, Changsha, China., 2022.

[14] P. Yadav and C. D. Parekh, "A report on CSRF security challenges & prevention techniques," 2017. [Online]. Available: https://ieeexplore.ieee.org/document/8275852.

[15] J. Walke, "Encyclopedia, Wikipedia The free," 12 May 2024. [Online]. Available: https://en.wikipedia.org/wiki/React_(JavaScript_library).

[16] M. M. M. Lubis, Tommy, D. Handoko and N. Wulan, "Analisis Implementasi Laravel 9 Pada Website E-Book Dalam Mengatasi N+1 Problem Serta Penyerangan Csrf dan Xss," JIRSI, p. 32, 2023.