Kombinasi *Timestamp Nonce* dan Nilai *Salt* pada Autentikasi *Single Sign-On*

Rizky Fenaldo Maulana(1), Ardian Yusuf Wicaksono(2), Kharisma Monika Dian Pertiwi(3), Muhammad Dzulfikar Fauzi(4)

Institut Teknologi Telkom Surabaya
Jalan Ketintang No 156, Surabaya, Jawa Timur, Indonesia

Email: [1]aldo@ittelkom-sby.ac.id, [2]ardian@ittelkom-sby.ac.id, [3]kharismamonika@ittelkom-sby.ac.id

Tersedia Online di
http://www.jurnal.unublitar.ac.id/index.php/briliant

Sejarah Artikel
Diterima pada 8 Mei 2023
Disetuji pada 6 September 2023
Dipublikasikan pada 30 November 2023
Hal. 1049-1061

Kata Kunci:
Single Sign-On; Timestamp Nonce; Salt; Autentikasi; Man In The Middle

DOI:
http://dx.doi.org/10.28926/briliant.v8i4.1389

Abstrak: *Single Sign-On* atau SSO merupakan metode autentikasi tunggal dimana pengguna cukup melakukan autentikasi sekali untuk mengakses beberapa sistem atau aplikasi yang terdaftar. Proses autentikasi yang digunakan pada SSO menggunakan metode autentikasi klasik. Penyerang dapat mencuri data pengguna pada proses komunikasi autentikasi antara pengguna dan server, serangan ini disebut dengan *Man in The Middle Attack* (MiTM). *Nonce* adalah tindakan untuk melindungi informasi pribadi dan rahasia serta nilai unik yang tidak akan diulang. Pada makalah ini mengusulkan metode autentikasi pada SSO dengan kombinasi algoritma *timestamp nonce* dan nilai *salt*. Kombinasi dari kedua algoritma tersebut akan menghasilkan sebuah nilai acak yang bersifat unik. Nilai unik yang dihasilkan tersebut akan digunakan untuk mengenkripsi nama pengguna dan kata sandi pada tahap autentikasi. Sehingga diharapkan dengan metode tersebut dapat mengamankan data pengguna dari pencurian data dan serangan MiTM. Penerapan kombinasi algoritma *Timestamp Nonce* dan nilai *salt* dapat dijadikan salah satu alternatif pengamanan autentikasi pada sistem SSO. Selain itu, peneliti juga melakukan uji coba serangan MiTM dengan melakukan pembacaan setiap paket data yang bertujuan mendapatkan nama pengguna dan kata sandi. Hasilnya adalah variabel nama pengguna dan kata sandi bersifat unik dan tidak bisa di dekripsi.

PENDAHULUAN

Penggunaan teknologi informasi kini telah meluas di berbagai sektor. Salah satu keuntungannya adalah memudahkan tugas manusia (Wahyudiyono, 2016). Salah satu aspek penting yang perlu diperhatikan dalam penggunaan teknologi informasi dan komunikasi adalah keamanan. Salah satu cara menjaga keamanan suatu aplikasi yaitu dengan autentikasi (Wanani, Aouda,2017).

Proses autentikasi adalah cara bagi sistem untuk menentukan apakah pengguna berhak untuk mengakses suatu sistem atau tidak (Aminudin, 2014). Umumnya, proses autentikasi dilakukan dengan menggunakan skema login, dimana pengguna memasukkan nama pengguna dan kata sandi ke dalam sistem. Sistem kemudian akan memeriksa dan memvalidasi apakah data yang dimasukkan

oleh pengguna tersebut benar atau tidak. Jika valid, pengguna akan diizinkan untuk mengakses sistem, tetapi jika tidak valid, pengguna akan ditolak untuk mengakses.

Sebuah perusahaan atau organisasi besar dapat memiliki sistem informasi lebih dari satu sesuai kebutuhan proses bisnis maupun manajemen informasi yang digunakan (PaulaBajdor,2015). Setiap sistem informasi memiliki proses autentikasi yang berbeda-beda. Hal ini menyebabkan pengguna kesulitan karena harus mengingat setiap nama pengguna dan kata sandi yang mereka gunakan. Oleh karena itu perusahaan atau organisasi memerlukan sebuah sistem autentikasi tunggal. Dengan sistem autentikasi tunggal, pengguna hanya perlu melakukan satu kali autentikasi dan dapat mengakses aplikasi pada perusahaan tersebut sesuai dengan hak aksesnya.

SSO (*Single Sign-On*) adalah teknologi autentikasi tunggal yang memungkinkan pengguna untuk mengakses beberapa sistem informasi dengan hanya menggunakan satu akun. Dengan menggunakan SSO, pengguna hanya perlu melakukan autentikasi sekali saja. Teknologi SSO sudah banyak digunakan untuk mempermudah akses pengguna ke berbagai layanan (Beltran, 2016).

Pada era digital saat ini, menjaga keamanan data dan informasi pribadi sangatlah krusial. Tidak ada orang yang ingin data atau informasi pribadi yang dikirim melalui internet dapat diakses oleh pihak yang tidak bertanggung jawab. Penggunaan jaringan internet sangat rawan terjadi pencurian atau penyadapan informasi. Hal tersebut dapat terjadi saat proses pengiriman data dari *client* ke server (Liu Jia, 2020).

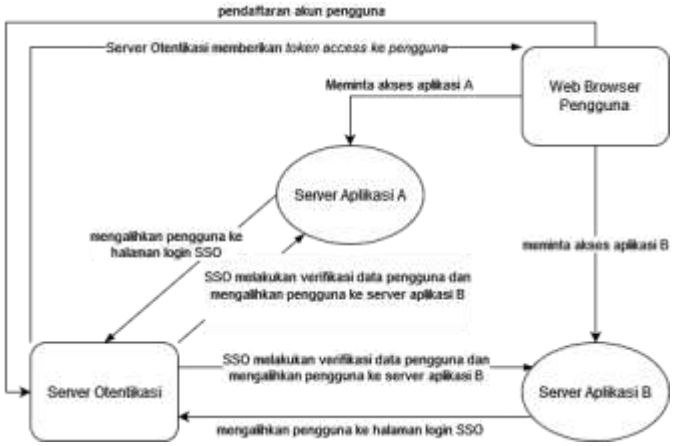
Salah satu tantangan dalam implementasi SSO pada suatu lingkungan sistem informasi adalah keandalan terhadap serangan pencurian data. Salah satu jenis pencurian data yang dapat terjadi pada skema SSO adalah serangan *Man in- The Middle* (MiTM). Serangan ini berfokus pada pencurian data autentikasi maupun data *access token* yang didapatkan setelah proses autentikasi dilakukan (Nickson M. Karie,dkk, 2020). Serangan dapat berdampak langsung atau tidak langsung pada sistem informasi karena penyerang dapat mengaksesnya dan melakukan pencurian atau perusakan data melalui serangan MiTM (*Man in The Middle*).

Yang Tie-jun (2014) telah melakukan penelitian dengan mengusulkan sistem SSO yang dapat digunakan untuk autentikasi pada aplikasi berbagai platform. Konsep yang diajukan adalah ketika pengguna *login* ke sistem SSO, informasi pengguna akan disebarluaskan ke seluruh aplikasi dalam sistem menggunakan teknologi *Asynchronous Javascript and XML* (AJAX).

Aminudin (2014) telah mengimplementasikan SSO menggunakan protokol Oauth. OAuth 2.0 merupakan sebuah protokol standar industri yang digunakan untuk proses otorisasi. Fokus utama dari OAuth 2.0 adalah untuk mempermudah klien dalam memberikan alur otorisasi yang spesifik untuk aplikasi yang berbasis *website, desktop, mobile,* dan perangkat *Internet of Things* (OAuth 2.0, *Accessed*: Jun. 21, 2021). OAuth 2.0 bertugas sebagai penghubung antara *Resource Owner* (pengguna) dan *Client* (Aplikasi Pihak Ketiga). *Client* akan menerima token akses dari OAuth jika proses autentikasi dan otorisasi berhasil dilakukan (T. Febrian, Accessed Jun. 22, 2021). Sebuah penelitian yang dilakukan oleh Z. Triartono pada tahun 2019 mengimplementasikan OAuth 2.0 untuk proses autentikasi berbasis *role* dengan menggunakan kerangka kerja *Laravel*. Penelitian tersebut menunjukkan bahwa model sistem autentikasi yang diusulkan mampu bekerja dengan baik dan

dapat menangani permintaan dalam jumlah tertentu meskipun dengan sumber daya yang terbatas.

Single Sign-On atau SSO merupakan suatu metode yang memungkinkan pengguna memiliki kontrol akses dengan beberapa sistem dengan satu kali autentikasi (M. Kihara and S. Iriyama, 2020). SSO dapat diterapkan pada berbagai platform seperti aplikasi *desktop*, aplikasi berbasis *website* serta aplikasi *mobile*. Pada penerapan SSO berbasis *website* terdapat 3 komponen utama penyusun, yaitu *web browser* yang digunakan oleh pengguna, server autentikasi dan server layanan aplikasi (L. S. Ramamoorthi and D. Sarkar, 2020).



Gambar 1. Cara Kerja SSO.

Cara kerja SSO dengan melakukan verifikasi data pengguna pada setiap aksi yang dilakukan oleh pengguna (V. Radha and D. H. Reddy, 2018). Pada Gambar 1 Pengguna akan mendaftar pada server autentikasi. Setelah itu apabila pengguna ingin mengakses aplikasi A, maka server aplikasi A akan mengarahkan pengguna ke halaman *login* yang akan dilayani oleh server autentikasi. Selanjutnya, server autentikasi akan melakukan verifikasi data pengguna dan akan mengarahkan pengguna ke halaman aplikasi A serta memberikan *access token* kepada pengguna yang akan disimpan pada *cookie web browser* pengguna. Selanjutnya apabila pengguna menginginkan untuk mengakses aplikasi B, maka aplikasi B akan mengalihkan pengguna ke halaman *login* server autentikasi, namun karena pengguna telah memiliki *access token*, maka server autentikasi akan melakukan verifikasi *access token* dan mengarahkan pengguna kembali ke halaman aplikasi B.

Nonce adalah tindakan untuk melindungi data pribadi dan sensitif dengan menggunakan nilai yang unik dan tidak akan berulang. *Nonce* berfungsi untuk mencegah serangan *replay* dan perlindungan dari serangan *Cross-Site Request Forgery* (CSRF*)*. Selain itu, dalam kriptografi, *nonce* dapat digunakan sebagai kunci dalam proses enkripsi dan dekripsi. Terdapat beberapa jenis *nonce*, seperti *random nonce, sequential nonce*, dan *timestamp nonce.* (G. M. Køien, 2015). *Random Nonce,* yakni angka yang dihasilkan secara acak dari program, yang kemudian dapat digunakan dalam proses autentikasi dan otorisasi. (OWASP- Cheat Sheet Series,2021). Dalam autentikasi dan otorisasi, *sequential nonce* merupakan penggunaan urutan angka yang biasanya dihasilkan secara increment. Sedangkan *timestamp nonce* adalah angka yang dihasilkan dari *timestamp*, digunakan karena

selalu memiliki nilai yang unik dan berguna dalam memberikan batasan waktu untuk autentikasi dan otorisasi.

Salt merupakan bagian utama dari metode enkripsi, dimana nilai *salt* dapat berupa sebuah kata acak maupun statis. Nilai *salt* digunakan untuk menambahkan parameter pada fungsi enkripsi untuk membuat nilai awal menjadi nilai unik yang tidak bisa dikembalikan lagi ke nilai semula (A. Naiakshina,dkk, 2017). Pada skema SSO yang diusulkan, penulis menggunakan 2 algoritma *Hashing*, yaitu MD5 dan SHA 256. Pemilihan kedua algoritma tersebut didasari karena kedua algoritma tersebut bersifat *irreversible*. Namun, untung meningkatkan keandalan kedua algoritma tersebut, penulis menambahkan nilai *salt* pada proses enkripsi. Hal ini didasari pada penelitian Mohammed Ali dkk (2020) yang menambahkan nilai suatu *key* pada proses algoritma md5 pada pengamanan dokumen elektronik dan penelitian Alsemi(2022) yang meningkatkan keandalan SHA256 dengan menambahkan nilai *salt* pada proses enkripsi. Penambahan nilai *salt* ini membuat penyerang yang akan kesulitan untuk meretas sistem dengan memanfaatkan data pengguna yang dicuri karena data tersebut sudah terenkripsi dengan nilai *salt* sehingga data bersifat unik dan sekali pakai (F. Ayankoya and B. Ohwo,2019).

Dalam paper ini diusulkan suatu teknik autentikasi pengguna pada sistem SSO yang menggunakan kombinasi algoritma *Timestamp Nonce* dan nilai *Salt*. Kombinasi dari kedua algoritma tersebut akan menghasilkan sebuah nilai acak yang bersifat unik, nilai ini akan digunakan untuk mengenkripsi data pengguna pada tahap autentikasi sehingga dapat mengamankan data pengguna dari pencurian data dan serangan *Man in The Middle*. Penerapan kombinasi algoritma *Timestamp Nonce* dan nilai *salt* dapat dijadikan salah satu alternatif pengamanan autentikasi dalam sistem SSO.

METODE

Dalam penelitian ini, digunakan metode kualitatif untuk menganalisis penerapan kombinasi metode *timestamp nonce* dan nilai *salt* sebagai upaya untuk meningkatkan keamanan *login* pengguna pada sistem SSO. Pengujian akan dilakukan dengan menganalisis paket data pada skema SSO yang diusulkan. Analisa ini dilakukan untuk menanggulangi serangan MiTM dalam ruang lingkup pencurian dan pembacaan paket data.

Skenario uji coba dimulai setelah pengguna melakukan *login*. Setiap proses komunikasi paket data antara pengguna dan server akan dibaca menggunakan aplikasi *wireshark*. Selanjutnya, uji coba serangan MiTM dengan melakukan pembacaan setiap paket data serta melakukan uji coba dekripsi pada hasil variabel nama pengguna dan kata sandi menggunakan *database* md5 *online*.

Dalam penelitian ini, digunakan seperangkat komputer dan server dengan spesifikasi yang memadai untuk menjalankan aplikasi, serta beberapa perangkat lunak yang dibutuhkan seperti server dengan sistem operasi Ubuntu 20.04. Selain itu, berbagai perangkat lunak pendukung juga digunakan antara lain:

- *Visual Studio Code* sebagai perangkat lunak editor untuk memudahkan dalam menuliskan kode program.
- *Linux-CentOS* sebagai sistem operasi yang digunakan menggunakan IP statis 172.20.7.207.

- *NginX* sebagai web server yang digunakan untuk melayani permintaan serta memberikan respon yang dibutuhkan oleh aplikasi dan pengguna. Protokol yang digunakan adalah protokol HTTP dengan menggunakan *port* 80.
- *Mariadb* sebagai *database* server yang digunakan untuk menyimpan data aplikasi
- *PHP* dan *Laravel Passport* sebagai bahasa pemrograman serta kerangka kerja *Single Sign-On* yang bersifat *open source*.
- Aplikasi *database* md5 berbasis web untuk melakukan dekripsi atau proses *reverse engineering* pada variabel nama pengguna dan kata sandi dengan menggunakan aplikasi md5decrypt.net.

Berikut adalah tahapan proses autentikasi SSO menggunakan kombinasi *timestamp nonce* dan nilai *salt* :

1. Pengguna melakukan *login* dan mengajukan permintaan untuk mengakses halaman masuk pada aplikasi.
2. Aplikasi mengirim permintaan ke server otorisasi untuk menampilkan halaman *login*, dengan melampirkan informasi seperti *client_id, response_type, scope,* dan *redirect_uri*.
3. Server otorisasi memvalidasi permintaan dari aplikasi dengan memeriksa apakah *client_id* telah terdaftar pada server otorisasi, apakah *client_id* cocok dengan *redirect_uri* yang diberikan, apakah pengguna telah memberikan izin untuk *response_type* dan *scope* yang diminta oleh aplikasi.
4. Jika permintaan aplikasi telah dinyatakan valid oleh server otorisasi, maka server otorisasi akan menampilkan halaman *login* ke server aplikasi.
5. Setelah permintaan aplikasi divalidasi oleh server otorisasi, server tersebut akan menampilkan halaman *login* ke pengguna.
6. Pengguna memasukkan informasi akun, yaitu nama pengguna dan kata sandi, lalu menekan tombol *login*.
7. Aplikasi memulai inisiasi *timestamp nonce* dengan mengirimkan *client_time*, *fingerprint*, dan *action* ke server otorisasi. *Client_time* adalah waktu ketika aplikasi mengirim permintaan ke server otorisasi dalam bentuk variabel *integer epoch*. *Fingerprint* merupakan *string* acak yang digunakan sebagai kredensial pada setiap komunikasi data. *Action* berisi fitur yang diminta aplikasi kepada server otorisasi.
8. Setelah menerima permintaan dari aplikasi, server otorisasi membuat *server_time* dan *server_nonce* (*snonce*). Selanjutnya, server otorisasi menghitung *delta_time* dengan menggunakan persamaan sebagai berikut:

$$\delta t = \left| \frac{(t_{server} - t_{client})}{21600} \right| \qquad (1)$$

Keterangan :

δt : Perubahan waktu antara waktu server dan waktu klien (berupa bilangan positif)

t_{server} : Waktu server

t_{client} : Waktu klien

Setelah mendapatkan *delta time*, server akan menggenerate nilai *salt* yang akan digunakan. nilai *salt* akan dinotasikan menjadi *GSALT* dan akan didapatkan dengan persamaan sebagai berikut:

$$GSALT = sha256(SALT, delta_{time}, fingerprint) \quad (2)$$

Keterangan :

$GSALT$	:	Nilai *global salt* yang akan digunakan untuk komunikasi
$sha256$	:	Algoritma *hashing*
$delta\_time$	:	Perbedaan waktu antara waktu server dan waktu klien, berupa bilangan positif
fingerprint	:	*Random string* yang diberikan oleh *client*

Setelah mendapatkan nilai *GSALT*, server dapat membuat *snonce*. *Server_nonce* adalah nilai unik yang dibuat oleh server otorisasi. Nilai ini hanya digunakan satu kali saja dan didapatkan melalui proses *hashing* dengan menggunakan algoritma md5. Persamaan yang digunakan untuk menghasilkan nilai *server_nonce* adalah sebagai berikut:

$$snonce = md5\left(GSALT, t_{server}, client_{ip}, user_{agent}, action\right) \quad (3)$$

Keterangan :

$snonce$	:	*Server_nonce*
$md5$	:	*Message-digest algorithm* 5 merupakan sebuah metode kriptografi *hash* yang menghasilkan nilai *hash* sebesar 128 bit.
$GSALT$	:	*Global Salt* adalah sebuah *string* yang digunakan sebagai tanda tangan unik dalam proses *hashing*.
t_{server}	:	*Server_time*
$client\_ip$	:	*IP Address* pengguna ketika mengirim *request* ke server otorisasi.
$user\_agent$	:	Informasi browser yang digunakan oleh pengguna.
$action$	:	Nilai variabel yang dikirimkan oleh aplikasi.

9. Aplikasi menerima *server_nonce, server_time, delta_time, gsalt,* dan *fingerprint* dari server otorisasi.
10. Aplikasi akan melakukan pencocokan *delta_time* yang diterima dan *delta_time* yang dihitung ulang. Serta akan mencocokan nilai *gsalt* yang diterima dan nilai *GSALT* yang dihitung ulang dan mencocokan nilai *fingerprint* yang diterima dengan nilai *fingerprint* yang berada di *client.*
11. Aplikasi melakukan perhitungan untuk mendapatkan nilai *client_nonce* dengan menggunakan persamaan sebagai berikut :

$$cnonce = md5(GSALT, t_{client}, snonce, action) \quad (4)$$

Keterangan :

cnonce	:	*Client_nonce*
md5	:	*Message-digest algorithm* 5 merupakan sebuah metode kriptografi *hash* yang menghasilkan nilai *hash* sebesar 128 bit.
GSALT	:	*Global Salt* adalah sebuah *string* yang digunakan sebagai tanda tangan unik dalam proses *hashing*.
t_{server}	:	*Server_time*
snonce	:	*Server_nonce*
action	:	Nilai variabel yang dikirimkan oleh aplikasi.

Setelah mendapatkan nilai *client_nonce*, aplikasi akan mengirimkan kembali nilai *server_nonce* dan *client_nonce* ke server otorisasi.

12. Selanjutnya, *client* akan mengirimkan *cnonce* untuk mendapatkan otp (*one time* kata sandi) ke server untuk mendapatkan ijin melakukan autentikasi pengguna.
13. Server akan membuat kode otp dengan persamaan sebagai beikut :

$$OTP = sha256(GSALT, snonce, delta_{time}, useragent, client_{ip}, machine_{id}) \quad (5)$$

Keterangan :

OTP	:	Nilai *global salt* yang akan digunakan untuk komunikasi
$sha256$	:	Algoritma *hashing*
$GSALT$	:	*Global Salt*
$snonce$	:	Server *nonce*
$cnonce$	:	*Client nonce*
$delta_{time}$	:	Selisih waktu antara waktu server dan waktu klien (berupa

bilangan positif)

$user_{agent}$: Informasi browser yang digunakan oleh *client*

$client_{ip}$: *Client IP address*

$machine_{id}$: Id mesin server linux yang bersifat unik

14. Setelah menerima OTP, server otorisasi akan melakukan *hashing* pada data nama pengguna dan kata sandi yang telah diisi oleh pengguna sebelumnya.
15. Sandi yang dimasukkan oleh pengguna akan dijadikan sebuah *hash* dengan menggunakan persamaan berikut :

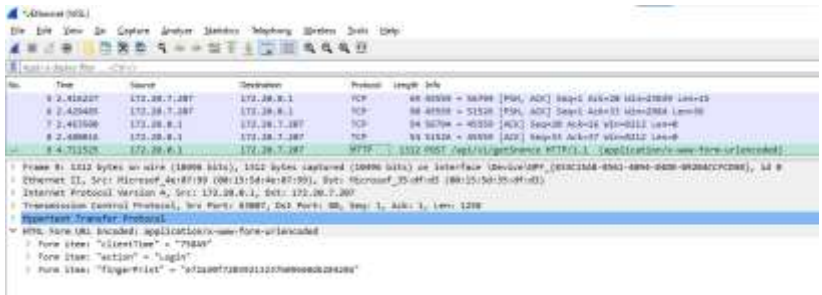
$$cphash = md5(otp, fingerprint, gsalt, snonce, username,\ cnonce) \quad (6)$$

$$cphash = md5(otp, fingerprint, gsalt, snonce, md5(password), cnonce) \quad (7)$$

16. Server otorisasi melakukan autentikasi *cuserhash* dan *cpasshash* ke *database*.
17. Server otorisasi mengirimkan *authorization_code* ke aplikasi.
18. Aplikasi membalas dengan mengirimkan *authorization_code*, *client_id* dan *client secret* ke server otorisasi
19. Server otorisasi melakukan pengecekan *authorization_code*.
20. Server otorisasi mengirimkan *access_token* ke pengguna.
21. Pengguna membalas dengan mengirimkan permintaan menggunakan *access_token* ke server aplikasi.
22. Server aplikasi meminta server otorisasi untuk memvalidasi token.
23. Server aplikasi menerima respons dari server otorisasi yang berisi detail *access_token*.
24. Server aplikasi mengirimkan data *respon* ke aplikasi pengguna.
25. Aplikasi pengguna menampilkan informasi atau data yang diminta pengguna.

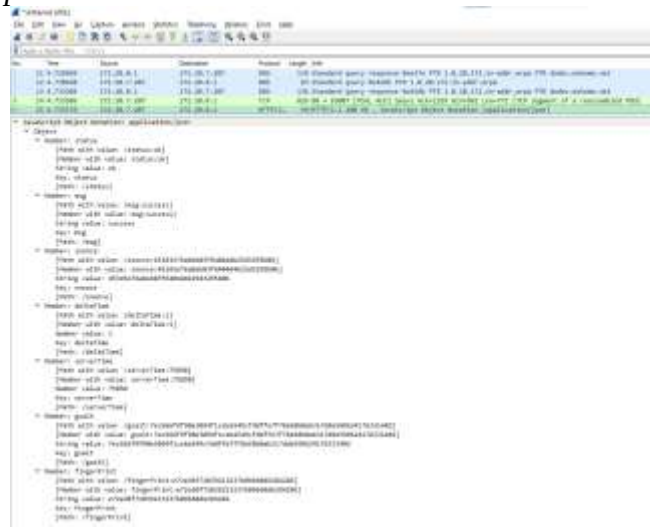
HASIL DAN PEMBAHASAN

Hasil didapatkan dari analisa paket menggunakan aplikasi *wireshark*. Setelah pengguna mengisi nama pengguna dan kata sandi, maka *client* akan membuat sebuah variabel *fingerprint* yang berisi *random string*, mendapatkan *client_time* dan akan mengirimkannya ke server otorisasi untuk mendapatkan *snonce*.



Gambar 2. Variabel clientTime,action dan fingerprint yang dikirim client.

Pada Gambar 2 dapat dilihat bahwa *client* mengirim 3 variabel, yaitu *clientTime*, *action* dan *fingerprint* dalam bentuk POST.

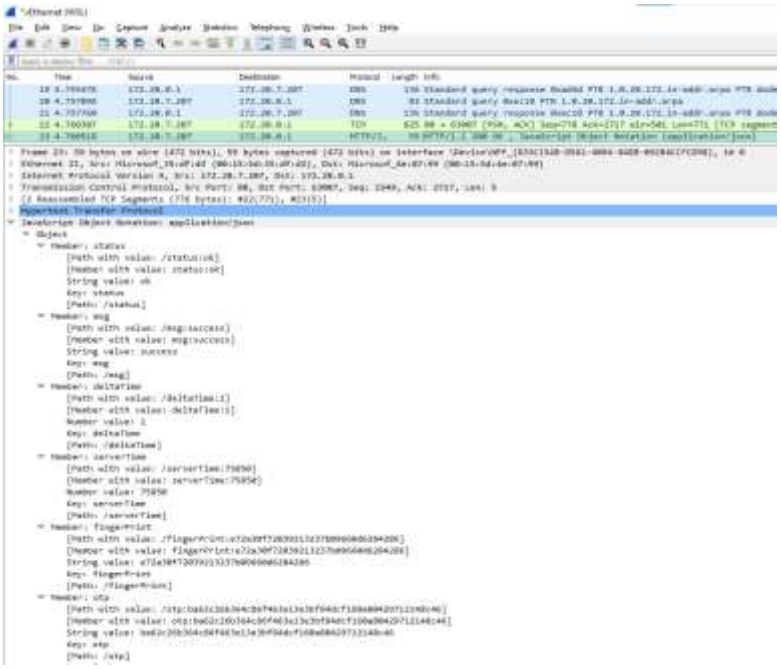


Gambar 3. Respon yang dikirimkan server.

Selanjutnya server akan memberikan *respon* berupa status dan *message* yang berisi informasi valid tidak suatu *request*, *snonce*, *deltatime*, *servertime*, *gsalt*, dan *fingerprint*. Lantas *client* akan memverifikasi nilai *delta time* dengan melakukan penghitungan *delta time* ulang di *client* serta akan mencocokan nilai *fingerprint* yang diterima dengan variabel *fingerprint* yang berada di *client* dan mencocokan nilai *GSALT* dengan nilai penghitungan ulang *GSALT* pada *client*. Apabila cocok, *client* akan menghitung *cnonce* dan akan mengirimkan nya ke server untuk mendapatkan nilai otp. Hal ini bisa dilihat pada Gambar 3.

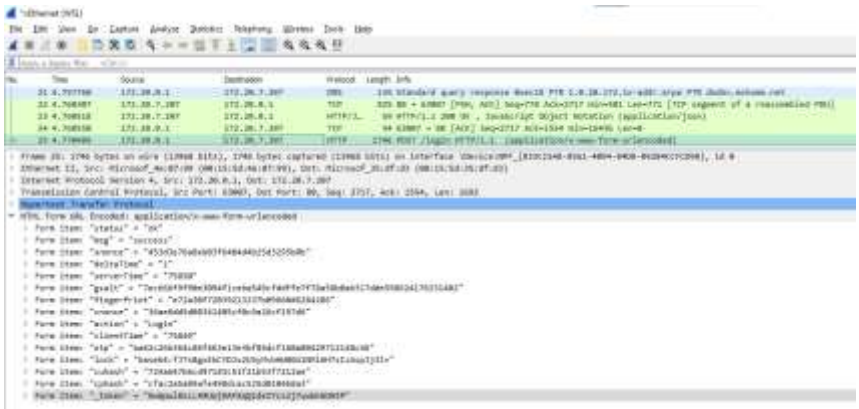
Gambar 4. Request Client ke Server Otorisasi.

Pada Gambar 4 merupakan bentuk *request* yang diberikan oleh *client* ke server *otorisasi*. Lantas server *otorisasi* akan memverifikasi nilai *snonce*, *deltatime*, *gsalt* dan *cnonce* dengan penghitungan masing masing pada server otorisasi. Setelah verifikasi dilalui, maka server akan memberikan respon informasi nilai OTP. Hal ini bisa dilihat pada Gambar 5.



Gambar 5. Respon Server yang Memberikan Kode OTP

Setelah mendapatkan nilai OTP, maka *client* akan melakukan *hashing* pada nilai nama pengguna dan kata sandi. Sehingga *request* untuk *login* berupa pada Gambar 6. Didapati pada *request* tersebut, nilai nama pengguna dan kata sandi telah berubah menjadi *cuhash* dan *cphash*. Nilai tersebut tidak bisa digunakan ulang karena diper*ole*h dengan metode *hashing* antara nilai *snonce, cnonce*, *gsalt*, otp dan *fingerprint*. Penulis menggunakan algoritma md5 dan sha256 karena merupakan algoritma yang bersifat *irreversible*, sehingga kemungkinan peretas mengguakan metode MiTM tidak memungkinkan. Karena variabel yang digunakan untuk autentikasi pengguna tidak bisa digunakan berulang kali. Percobaan MiTM untuk melaukan pencurian data berdasarkan variabel *cuhash* dan *cphash* akan diujicoba *reverse engineering* menggunakan database md5 yang tersedia pada aplikasi md5decrypt.net. Hasil bahwa kedua variabel tersebut bersifat unik dan tidak bisa di dekripsi ditunjukan pada Gambar 7 dan Gambar 8.

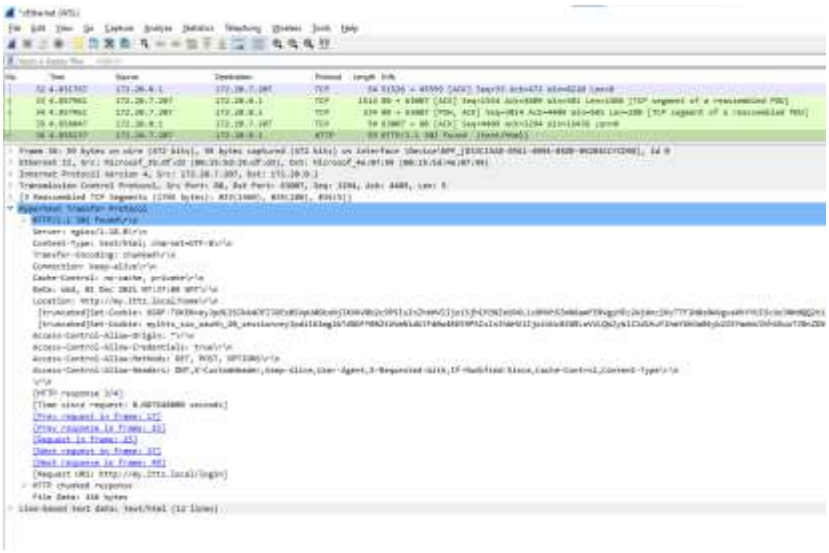


Gambar 6. Client Request Login

Gambar 7. Hasil reverse engineering variabel cuhash

Gambar 8. Hasil reverse engineering variabel cphash

Selanjutnya, server *otorisasi* akan mencocokan nilai *cuhash* dan *cphash* dengan di *database*. Pada mySQL memungkinkan untuk melakukan *query* dengan memberikan *hashing* pada kondisi yang akan dicari. Selanjutnya, server *otorisasi* akan mencocokan nilai *cuhash* dan *cphash* dengan di *database*. Pada mySQL memungkinkan untuk melakukan *query* dengan memberikan *hashing* pada kondisi yang akan dicari.



Gambar 9. Hasil Respon Autentikasi dari Server Otorisasi

Hasil respon autentikasi dari server otorisasi dapat dilihat pada Gambar 9, dimana *client* akan memberikan informasi *header location*, dimana *client* akan meneruskan halaman ke nilai dari *location* yang diberikan oleh server aplikasi.

KESIMPULAN

Kesimpulan yang dapat diambil dari penelitian ini adalah autentikasi pengguna menggunakan kombinasi *timestamp nonce* dan nilai *salt* tidak dapat diretas menggunakan metode *man in the middle attcak* pada ruang lingkup pencurian dan pembacaan paket data. Hal ini karena variabel yang digunakan untuk autentikasi pengguna bersifat unik dan tidak bisa digunakan berulang kali.

SARAN

Penilitian ini merupakan awalan dari pengembangan sistem aplikasi SSO. Dibutuhkan analisa lebih lanjut dari metode yang diusulkan dengan metode-metode pengetesan keamanan data dan jaringan.

DAFTAR RUJUKAN

Wahyudiyono. (2016). Penggunaan Teknologi Informasi Dan Komunikasi Di Nusa Tenggara Barat. *Jurnal Komunika : Jurnal Komunikasi, Media Dan Informatika*, *5*(1), 29. https://doi.org/10.31504/komunika.v5i1.636

Anani, W., & Ouda, A. (2017). The importance of human dynamics in the future user authentication. *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, 1–5. https://doi.org/10.1109/CCECE.2017.7946790

Aminudin, A. (2014). Implementasi Single Sign On (SSO) Untuk Mendukung Interaktivitas Aplikasi E-Commerce Menggunakan Protocol Oauth. *Jurnal Gamma*, *10*(1), 109–115.

Bajdor, P. (2015). The Use of Information and Communication Technologies in Polish Companies in Comparison to Companies from European Union. *Procedia Economics and Finance*, *27*(15), 702–712. https://doi.org/10.1016/s2212-5671(15)01051-5

Beltran,V. (2016). Characterization of web single sign-on protocols. IEEE Communications Magazine (2016) 54(7) 24-30

Jia, L. (2020). Research on Information Security of Large Enterprises. *2020 IEEE 8th International Conference on Information, Communication and Networks, ICICN 2020*, 219–223. https://doi.org/10.1109/ICICN51133.2020.9205077

Karie, N. M., Kebande, V. R., Ikuesan, R. A., Sookhak, M., & Venter, H. S. (2020, March 31). Hardening SAML by Integrating SSO and Multi-Factor Authentication (MFA) in the Cloud. *ACM International Conference Proceeding Series*. https://doi.org/10.1145/3386723.3387875

Yang, T. J., & Yang, X. J. (2014). Method of single sign-on for independent web systems based on AJAX. *Proceedings of 2013 3rd International Conference on Computer Science and Network Technology, ICCSNT 2013*, 310–314. https://doi.org/10.1109/ICCSNT.2013.6967119

"OAuth 2.0." Accessed: Jun. 21, 2021. [Online]. Available: https://oauth.net/.

Febrian, T. (2017). *Memahami OAuth 2.0 (API Security)*. https://medium.com/codelabs-unikom/memahami-oauth-2-0-api-security-9376bc3a307b

Z. Triartono, R. M. Negara, and Sussi, "Implementation of Role-Based Access Control on OAuth 2.0 as Authentication and Authorization System," in 2019 6th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI), 2019, pp. 259–263. doi: 10.23919/EECSI48112.2019.8977061

M. Kihara and S. Iriyama, "Security and performance of single sign-on based on one-time pad algorithm," Cryptography, vol. 4, no. 2, pp. 1–29, 2020, doi: 10.3390/cryptography4020016.

Ramamoorthi, L. S., & Sarkar, D. (2020). Single sign-on: A solution approach to address inefficiencies during sign-out process. *IEEE Access*, *8*, 195675–195691. https://doi.org/10.1109/ACCESS.2020.3033570

Radha, V., & Reddy, D. H. (2018). A Survey on Single Sign-On Techniques. *Procedia Technology*, *4*(2), 134–139. https://doi.org/10.1016/j.protcy.2012.05.019

G. M. Køien, "A Brief Survey of Nonces and Nonce Usage," SECURWARE 2015: The Ninth International Conference on Emerging Security Information, Systems and Technologies, no. c, pp. 85–91, 2015.

CheatSheets Series Team, "OWASP Cheat Sheet Series," 2021. https://cheatsheetseries.owasp.org (accessed Jul. 12, 2021).

Naiakshina, A., Danilova, A., Tiefenau, C., Herzog, M., Dechand, S., & Smith, M. (2017). Why Do Developers get password storage wrong? a qualitative usability study. *Proceedings of the ACM Conference on Computer and Communications Security*, 311–328. https://doi.org/10.1145/3133956.3134082

A. Mohammed Ali and A. Kadhim Farhan, "A novel improvement with an effective expansion to enhance the MD5 hash function for verification of a secure E-Document," IEEE Access, vol. 8, pp. 80290–80304, 2020, doi: 10.1109/ACCESS.2020.2989050

F. A. Alselami, "Blockchain and Bigdata to Secure Data Using Hash and Salt Techniques," Journal of Information Technology Management, vol. 14, no. 2, pp. 15–25, 2022, doi: 10.22059/JITM.2022.86924.

F. Ayankoya and B. Ohwo, "Brute-Force Attack Prevention in Cloud Computing Using One-Time Password and Cryptographic Hash Function," International Journal of Computer Science and Information Security (IJCSIS), vol. 17, no. 2, pp. 7–19, 2019.