KINETIK, Vol.1, No.2, Agustus 2016, Hal. 63-68
ISSN : 2503-2259,
E-ISSN : 2503-2267

Pengembangan Mekanisme Otentikasi dan Otorisasi Pada Manajemen Konfigurasi Untuk Kasus User Web Hosting Berbasis Kontainer

Syaifuddin[*1], Royyana Muslim[2], Baskoro Adi Pratomo[3]
[1]Universitas Muhammadiyah Malang, [2,3]Institut Teknologi Sepuluh Nopember Surabaya
udinsaif@gmail.com[*]

Abstrak

Sebagian besar penyedia layanan website menggunakan sistem operasi Linux. Ketika salah satu website dalam web server dapat diambil alih, kemungkinan besar website yang lain juga akan dapat diambil alih dengan cara membaca konfigurasi penghubung ke database. Mekanisme yang digunakan untuk membaca sebuah file konfigurasi dengan perintah di linux secara default memang tersedia, dengan menggunakan perintah "ln" yang dikenal dengan istilah "Symlink" yang dapat membaca direktori konfigurasi web walaupun berbeda direktori. Hasil penelitian yang dilakukan, konfigurasi yang terdapat di dalam aplikasi web yang berada direktori dalam satu server, walaupun tetap terbaca dengan menggunakan metode tersebut, namun tidak dapat di-decode untuk membaca username dan password, serta database name, karena sudah diberi otorisasi yang bisa medecode hanya dari direktori yang sudah terdaftar, sedangkan pada pengujian performa untuk latency, memory, dan CPU system yang diusulkan tidak sebagus dengan system sebelumnya, namun dengan menggunakan cache, respond time yang dihasilkan ketika diakses secara simultan dengan 20 kali klik per-user menunjukkan system yang lama sebesar 941,4 ms, sedangkan untuk sistem yang diusulkan sebesar 786,6 ms.

Kata kunci: *Cybercrime, De-Militarised Zone, Docker engine, Symlink*

Abstract

Most of the website providers are using Linux as its operation system. If one website in a web server could be taken over by reading the configuration connector to database, so is the other website. The mechanism of reading the configuration directory in Linux is available by default, by using command "in" – known as "Symlink" – which could read the web configuration even though they are in different directory. The result shows that even if the configuration in web application in one server is able to read, it could not be decoded to read the username and password, as well as the database name, because it has been authorized to decode only from the registered directory. Meanwhile the suggested performance evaluation for the latency, memory, and CPU system is not as good as the previous one. However, by using cache, the respond time needed based on the simultaneous access of 20 clicks per user of the old system is 941.4 and the suggested system is 786.6 ms.

Keywords: *Cybercrime, De-Militarised Zone, Docker engine, Symlink*

1. Pendahuluan

Semakin majunya teknologi saat ini dirasa seperti dua sisi mata uang yang berbeda, disisi lain sangat membantu terutama dibidang komunikasi yang membuat jarak tidak berarti. Seperti adanya sosial media yang menggunakan teknologi *website* dengan berbagai jenis *platform* bahasa pemrograman, seperti PHP yang paling populer di kalangan penyedia layanan *Web Hosting*. Namun disisi lain banyaknya kejahatan *cyber*, seiring berkembangnya teknologi menjadi kompetisi di kalangan para *cybercrime* untuk menjadi yang paling unggul melakukan penetrasi *website* dan hasilnya diunggah pada situs (*Zone*-h arsip penyerangan digital, http://www.*Zone*-h.org) sebagai tanda telah berhasil [1].

Metode yang berbeda dengan tujuan yang sama untuk mengamankan konfigurasi yang dapat diakses dari direktori lain, yaitu menggunakan suEXEC [1], merupakan sebuah fitur atau modul yang dapat disisipkan ke dalam sebuah *web server*, terutama Apache yang membuat semua *script* php yang dibuat dapat memiliki otoritas seperti *Unix user*. Contohnya, ketika akan

Makalah dikirim 30 Juni 2016; Revisi 30 Juli 2016; Diterima 1 Agustus 2016

membuat sebuah *file* tidak perlu mengganti otoritas *file* tersebut, karena pada fungsi PHP suEXEC dijalankan pada masing-masing *user*.

Beberapa metode tersebut menurut [2] dapat di *bypass* dengan menggunakan *symbolic link*, memungkinkan untuk mengabaikan otoritas yang telah dibuat oleh suEXEC. Sedangkan mekanisme *open_basedir* dilakukan konfigurasi pada php ini yang diletakkan pada setiap direktori di setiap *user*, sehingga ketika *file* tersebut diubah atau diganti dengan php yang dibuat oleh *attacker* maka fungsi yang telah dinonaktifkan menjadi aktif kembali.

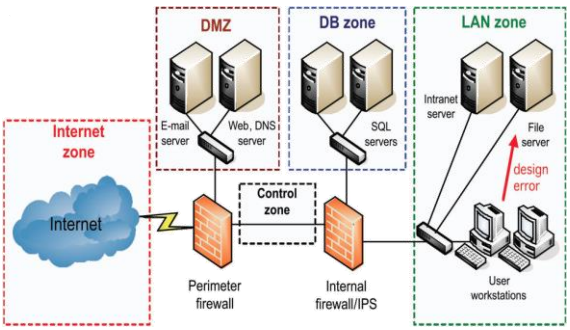
Dari beberapa penelitian yang disebutkan muncul sebuah hepotosa, bahwa jika semua konfigurasi dijadikan satu dalam sebuah *server* konfigurasi yang terhubung dengan *server* aplikasi *website* pada *user web*, mengacu pada penelitian [3] menggunakan konsep *De-Militarized Zone* (DMZ) yang memisahkan jaringan internal dengan jaringan publik. Secara esensial, DMZ melakukan perpindahan semua layanan suatu jaringan ke jaringan lain yang berbeda dengan memberikan aturan-aturan sesuai dengan kebutuhan. Secara umum DMZ dibangun berdasarkan tiga buah konsep, yaitu: NAT (*Network Address Translation*), PAT (*Port Addressable Translation*), dan *Access List.*

Penggunaan DMZ pada konsep ini menggunakan Docker *engine*, Docker membuat proses pemaketan aplikasi bersama komponennya *dependencies* secara cepat dalam sebuah *container* yang terisolasi, sehingga dapat dijalankan dalam infrastruktur lokal data *center* ataupun *cloud* tanpa melakukan perubahan konfigurasi lagi pada *container*, selama *host* menjalankan Docker *engine* [2]. Proses pembacaan *file* konfigurasi dari aplikasi *web* selain menggunakan mekanisme DMZ juga menggunakan mekanisme *representational state transfer* (REST), yang merupakan salah satu jenis *web service* yang menerapkan konsep perpindahan antar *state* berorientasi pada *resource* [4], sehingga proses pembacaan *file* konfigurasi yang berada pada *server* saling terhubung melalui beberapa *link* HTTP.

2. Metode Penelitian

Langkah pertama yang harus dilakukan adalah mempersiapkan lingkungan *server* konfigurasi menggunakan Docker. Di sisi lain mempersiapkan lingkungan yang hampir sama dengan *User Web Hosting* dengan beberapa komponen terkait, konsep pemisahan konfigurasi mengacu pada mekanisme *Demilitarized Zone* [5] dengan cara membagi beberapa jaringan ke dalam beberapa zona yang mempunya ketentuan masing-masing, seperti pada Gambar 1.

Pemisahan *server* antara aplikasi *web* dengan konfigurasi yang berbeda *server* menggunakan arsitektur dari Docker diimplementasikan dalam sebuah *container* terisolasi, dan hanya dapat dijalankan dalam jaringan lokal tanpa melakukan perubahan atau konfigurasi pada *server*. *Container* merupakan sebuah *image* bersifat *read write* yang berjalan di atas *image*.

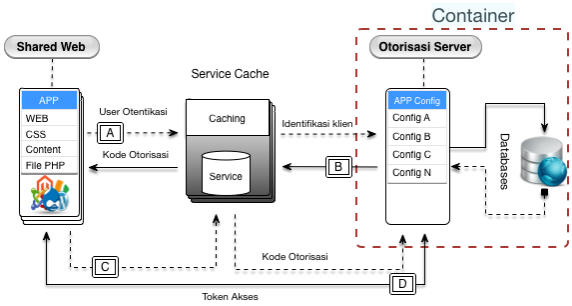


Gambar 1. Topologi Proteksi System Resources [5]

Proses otentikasi mempunyai tujuan untuk memastikan apakah sesuatu yang datang (*principal*) sesuai dengan yang diakui, dalam mekanisme ini *server* konfigurasi akan memverifikasi apakah yang melakukan *request* benar-benar *user* yang asli, dengan cara melakukan verifikasi terhadap beberapa *principal* dan *credential*. Dalam tahapan mekanisme Gambar 1, aplikasi *web* mempunyai konfigurasi tersendiri yang dibuat untuk mengakses *konfigurasi* yang sebenarnya berada di dalam *server* lain sebagai penghubung untuk bisa mengakses ke dalam *database*.

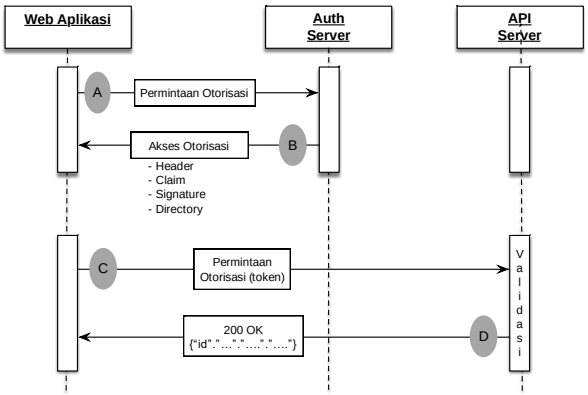
Server konfigurasi yang dibangun dari Docker juga menggunakan sistem Ubuntu yang sama, dengan beberapa servis, seperti Apache dipilih sebagai *server web* karena sudah banyak digunakan dan terbukti keandalannya. Di dalam Docker terdapat tiga *container* sebagai penunjang *server* konfigurasi, yaitu *container* dengan id=6153260cabf1 sebagai *server*

konfigurasi, yang berisi API dari beberapa aplikasi *website*, sedangkan *container* yang lain dengan id 3ab79d4eb5d9 dan 5f26c7501df5 berisikan *database*



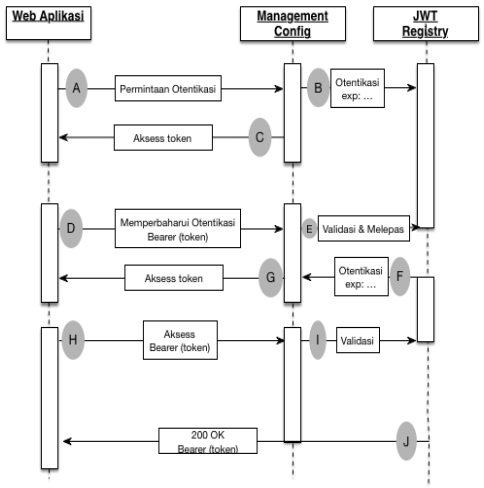
Gambar 2. Arsitektur Server Konfigurasi

Mekanisme proses otentikasi menggunakan mekanisme *Json Web Token* (JWT), dengan menambahkan beberapa mekanisme yang berbeda dari JWT pada umumnya, yaitu penambahan otorisasi berupa *directory* dan mekanisme untuk *decode token* menggunakan str_rot13 yang digabungkan dengan *base*64, dengan tujuan *token* yang dihasilkan tidak bisa di *decode* dari tempat lain. Dari sisi klien sendiri sudah dibuat *decode* khusus untuk melakukan *decoding* pada *token* yang dihasilkan, proses otentikasi tersebut pada Gambar 3.



Gambar 3. Arsitektur Otorisasi

Mekanisme pembatasan waktu *token* yang diberikan bertujuan untuk membatasi waktu penyerang yang dapat menyalahgunakan *token*, menurut [6] kode rahasia berupa *token* dapat berpotensi bocor dengan menggunakan serangan *forced-login* CSRF dengan cara melakukan *replay attack*, seperti menjadi klien yang resmi, penjelasan proses pembaharuan *token* yang akan dilakukan seperti Gambar 4.



Gambar 4. Arsitektur Otentikasi

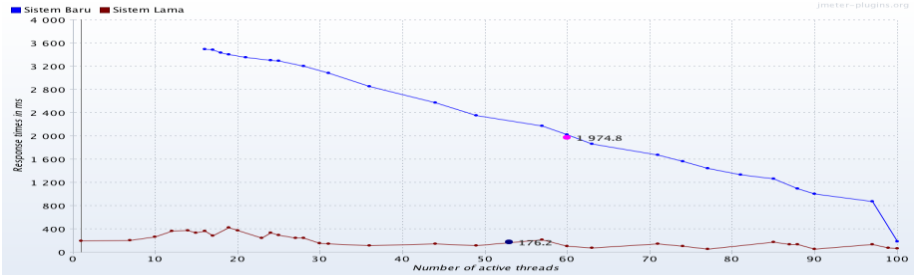
3. Hasil Penelitian dan Pembahasan

Uji coba yang dilakukan untuk mengukur performa akan dilakukan dengan beberapa skenario, salah satu pengukurannya adalah sumber daya memori dan CPU terhadap penerapan sistem yang baru, dibandingkan dengan sistem lama, ketika diakses secara berkala dengan skenario yang berbeda. Pengukuran *latency* juga akan menjadi uji coba untuk mengetahui performa dari sistem yang akan diterapkan. Pengujian ini akan dilakukan dengan cara mengakses sistem dengan jumlah *user* yang berbeda mulai dari 10 sampai dengan 100 *user*. Uji coba performa akan dilakukan dengan beberapa parameter.

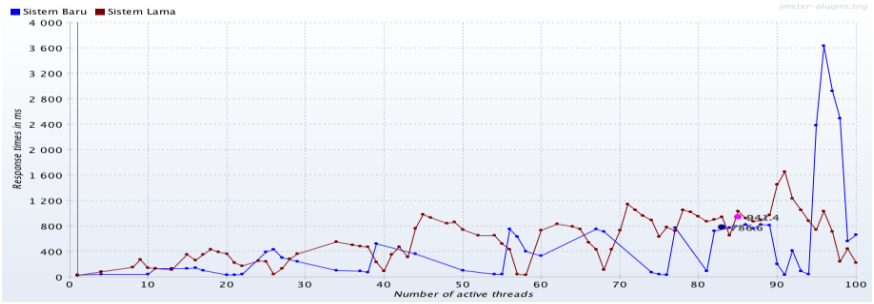
Tabel 1. Skenario Pengujian Performa

Jumlah *User*		100/1	100/20	300/1	300/20	900	900/20
Latency	x	1988	951	4615	1658	18786	7537
	y	175	789	2246	3364	3283	7684
Throughput	x	28	47.9	33.8	43.4	27.5	41.2
	y	33	48.7	35.9	44.1	34.5	42
Total *User*		100	2,000	300	6,000	900	18,000

Tabel 1 merupakan skenario pengujian performa, sedangkan Gambar 5 menunjukkan grafik jumlah *user* pada uji coba pertama sebanyak 100 *user* dengan sekali klik, Gambar 6 untuk uji coba kedua sebanyak 100 *user* namun untuk masing-masing *user* akan melakukan akses ke *server* sebanyak 20 kali, artinya ada sekitar 2000 *user* yang melakukan permintaan ke *server*, penambahan jumlah klik untuk mengetahui apakah *cache* yang sudah dibuat berjalan. Pada Gambar 7, pengujian sebanyak 300 *user*, merupakan tiga kali dari jumlah pengujian yang pertama. Pengujian performa ini dilakukan sampai dengan 900 *user* dengan masing-masing *user* melakukan 20 kali akses ke *server*, artinya ada sekitar 18.000 *user* melakukan akses ke *server,* seperti di Gambar 8, Gambar 9, dan Gambar 10.



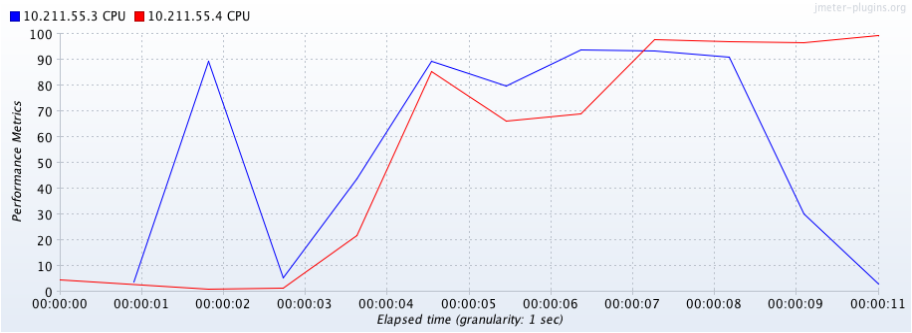
Gambar 5. Response Time 100 User dengan Sekali Akses



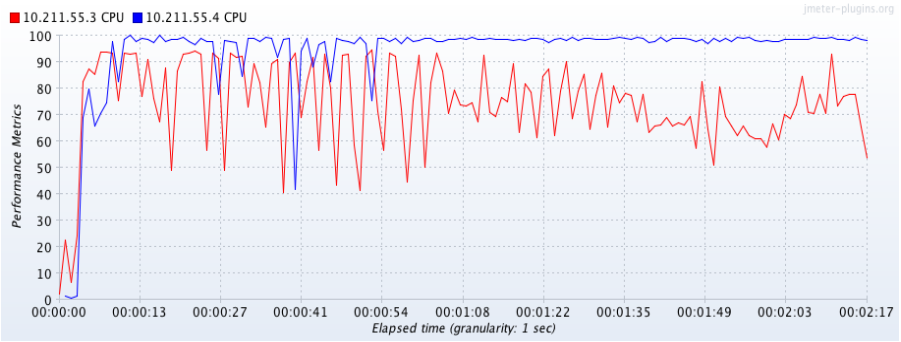
Gambar 6. Respons Time 100 User dengan 20 Kali Akses

Uji coba performa pada respons *time* bertujuan untuk mengukur respons yang diberikan *server* terhadap *user* yang melakukan permintaan secara berkala sesuai dengan skenario yang ada pada Tabel 1.

Uji performa dilakukan untuk mengetahui pengaruh *system* yang diterapkan terhadap penggunaan CPU. Uji coba dilakukan dengan membandingkan beberapa *user* secara berkala sesuai dengan skenario pada Tabel 1. Pada uji coba ini *system* yang diterapkan menggunakan IP *Address* 10.211.55.3 sedangkan untuk *system* yang lama dengan IP 10.211.55.4.

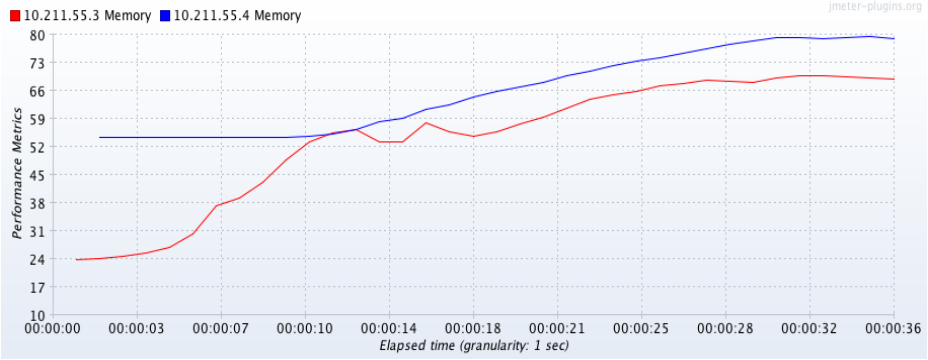


Gambar 7. Penggunaan CPU 300 User dengan Sekali Akses

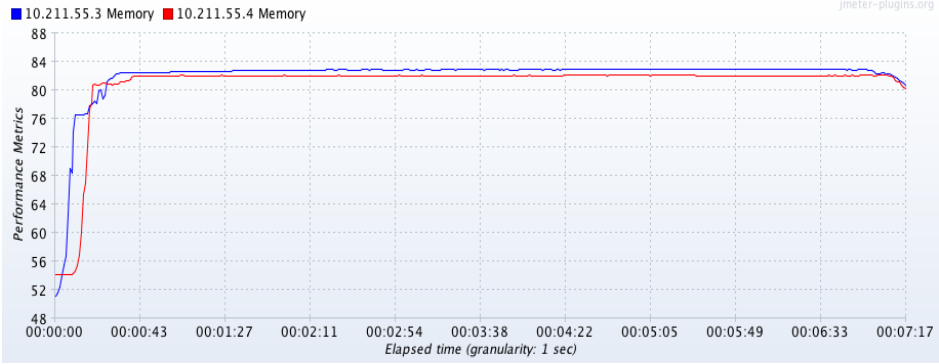


Gambar 8. Penggunaan CPU 300 User dengan 20 Kali Akses

Memori (RAM) merupakan tempat penyimpanan data sementara dari aplikasi sebelum diproses oleh CPU. Uji coba ini dilakukan untuk mengetahui pengaruh mekanisme perbaikan *reporting* protokol terhadap penggunaan memori. Pengujian dilakukan dengan membandingkan penggunaan memori mekanisme pada penelitian ini (mekanisme otentikasi dan *management* konfigurasi) dengan penggunaan CPU dari mekanisme sebelumnya (konfigurasi dan aplikasi *web* satu *server*).



Gambar 9. Penggunaan Memori 300 User dengan Sekali Akses



Gambar 10. Penggunaan Memori 300 User dengan 20 kali Akses

Uji coba keamanan untuk akses *token* yang dihasilkan dari direktori lain dilakukan juga dengan skenario, *token* yang ada pada aplikasi *web* A dijalankan pada aplikasi *web* B setelah dilakukan uji coba *symbolic link*, seperti Tabel 2.

Tabel 2. Pengujian Decode Token

No	Direktori	Hasil
1.	/home/*user*A/public_html/wp/	Bisa
2.	/home/*user*A/public_html/wp2/	Tidak Bisa
3.	/home/*user*B/public_html/wp/	Tidak Bisa
4.	/home/*user*C/public_html/wp/	Tidak Bisa
5.	/var/www/html/wp/	Tidak Bisa
6.	/var/www/html/wp2/	Tidak Bisa

Pengujian ini dilakukan dengan cara memindahkan *token* yang akan di *decode* pada pada direktori lain di dalam satu *server*. Pada tabel pengujian *decode token*, dari *user* A dengan direktori /home/*user*A/public_html/wp/ berhasil dilakukan dari direktori sendiri, sedangkan ketika dilakukan *decode* dari direktori lain atau beda *user* tidak bisa dilakukan *decode*.

4. Kesimpulan

Dari langkah-langkah penelitian yang telah dilakukan, mulai dari studi literatur sampai uji coba, penelitian ini dapat menyimpulkan sebagai berikut:

1. Pada mekanisme yang diusulkan, penggunaan sumber daya dari CPU dan *memory* tidak sebaik dengan *system* yang sebelumnya.
2. Penggunaan *cahce* yang diterapkan pada mekanisme yang diusulkan berfungsi ketika satu *user* mengakses beberapa kali terhadap *system*, terlihat dari *respond time* yang dihasilkan ketika diakses secara simultan dengan 20 kali klik per-*user* menunjukkan untuk *system* yang lama sebesar 941,4 Ms, sedangkan untuk *system* yang diusulkan sebesar 786,6 Ms.
3. Pengujian keamanan mampu mengatasi teknik *symbolic link* karena *username* dan *password* yang terdapat di dalam *konfigurasi* terlindungi oleh *token*, hanya direktori yang sah dapat melakukan *decode token*.

5. Pengembangan

Terdapat beberapa hal yang bisa dikembangkan untuk meningkatkan kinerja metode yang diajukan dalam penelitian ini. Saran pengembangan tersebut antara lain:

1. Pengembangan fitur *generate token* yang langsung otomatis membuat *konfigurasi* pada masing-masing *web* atau beberapa jenis CMS yang sering digunakan.
2. Pengembangan pada *plugins* sebagai *widgets* khusus untuk CMS *Wordpress* untuk mengganti *konfigurasi* yang belum terlindungi.
3. Pembuatan *server data Base* serta *server konfigurasi* secara *online* sehingga bisa dilakukan kerja sama dengan penyedia layanan *web hosting*.

Referensi

[1] Mirheidari, Seyed Ali, et al. "A Comprehensive Approach to Abusing Locality in User Web Hosting Servers." Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on. IEEE, 2013.
[2] Bernstein, David. "Containers and cloud: From lxc to Docker to kubernetes." IEEE Cloud Computing 3 (2014): 81-84.
[3] Suzuki, Etsuko. "A design of authentication system for distributed education." Information Technology Based Higher Education and Training, 2004. ITHET 2004. Proceedings of the FIfth International Conference on. IEEE, 2004.
[4] Gao, Lei, Chunhong Zhang, and Li Sun. "RESTful web of things API in sharing sensor data." Internet Technology and Applications (iTAP), 2011 International Conference on. IEEE, 2011.
[5] Stawowski, Mariusz. "The Principles of Network Security Design." ISSA Journal (2007): 29-31.
[6] Niu, Zhixiang, Cheng Yang, and Yingya Zhang. "A design of cross-terminal web system based on JSON and REST." Software Engineering and Service Science (ICSESS), 2014 5th IEEE International Conference on. IEEE, 2014.