PETIR: Jurnal Pengkajian dan Penerapan Teknik Informatika
Vol. 17, No. 2, September 2024, P-ISSN 1978-9262, E-ISSN 2655-5018
DOI: https://doi.org/10.33322/petir.v17i2.2494

Mitigasi Pembangkitan Token Pada Kerentanan Cross Site Request Forgery dengan Pengujian Mutasi

Naufal Pandu Irsyadi[1]; Richard Reinhart[2)]; Abdurrasyid[2]; Gusti Ayu Putri Saptawati[1]*

1. Institut Teknologi Bandung, Jl. Ganesa No.10, Lb. Siliwangi, Coblong, Bandung, Jawa Barat 40132, Indonesia
2. Institut Teknologi PLN, Menara PLN, Jl. Lingkar Luar Barat, Duri Kosambi, Cengkareng, Jakarta Barat, DKI Jakarta 11750, Indonesia

[*)]Email: richard.rein16@gmail.com

Received: 02 Juni 2024 | Accepted:13 Desember 2024 | Published: 10 Januari 2025

ABSTRACT

In the digital era, software security is crucial to prevent financial losses and data theft due to cyber attacks such as Cross-Site Request Forgery (CSRF). Based on CVEdetails data, CSRF attacks increased significantly from 416 cases in 2020 to 1398 cases in 2023. This research explores the use of Static Application Security Testing (SAST) to detect and prevent CSRF attacks. In addition, anti-CSRF token and mutation testing are used to improve the quality of test cases in detecting CSRF vulnerabilities. The research results show that the mutation testing technique is effective in improving the quality of test cases, with the Mutation Score Index (MSI) value increasing from 50% to 100% after improvement. This research concludes that mutation testing can improve the quality of test cases, thereby providing better software protection against CSRF attacks.

Keywords: *Cross-Site Request Forgery, Static Application Security Testing, Mutation Testing*

ABSTRAK

Dalam era digital, keamanan perangkat lunak menjadi krusial untuk mencegah kerugian finansial dan pencurian data akibat serangan siber seperti Cross-Site Request Forgery (CSRF). Berdasarkan data CVEdetails, serangan CSRF meningkat signifikan dari 416 kasus pada tahun 2020 menjadi 1398 kasus pada tahun 2023. Penelitian ini mengeksplorasi penggunaan Static Application Security Testing (SAST) untuk mendeteksi serta mencegah serangan CSRF. Selain itu, anti-CSRF token dan uji mutasi digunakan untuk meningkatkan kualitas kasus uji dalam mendeteksi kerentanan CSRF. Hasil penelitian menunjukkan bahwa teknik uji mutasi efektif dalam meningkatkan kualitas test case, dengan nilai Mutation Score Index (MSI) meningkat dari 50% menjadi 100% setelah perbaikan. Penelitian ini menyimpulkan bahwa uji mutasi dapat meningkatkan kualitas kasus uji, sehingga memberikan perlindungan pada perangkat lunak yang lebih baik terhadap serangan CSRF.

Kata kunci: *Cross-Site Request Forgery, Static Application Security Testing, Uji Mutasi*

Vol. 17, No. 2, September 2024, P-ISSN 1978-9262, E-ISSN 2655-5018
DOI: https://doi.org/10.33322/petir.v17i2.2494

1. PENDAHULUAN

Keamanan perangkat lunak merupakan salah satu dari delapan bidang utama dalam keamanan siber yang bertujuan untuk melindungi properti keamanan informasi dan sistem selama pengembangan dan penggunaan perangkat lunak [1]. Pengujian keamanan perangkat lunak bertujuan untuk mendeteksi kelemahan dalam program yang dapat dieksploitasi atau kerentanan keamanan, serta mendokumentasikan temuan ini untuk diperbaiki [2]. Khususnya dalam konteks web, mengingat kompleksitas perangkat lunak situs web yang terus meningkat dan semakin canggihnya teknik serangan yang digunakan untuk mengeksploitasi kerentanan web, pengujian keamanan perangkat lunak menjadi semakin penting dan membutuhkan banyak biaya, dapat mencapai 50% dari total biaya pengembangan [3]. Hal ini menekankan pentingnya menciptakan perangkat lunak web yang lebih aman dan Tangguh [4]. Pelaksanaan pengujian keamanan perangkat lunak memerlukan test case, di mana kualitas kasus uji yang baik dapat mempengaruhi kualitas perangkat lunak secara keseluruhan [5].

Menurut komunitas terbuka Open Web Application Security Project (OWASP) yang didedikasikan untuk memungkinkan organisasi mengembangkan, membeli, dan memelihara aplikasi yang dapat dipercaya, serangan siber yang paling banyak terjadi adalah Cross-Site Request Forgery (CSRF) [6]. Berdasarkan data dari CVEdetails, sebuah website yang menyediakan data terkait software vulnerabilities, serangan CSRF terus meningkat setiap tahunnya, tahun 2020 terdapat 416 serangan dan pada tahun 2023 sebanyak 1398 serangan. CSRF adalah tipe serangan siber di mana penyerang mengeksekusi tindakan yang tidak diinginkan pada sebuah aplikasi web yang telah diautentikasi oleh pengguna tanpa sepengetahuan mereka. Serangan ini memanfaatkan kekurangan pada mekanisme otentikasi dan otorisasi, dengan mengirimkan permintaan palsu dari situs yang dipercayai oleh pengguna ke aplikasi target. Dampak kerentanan software terhadap serangan CSRF dapat sangat merugikan, mulai dari pencurian informasi pribadi dan keuangan pengguna hingga penyebaran malware, perubahan data sensitif, dan bahkan pengambilalihan akun secara keseluruhan. Serangan CSRF menunjukkan pentingnya menerapkan tindakan perlindungan seperti token otentikasi yang dapat mencegah eksekusi permintaan yang tidak sah.

Static Application Security Testing (SAST) merupakan metode untuk menjaga keamanan perangkat lunak. SAST memungkinkan pengembang untuk mendeteksi kerentanan dalam kode sumber sebelum perangkat lunak dijalankan [7]. Dengan menggunakan SAST, pengembang dapat menemukan potensi celah keamanan, seperti kerentanan CSRF. Ini memungkinkan perbaikan dilakukan lebih dini, sehingga mengurangi risiko eksposur terhadap serangan ketika perangkat lunak sudah digunakan oleh pengguna. Terdapat penelitian juga yang mengatakan bahwa pengujian perangkat lunak saat pengembangan lebih baik dibandingkan saat sudah berjalan [8].

Untuk memitigasi serangan CSRF adalah dengan menggunakan anti-CSRF token. Token ini di bangkitkan secara random dan harus diikutsertakan pada setiap form yang di submit user ke server dan kemudian dilakukan validasi. Keberadaan token ini mempersulit *malicious agent* untuk melakukan CSRF karena mereka harus menebak token yang ada di pada sisi server agar *request* nya dapat divalidasi. Terdapat 4 prinsip yang perlu dijaga dalam menerapkan sistem token ini [9]:

1. Menggunakan random number generator yang sulit diprediksi
2. Token memiliki expiry time yang singkat
3. Menggunakan metode yang lebih aman untuk validasi token (menggunakan hash)
4. Jangan mengirimkan token via request GET

Vol. 17, No. 2, September 2024, P-ISSN 1978-9262, E-ISSN 2655-5018
DOI: https://doi.org/10.33322/petir.v17i2.2494

Pengembangan kasus uji diperlukan untuk memastikan terjaganya 4 prinsip ini pada *source code*. Karenanya, diperlukan suatu metode untuk mengukur kinerja kasus uji dalam menjaga 4 prinsip ini. Salah satunya adalah dengan menggunakan pengujian mutasi.

Penelitian ini bertujuan untuk meningkatkan kualitas kasus uji, sehingga source code yang digunakan mampu meminimalkan risiko terjadinya serangan CSRF [10]. Meningkatkan kualitas kasus uji dengan menggunakan pengujian mutasi[11]. Pada penelitian ini, mutasi akan ditujukan pada fungsi dan algoritma untuk pembuatan dan validasi token, dalam pengujian mutasi dibutuhkan operator mutasi yang digunakan untuk membangkitkan mutan, adapun pada penelitian ini digunakan tiga operator mutasi yang diadaptasi dari penelitian yang telah dilakukan sebelumnya spesifik untuk kebutuhan kerentanan CSRF [12].

Penelitian sebelumnya [13] melakukan penelitian tentang efektivitas atribut SameSite dan kebijakan LaxByDefault dalam mengatasi serangan *Cross-Site Request Forgery* (CSRF). Atribut SameSite yang diperkenalkan untuk mencegah serangan CSRF, dievaluasi dalam hal adopsi dan efektivitas, terutama setelah penerapan kebijakan LaxByDefault oleh browser utama. Studi ini berfokus pada potensi mekanisme SameSite untuk melindungi website terhadap serangan CSRF, mengukur penerapan SameSite sebagai pertahanan, khususnya pada cookie pra-otentikasi dan pasca-otentikasi, serta menganalisis dampaknya terhadap CSRF, namun sayangnya meskipun mekanisme SameSite efektif terhadap skenario serangan CSRF yang besar, mekanisme ini bukanlah solusi yang komprehensif dan mungkin tidak sepenuhnya menghilangkan kerentanan CSRF. Oleh karena itu, disarankan untuk menggabungkan mekanisme SameSite dengan mekanisme pertahanan lainnya, seperti token anti-CSRF dan perlindungan sisi klien, untuk memberikan perlindungan komprehensif terhadap serangan CSRF [9].

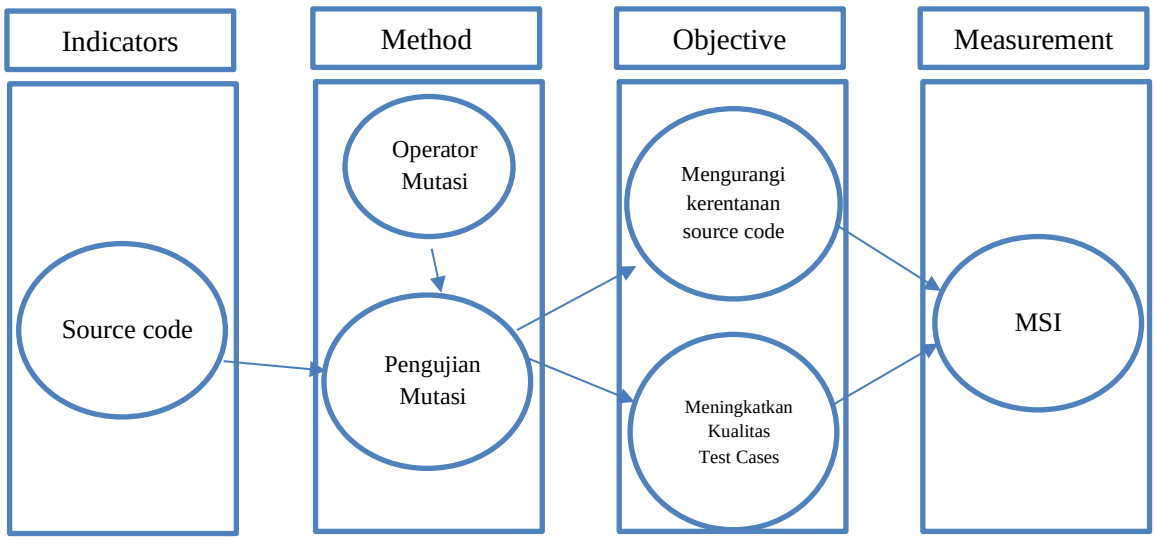
Penelitian lainnya [14] melakukan penelitian kerangka kerja berbasis deep learning yang dikembangkan untuk mendeteksi kerentanan CSRF dalam aplikasi web. Kerangka kerja ini bertujuan untuk mengidentifikasi permintaan HTTP yang sensitif terhadap keamanan untuk mencegah serangan CSRF, yang mengeksploitasi kerentanan dalam aplikasi web untuk melakukan tindakan tidak sah atas nama pengguna yang diautentikasi. Studi ini menekankan pentingnya mengkategorikan permintaan yang sensitif terhadap keamanan secara akurat dan potensi implikasinya terhadap kemajuan strategi keamanan web. Namun, karena menggunakan model deep learning, hasil yang diberikan bergantung pada dataset latih. Jika dilihat dari metric evaluasinya, masih terdapat sekitar 15% sampai 20% error yang dilakukan model tersebut. Sedangkan jika menggunakan uji mutasi, kasus uji dapat terus diperbaiki sampai memperoleh *Mutation Score Indicator* (MSI) yang diharapkan atau mencapai MSI 100%. MSI adalah instrumen pengujian yang digunakan pada pengujian mutasi untuk mengukur hasil dari pengujian mutasi yang dilakukan dengan rentang nilai 0-1.

Tidak sedikit penelitian-penelitian yang dilakukan terkait pengujian mutasi namun penelitian yang menggunakan pendekatan pengujian mutasi dalam domain software security masih terbatas, seperti yang dilakukan oleh Huang yang mengusulkan 13 operator mutasi untuk *SQL Injection* [15], mengulang apa yang sudah dilakukan Appelt pada tahun 2014 [16], sedangkan Loise mengusulkan 15 operator mutasi untuk jenis kerentanan *SQL Injection* dan XSS pada aplikasi berbasis java, Siavashi melakukan penilaian kerentanan user and session authentication pada web service yang merupakan no tujuh pada Top 10 OWASP dengan mengenalkan 8 operator mutasi [17]. Dari penelitian yang telah dilakukan sebelumnya dapat disimpulkan bahwa belum ada penelitian yang mencoba menerapkan pengujian mutasi pada perangkat lunak untuk memastikan kualitas kasus uji dapat melihat kerentanan CSRF dengan menggunakan operator mutasi yang tersedia, ini yang menjadi kontribusi dari penelitian ini.

PETIR: Jurnal Pengkajian dan Penerapan Teknik Informatika
Vol. 17, No. 2, September 2024, P-ISSN 1978-9262, E-ISSN 2655-5018
DOI: https://doi.org/10.33322/petir.v17i2.2494

2. METODE/PERANCANGAN PENELITIAN

2.1. Kerangka Pemikiran



Gambar 1. Kerangka Pemikiran

Gambar 1 di atas menunjukkan kerangka pemikiran dari penelitian yang dilakukan, diawali dengan indikator yang dianggap sebagai sumber input dari pendekatan yang digunakan, *source code* menjadi bahan yang akan dimutasi dengan pendetakan pengujian mutasi mengubah dari source code asli menjadi source code lainnya yang disebut mutant mengacu pada operator mutasi yang digunakan. Tujuannya adalah untuk mengurangi kerentanan source code dengan mengevaluasi kasus uji yang digunakan, karna tidak menutup kemungkinan kasus uji yang kurang tepat tidak dapat mendeteksi adanya kerentanan pada fungsi keamanan yang dibuat, hasil evaluasi akan membantu software tester untuk meningkatkan kualitas dari kasus uji. Instrumen *Mutation Score Indicator* digunakan untuk menguji dari pendekatan pengujian mutasi yang dilakukan.

2.2. Pengujian Mutasi

Metode yang akan dilakukan pada penelitian ini adalah menggunakan mutation testing atau uji mutasi. Mutation testing adalah metode dalam software testing untuk mengevaluasi kasus uji yang digunakan pada pembuatan perangkat lunak dengan mengubah sumber code berdasarkan operator mutasi yang digunakan[12]. Terdapat 5 tahap dalam mutation testing, yaitu menyiapkan baseline source dan kasus uji, mutant generation, running kasus uji, analisis hasil, dan tingkatkan kualitas kasus uji.

1. Menyiapkan *baseline source* dan kasus uji
   Menyiapkan *source code* yang akan diuji menggunakan fungsi-fungsi terbaik untuk memitigasi serangan CSRF berdasarkan [18] dan juga kasus uji untuk menguji *source code* yang telah dimutasi.
2. Membangkitkan mutant
   Melakukan mutasi pada fungsi-fungsi dalam source code yang digunakan untuk pembuatan dan validasi token
3. Menjalankan kasus uji
   Menjalankan kasus uji pada source code yang telah dimutasi

Vol. 17, No. 2, September 2024, P-ISSN 1978-9262, E-ISSN 2655-5018
DOI: https://doi.org/10.33322/petir.v17i2.2494

4. Analisis hasil uji mutasi
   Selanjutnya, mengidentifikasi mutan apa saja yang berhasil dideteksi oleh kasus uji (mutation killed) dan gagal dideteksi (mutation escaped). Kinerja kasus uji diukur menggunakan Mutation Score Index (MSI), perbandingan mutation killed dan total semua mutasi. Semakin tinggi nilai MSI maka semakin baik kualitas kasus uji.
5. Tingkatkan kualitas kasus uji
   Perbaiki kasus uji secara iteratif sampai semua mutan berhasil dideteksi oleh kasus uji atau mencapai nilai MSI tertentu.
   Mutation testing dapat diterapkan untuk mengembangkan kasus uji untuk menjaga agar fungsionalitas sistem anti-CSRF token berjalan sesuai prinsip-prinsip yang ada. Fungsi-fungsi pada baseline akan dimutasi menggunakan fungsi yang serupa, tapi lebih rentan terhadap serangan CSRF. Tabel 1. merupakan operator mutasi yang akan diuji pada penelitian ini.

Tabel 1. Operator Mutasi CSRF

No	Kategori	Fungsi	Variasi Operator Mutasi
1	Pembuatan Token	Fungsi Random Bytes	`Openssl_random_pseudo_bytes, random_int, mcrypt_create_iv, mt_rand
2	Pembuatan Token	Fungsi HMAC	md5, sha1, sha256, Whirlpool, RIPEMD-160
3	Validasi Token	Fungsi Validasi Token	Strcmp, strcasecmp, strcoll, levenshtein, Hash_equal(), “===”, “==”

Tabel 1 di atas menunjukkan operator mutasi yang diusulkan pada penelitian ini, terdapat 3 fungsi yang diusulkan, yaitu operator mutasi untuk fungsi random bytes, fungsi pembuatan Hash-based Message Authentication Code (HMAC) dan fungsi validasi token.

2.3. Skor Mutasi

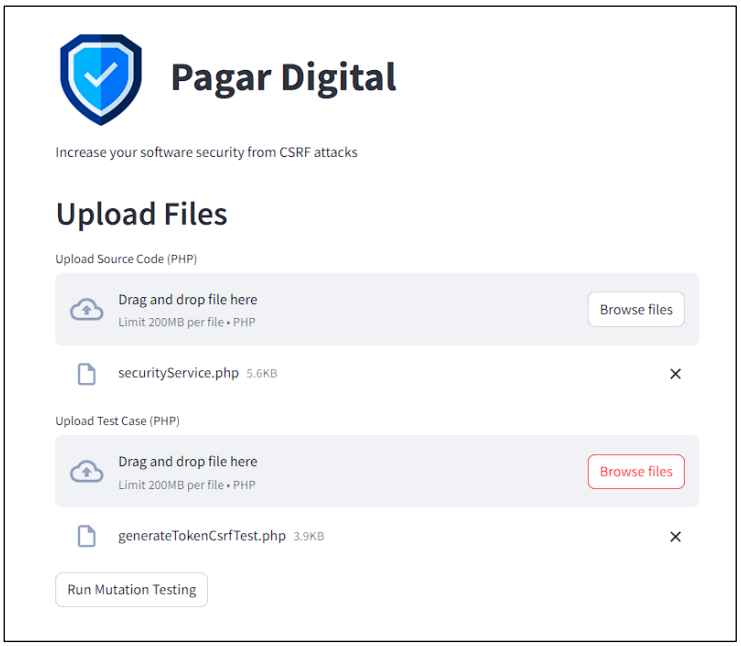
$$MSI = \frac{Number\ of\ killed\ mutants}{Total\ number\ of\ mutants} \times 100\% \quad (1)$$

Mutation score indikator digunakan untuk mengukur keberhasilan dari pengujian mutasi yang dilakukan, number of killed mutatants didapat dari jumlah mutant (source code yang diubah) dapat dideteksi oleh kasus uji yang dibuat, dibandingkan dengan jumlah mutant yang dihasilkan secara keseluruhan.

3. HASIL DAN PEMBAHASAN

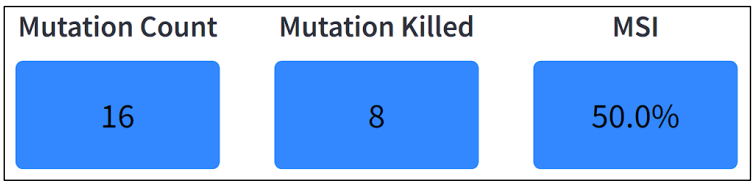
Hasil penelitian ini ditampilkan pada sebuah aplikasi web untuk melakukan uji mutasi pada *source code* menggunakan kasus uji yang sudah dibuat. Hasil pengujian mutasi akan ditampilkan dalam sebuah dashboard sederhana. Gambar 2 merupakan tampilan dari aplikasi web.

Vol. 17, No. 2, September 2024, P-ISSN 1978-9262, E-ISSN 2655-5018
DOI: https://doi.org/10.33322/petir.v17i2.2494



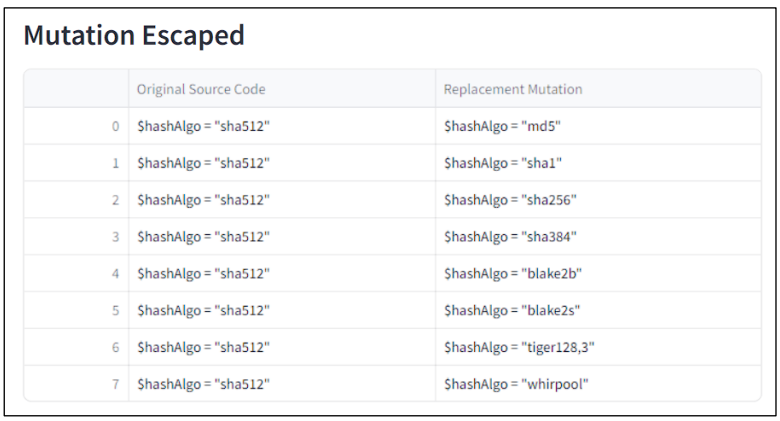
Gambar 2. Tampilan Aplikasi Web

Pada tahap pertama, kasus uji yang dibuat belum terlalu bagus untuk mendeteksi fungsi-fungsi dalam source code yang telah dimutasi. MSI yang dihasilkan pada kasus uji pertama sebesar 50%, dengan 8 mutation killed dan 8 mutation escaped seperti pada Gambar 3.



Gambar 3. Hasil Mutation Testing 1

Hal ini dapat terjadi karena kasus uji pertama tidak menguji fungsi HMAC. Dapat dilihat pada Gambar 4, semua *mutation escaped* atau mutasi yang tidak dapat dideteksi dengan kasus uji yang ada terdapat pada mutasi yang dilakukan pada fungsi HMAC.



	Original Source Code	Replacement Mutation
0	$hashAlgo = "sha512"	$hashAlgo = "md5"
1	$hashAlgo = "sha512"	$hashAlgo = "sha1"
2	$hashAlgo = "sha512"	$hashAlgo = "sha256"
3	$hashAlgo = "sha512"	$hashAlgo = "sha384"
4	$hashAlgo = "sha512"	$hashAlgo = "blake2b"
5	$hashAlgo = "sha512"	$hashAlgo = "blake2s"
6	$hashAlgo = "sha512"	$hashAlgo = "tiger128,3"
7	$hashAlgo = "sha512"	$hashAlgo = "whirpool"

Gambar 4. List Mutation Escaped pada Mutation Testing 1

Vol. 17, No. 2, September 2024, P-ISSN 1978-9262, E-ISSN 2655-5018
DOI: https://doi.org/10.33322/petir.v17i2.2494

Gambar 5 merupakan contoh *source code* awal menggunakan algoritma SHA512 dan Gambar 6 contoh *source code* yang telah dimutasi menggunakan SHA256. Selanjutnya kasus uji perlu diperbaiki agar mampu mendeteksi mutasi pada fungsi HMAC.

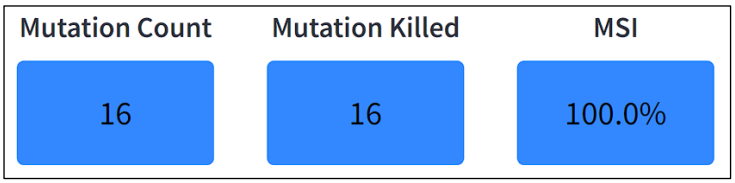
```
private $hashAlgo = "sha512";
private function hMacWithIp($token)//hash_message_authentication_code
{//based on
    $_COOKIE["PHPSESSID"] = 'dummy_cookie_phpsessid';
    $message = $_COOKIE["PHPSESSID"]. "!" .$token;
    $hashHmac = \hash_hmac($this->hashAlgo, $message , $this->hmacData);
    return $hashHmac;
}
```

Gambar 5. Source Code Sebelum Mutasi

```
private $hashAlgo = "sha256";
private function hMacWithIp($token)//hash_message_authentication_code
{//based on
    $_COOKIE["PHPSESSID"] = 'dummy_cookie_phpsessid';
    $message = $_COOKIE["PHPSESSID"]. "!" .$token;
    $hashHmac = \hash_hmac($this->hashAlgo, $message , $this->hmacData);
    return $hashHmac;
}
```

Gambar 6. Source Code Setelah Mutasi

Kasus uji telah diperbaiki dengan menambahkan pengujian untuk fungsi HMAC. MSI yang dihasilkan menjadi 100%, total 16 mutasi berhasil ter-killed.



Gambar 7. Hasil Mutation Testing 2

4. KESIMPULAN DAN SARAN

Penelitian ini telah berhasil menunjukkan bahwa teknik uji mutasi efektif dalam membantu mengevaluasi kasus uji, sehingga dapat membantu seorang *software tester* untuk meningkatkan kualitas kasus uji untuk meminimalkan risiko terjadinya serangan Cross-Site Request Forgery (CSRF). Hasil penelitian menunjukkan bahwa penggunaan uji mutasi mampu mendeteksi kelemahan dan celah keamanan. Dengan teknik ini, jumlah mutan yang terdeteksi meningkat, yang berarti kasus uji yang dihasilkan lebih komprehensif dalam menemukan potensi kerentanan keamanan.

Teknik uji mutasi juga membantu memvalidasi dan memperbaiki kasus uji yang ada, memastikan bahwa kasus uji tersebut tidak hanya memenuhi persyaratan fungsional tetapi juga mampu menangani skenario serangan keamanan. Selain itu, uji mutasi memberikan cara yang lebih sistematis dan terukur dalam mengevaluasi keamanan aplikasi web terhadap serangan CSRF. Dengan demikian, penelitian ini membuktikan bahwa teknik uji mutasi meningkatkan efisiensi dan efektivitas pengujian keamanan, memberikan perlindungan lebih baik terhadap serangan CSRF.

Vol. 17, No. 2, September 2024, P-ISSN 1978-9262, E-ISSN 2655-5018
DOI: https://doi.org/10.33322/petir.v17i2.2494

Ke depan, penelitian yang dapat diusulkan adalah menguji teknik uji mutasi pada berbagai lingkungan pengembangan dan framework web lainnya untuk mengevaluasi keefektifannya dalam berbagai konteks teknologi. Selain itu, pengembangan alat otomatisasi yang lebih canggih dan terintegrasi dengan pipeline CI/CD untuk penerapan teknik uji mutasi secara otomatis dapat meningkatkan efisiensi pengujian keamanan. Penelitian ini juga mengusulkan penggunaan pembelajaran mesin untuk mengoptimalkan proses pembuatan mutan dan melakukan studi kasus di industri untuk mengumpulkan data empiris yang lebih banyak.

UCAPAN TERIMAKASIH

Kami penulis mengucapkan terima kasih kepada Institut Teknologi Bandung yang telah memberi dukungan yang membantu pelaksanaan penelitian dan atau penulisan artikel.

DAFTAR PUSTAKA

[1] J. T. Quiroz, M. A. A. Oscategui, and J. Armas-Aguirre, "Cybersecurity Taxonomy: research and knowledge areas," in *2021 IEEE 1st International Conference on Advanced Learning Technologies on Education & Research (ICALTER)*, IEEE, 2021, pp. 1–4.

[2] N. F. Awang and A. A. Manaf, "Automated security testing framework for detecting SQL injection vulnerability in web application," in *Global Security, Safety and Sustainability: Tomorrow's Challenges of Cyber Security: 10th International Conference, ICGS3 2015, London, UK, September 15-17, 2015. Proceedings 10*, Springer, 2015, pp. 160–171.

[3] P. Liu, Z. Xu, and J. Ai, "An approach to automatic test case generation for unit testing," in *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, IEEE, 2018, pp. 545–552.

[4] H. ur Rehman, M. Nazir, and K. Mustafa, "Security of Web Application: State of the Art: Research Theories and Industrial Practices," in *Information, Communication and Computing Technology: Second International Conference, ICICCT 2017, New Delhi, India, May 13, 2017, Revised Selected Papers 2*, Springer, 2017, pp. 168–180.

[5] H. Pei, B. Yin, M. Xie, and K.-Y. Cai, "Dynamic random testing with test case clustering and distance-based parameter adjustment," *Inf. Softw. Technol.*, vol. 131, p. 106470, 2021.

[6] S. Elder, N. Zahan, V. Kozarev, R. Shu, T. Menzies, and L. Williams, "Structuring a comprehensive software security course around the OWASP application security verification standard," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, IEEE, 2021, pp. 95–104.

[7] J. Li, "Vulnerabilities mapping based on OWASP-SANS: a survey for static application security testing (SAST)," *arXiv Prepr. arXiv2004.03216*, 2020.

[8] A. W. Marashdih, Z. F. Zaaba, K. Suwais, and N. A. Mohd, "Web application security: An investigation on static analysis with other algorithms to detect cross site scripting," *Procedia Comput. Sci.*, vol. 161, pp. 1173–1181, 2019.

[9] W. H. Rankothge and S. M. N. Randeniya, "Identification and mitigation tool for cross-site request forgery (CSRF)," in *2020 IEEE 8th R10 Humanitarian Technology Conference (R10-HTC)*, IEEE, 2020, pp. 1–5.

[10] Q. Zhu, A. Zaidman, and A. Panichella, "How to kill them all: an exploratory study on the impact of code observability on mutation testing," *J. Syst. Softw.*, vol. 173, p. 110864, 2021.

[11] R. A. Silva, S. do R. S. de Souza, and P. S. L. de Souza, "A systematic review on search based mutation testing," *Inf. Softw. Technol.*, vol. 81, pp. 19–35, 2017.

[12] B. Sitohang, Y. D. W. Asnar, and G. A. P. Saptawati, "Securing Cross-Site Request Forgery

Vol. 17, No. 2, September 2024, P-ISSN 1978-9262, E-ISSN 2655-5018
DOI: https://doi.org/10.33322/petir.v17i2.2494

Vulnerabilities in Web Applications Using Mutation Analysis," in *2024 2nd International Conference on Software Engineering and Information Technology (ICoSEIT)*, IEEE, 2024, pp. 227–232.

[13] L. Compagna, H. Jonker, J. Krochewski, B. Krumnow, and M. Sahin, "A preliminary study on the adoption and effectiveness of SameSite cookies as a CSRF defence," in *2021 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, IEEE, 2021, pp. 49–59.

[14] G. Beltrano, C. Greco, M. Ianni, and G. Fortino, "Deep Learning-Based Detection of CSRF Vulnerabilities in Web Applications," in *2023 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, IEEE, 2023, pp. 916–920.

[15] Y. Huang *et al.*, "A mutation approach of detecting SQL injection vulnerabilities," in *Cloud Computing and Security: Third International Conference, ICCCS 2017, Nanjing, China, June 16-18, 2017, Revised Selected Papers, Part II 3*, Springer, 2017, pp. 175–188.

[16] D. Appelt, C. D. Nguyen, L. C. Briand, and N. Alshahwan, "Automated testing for SQL injection vulnerabilities: an input mutation approach," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, 2014, pp. 259–269.

[17] F. Siavashi, D. Truscan, and J. Vain, "Vulnerability assessment of web services with model-based mutation testing," in *2018 IEEE international conference on software quality, reliability and security (QRS)*, IEEE, 2018, pp. 301–312.

[18] D. Upadhyay, N. Gaikwad, M. Zaman, and S. Sampalli, "Investigating the avalanche effect of various cryptographically secure Hash functions and Hash-based applications," *IEEE Access*, vol. 10, pp. 112472–112486, 2022.