Pembuatan Konfigurasi SSL yang Aman untuk Diimplementasikan pada Apache dan Nginx

Eka Wijaya Budihardjo, Lily Puspa Dewi, Agustinus Noertjahyana.
Program Studi Informatika, Fakultas Teknologi Industri, Universitas Kristen Petra
Jl. Siwalankerto 121-131 Surabaya, 60236
Telp. (031) – 2983455, Fax. (031) - 8418658
E-mail: ekabudiharjo1999@gmail.com, lily@petra.ac.id, agust@petra.ac.id

ABSTRAK

Eksploitasi terhadap *web server* merupakan bentuk perilaku seseorang yang mengambil keuntungan dari kerentanan suatu *software* atau kelemahan keamanan. Eksploitasi terhadap *web server* dapat berupa *Zombie Poodle*, *Golden Doodle*, *Poodle*, dan lain-lain. Selain serangan yang menggunakan validasi *padding*, juga ada serangan *man–in–the–middle–attack* seperti *protocol downgrade attack* dan *cookie hijacking*. Oleh karena itu, penelitian ini bertujuan untuk meneliti keamanan sebuah *web server* berdasarkan konfigurasi *SSL* dan memberikan saran konfigurasi *SSL* yang lebih baik dan aman.

Pengujian pada penelitian ini menggunakan program yang bernama *SSLyze* dan penelitian ini menghasilkan sebuah laporan terperinci yang dapat membantu untuk melakukan konfigurasi *SSL* pada *Apache web server* dan *Nginx web server* dengan baik dan benar agar dapat mencegah terjadinya serangan. Selain meningkatkan keamanan pada *web server*, dengan menggunakan konfigurasi *SSL* yang tepat dapat membuat kinerja server menjadi lebih optimal.

Hasil pengujian menunjukkan bahwa implementasi konfigurasi SSL yang diuji mempunyai tingkat keamanan yang baik sehingga dapat mencegah terjadinya masalah keamanan pada web server serta dapat membuat kinerja web server lebih optimal dan efisien.

Kata Kunci: konfigurasi SSL, *Transport Layer Secure*, *Secure Socket Layer*, Apache *web server*, Nginx *web server*, keamanan *web*

ABSTRACT

Exploitation of a web server is a form of behavior of someone who takes advantage of a software vulnerability or security weakness. Exploitation of the web server can be in the form of Zombie Poodle, Golden Doodle, Poodle, and others. In addition, there are also man-in-the-middle-attacks such as protocol downgrade attacks and cookie hijacking. Therefore, this study aims to examine a web server based on SSL configuration and provide suggestions for a better and more secure SSL configuration.

Testing in this study uses a program called SSLyze and this research produces a detailed report that can help to configure SSL on the Apache web server and Nginx web server properly and correctly in order to prevent further attacks. In addition to increasing security on the web server, using the right SSL configuration can make web server performance become optimal.

The test results show that implementing the tested SSL configuration has a good level of security so that it can prevent security problems on the web server and can make web server performance become more optimal and efficient.

Keywords: *SSL configuration, Transport Layer Secure, Secure Socket Layer, Apache web server, Nginx web server, web security*

1. PENDAHULUAN

Masalah teknologi yang sering terjadi di dunia adalah eksploitasi terhadap *web server.* Eksploitasi terhadap *web server* merupakan bentuk perilaku seseorang yang mengambil keuntungan dari kerentanan suatu *software* atau kelemahan keamanan.

Eksploitasi terhadap *web server* dapat berupa *Zombie Poodle, Golden Doodle, Poodle,* dan lain-lain. Mereka adalah contoh eksploitasi terhadap *web server* yang memiliki kelemahan validasi *padding* untuk mendeskripsikan *ciphertext*. Selain serangan yang memanfaatkan validasi *padding*, juga ada serangan *man – in – the – middle – attack* seperti *protocol downgrade attack* dan *cookie hijacking*. Untuk mengatasi serangan-serangan diatas, maka konfigurasi dari SSL harus dilakukan dengan benar, yaitu dengan cara menyusun algoritma *cipher suite*, menerapkan *OCSP Stapling* dan *Content Security Policy Headers*. Oleh karena itu, penelitian ini bertujuan untuk meneliti keamanan sebuah *web server* berdasarkan konfigurasi SSL dan memberikan saran konfigurasi SSL yang lebih baik dan aman.

2. TINJAUAN STUDI

Banyak peneliti telah melakukan penelitian mengenai keamanan suatu web server berdasarkan konfigurasi SSL yang digunakan. Pengujian yang dilakukan menggunakan metode Frankencerts untuk melakukan *automated adversarial testing* dan *usage patterns*. Frankencerts merupakan sertifikat SSL yang dibuat secara special untuk menguji *validation code* di SSL/TLS. Frankencerts akan menggunakan beberapa sertifikat dan menggunakan mutasi acak untuk menghasilkan test certificate. Sedangkan untuk metode *usage patterns* digunakan untuk mendeteksi penggunaan API yang benar. Setelah terdeteksi, maka *usage pattern* akan diuji dengan model implementasi yang benar. Namun, meskipun metode yang digunakan mempunyai kelebihan dalam mendeteksi suatu konfigurasi SSL yang tidak baik, penggunaan metode ini tidak dapat menentukan konfigurasi apa yang harus dilakukan agar dapat membuat dan mengimplementasikan konfigurasi SSL dengan baik.

3. METODE

3.1 SSLyze

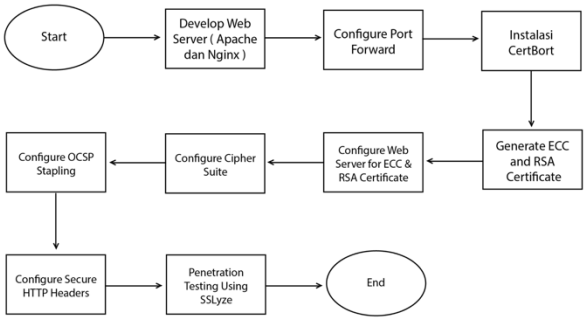
Merupakan *tools* yang dapat ditemukan di dalam Kali Linux OS. SSLyze dikembangkan oleh iSEC *Partners* dan dapat ditemukan secara manual di *github*. SSLyze sendiri digunakan untuk menemukan miskonfigurasi SSL dengan cara melakukan koneksi

ke *server*. SSLyze telah dibekali dengan berbagai *plugin* untuk membantu kita dalam melakukan *penetration testing* [1].

1. Dapat memeriksa versi SSL yang lama
2. Dapat menganalisa *cipher suite* yang lemah
3. Melakukan *scanning* lebih dari satu *server* menggunakan *input file*
4. Pengecekan terhadap *session resumption support*

3.2 Desain Sistem

Penelitian ini membahas analisa masalah dan rancangan pembuatan dari keseluruhan konfigurasi SSL yang akan dibuat. Penelitian ini dibuat agar dapat menjelaskan permasalahan yang dihadapi. Rancangan konfigurasi SSL yang tepat merupakan solusi yang akan diberikan untuk permasalahan yang ada, dan juga hasil analisa berupa laporan beberapa konfigurasi SSL yang dapat dilihat pada Gambar 1.



Gambar 1. Proses implementasi pada Apache dan Nginx

3.2.1 *Cipher Suite*

Bagian ini menjelaskan desain dan konfigurasi SSL yang di ajukan untuk *web server* Apache dan Nginx. Konfigurasi SSL akan diimplementasikan pada sistem komputer yang sudah di *install* Apache *Web Server* dan Nginx *Web Server*. Untuk konfigurasi *Cipher Suite* TLSv1.2 yang diajukan akan mengutamakan penggunaan ECDHE dan DHE sebagai *key exchange algorithm*, dikarenakan ECDHE dan DHE menggunakan *key* hanya sekali saja setiap melakukan enkripsi, lain dengan RSA, DH dan ECDH. Sehingga meskipun *private key* server sudah berhasil diambil oleh *attacker,* data di *web server* akan tetap aman [6].

Sedangkan untuk *authentication algorithm* mengutamakan penggunaan ECDSA dibandingkan RSA, dikarenakan ECDSA membutuhkan panjang *key* yang lebih kecil tetapi mempunyai tingkat keamanan yang sama dengan RSA. Dalam arti, panjang *key* ECDSA 192-*bit* mempunyai tingkat keamanan yang sama dengan 1024-bit RSA, dan untuk 256-*bit* ECDSA mempunyai tingkat keamanan yang sama dengan 2048-*bit* RSA. Ukuran *key* yang digunakan oleh ECDSA mempunyai ukuran yang kecil, sehingga *file* yang dienkripsi menggunakan ECDSA mempunyai ukuran *file* yang relatif lebih kecil daripada *file* yang dienkripsi menggunakan RSA. Selain itu, penggunaan ECDSA tidak membutuhkan *memory* dan *bandwith* yang besar [2]. Tabel 1 dan Tabel 2 menunjukkan perbandingan kecepatan RSA dan ECDSA dalam melakukan proses *signing* dan *verify*.

Tabel 1. Kecepatan RSA dalam proses *signing* dan *verify*

***Key Length* (*bits*)**	**512-*bit***	**1024-*bit***	**2048-*bit***	**4096-*bit***
Sign per key size (sec)	0.00006	0.000205	0.001508	0.010717
Verify per key size (sec)	0.000005	0.000013	0.000045	0.000175

Tabel 2. Kecepatan ECDSA dalam proses *signing* dan *verify*

***Key Length* (*bits*)**	**160-*bit***	**192-*bit***	**224-*bit***	**256-*bit***	**384-*bit***
Sign per key size (*sec*)	0.0003	0.0003	0.0007	0.0009	0.0016
Verify per key size (*sec*)	0.0015	0.002	0.0024	0.0039	0.0082

Lalu untuk kecepatan pembuatan *key* dapat dilihat pada Tabel 3 dan Tabel 4. Dapat dilihat bahwa ECDSA lebih cepat dalam menghasilkan *key* daripada RSA.

Tabel 3. Kecepatan RSA dalam menghasilkan *key*

Key Length (bits)	**512-*bit***	**1024-*bit***	**2048-*bit***	**4096-*bit***
Key generation (sec)	0.132	0.431	2.559	112.3

Tabel 4. Kecepatan ECDSA dalam menghasilkan *key*

Key Length (bits)	**160-*bit***	**192-*bit***	**224-*bit***	**256-*bit***	**384-*bit***	**521-*bit***
Key generation (sec)	0.161	0.165	0.229	0.31	0.799	1.584

Sedangkan untuk *bulk encryption algorithm* akan mengutamakan penggunaan AES dikarenakan AES adalah standar enkripsi yang di rekomendasikan oleh *National Institute of Standards and Technology* (NIST) dengan mode kerja *Gallois Counter Mode* (GCM) daripada *Cipher Block Chaining* (CBC). Penggunaan CBC mode harus dihindari karena mempunyai masalah keamanan yaitu *padding oracle* [4]. Setelah AES diikuti oleh ARIA dan CHACHA20 dengan POLY1305 *authenticator*.

Selanjutnya untuk *Message Authentication Code algorithm* (MAC) yang digunakan lebih mengutamakan penggunaan *Secure Hash Algorithm 256* (SHA256) atau *Secure Hash Algorithm 384* (SHA384) dikarenakan SHA merupakan salah satu *Message Authentication Code* yang telah disarankan oleh NIST agar dipakai sebagai MAC [7]. Tabel 5 merupakan *cipher suite TLSv1.2* yang diajukan untuk diimplementasikan pada *web server* Apache dan Nginx.

Tabel 5. Algoritma *Cipher Suite* TLS 1.2 yang digunakan

OpenSSL Name	***IANA Name***	***Hexadecimal Code***
ECDHE-ECDSA-AES256-GCM-SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	(0xc02c)
ECDHE-ECDSA-ARIA256-GCM-SHA384	TLS_ECDHE_ECDSA_WITH_ARIA_256_GCM_SHA384	(0xc05d)
ECDHE-ECDSA-AES128-GCM-SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	(0xc02b)
ECDHE-ECDSA-ARIA128-GCM-SHA256	TLS_ECDHE_ECDSA_WITH_ARIA_128_GCM_SHA256	(0xc05c)
ECDHE-ECDSA-CHACHA20-POLY1305	TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256	(0xcca9)
ECDHE-RSA-AES256-GCM-SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	(0xc030)
ECDHE-ARIA256-GCM-SHA384	TLS_ECDHE_RSA_WITH_ARIA_256_GCM_SHA384	(0xc061)
ECDHE-RSA-CHACHA20-POLY1305	TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256	(0xcca8)
ECDHE-ARIA128-GCM-SHA256	TLS_ECDHE_RSA_WITH_ARIA_128_GCM_SHA256	(0xc060)
ECDHE-RSA-AES128-GCM-SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	(0xc02f)
ECDHE-RSA-AES256-SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	(0xc028)

Cipher suite yang digunakan oleh *TLSv1.3* tidak membutuhkan *key exchange algorithm* dan *authentication algorithm*, dikarenakan *TLSv1.3* pada dasarnya langsung menggunakan *Ephemeral Diffie-Hellman key exchange* [11]. Pada Tabel 6 menunjukkan *cipher suite* yang diajukan untuk diimplementasikan pada *web server* Apache dan Nginx.

Tabel 6. Algoritma *Cipher Suite* TLS 1.3 yang digunakan

OpenSSL	Hexdecimal Code
TLS_AES_128_GCM_SHA256	(0x1301)
TLS_AES_256_GCM_SHA384	(0x1302)
TLS_CHACHA20_POLY1305_SHA256	(0x1303)

Selain menentukan urutan algoritma *cipher suite*, mengaktifkan "*SSLHonorCipherOrder"* untuk Apache dan "*ssl_prefer_server_ciphers"* untuk Nginx merupakan hal yang penting. Dengan menggunakan parameter tersebut, *web server* akan memilih *cipher suite* yang paling depan daripada yang lain pada *list*. Tidak cukup hanya menggunakan *cipher preferred order,* menonaktifkan parameter "*SSLCompression"* pada Apache *web server* merupakan hal yang krusial. Namun, untuk Nginx *web server "SSLCompression"* sudah nonaktif sehingga tidak perlu melakukan konfigurasi lebih lanjut. Dengan menonaktifkan "*SSLCompression" web server* dapat terlindungi dari serangan *Compression Ration Info-leak Made Easy* (CRIME). CRIME merupakan serangan yang memanfaatkan *Transport Layer Security* (TLS) dan *Speedy* (SPDY) *protocol* yang juga menggunakan kompresi data untuk mendapatkan *key* agar penyerang dapat melakukan *session hijacking* [12]. Selain itu menonaktifkan "*SSLSessionTickets"* untuk Apache *web server* dan "*ssl_session_tickets"* untuk Nginx *web server* juga dapat meningkatkan keamanan suatu *web server*, karena dengan "*SSLSessionTickets"* yang masih aktif dapat mempengaruhi integritas *Perfect Forward Secrecy*.

3.2.2 *OCSP Stapling*

OCSP Stapling membuat kinerja *web server* lebih efisien dikarenakan *web server* menyimpan hasil *response* dari *OCSP Responder* sehingga *web server* tidak perlu melakukan *request* terhadap *OCSP Responder* setiap kali *client browser* terhubung. Mengaktifkan *OCSP Stapling* pada Apache *web server* dapat menambahkan nilai "*on"* pada parameter "*SSLUseStapling"* dan menambahkan nilai "*shmcb:logs/ssl_stapling(32768)*" pada parameter "*SSLStaplingCache"* di *website configuration block*. Sedangkan untuk Nginx *web server* bisa menambahkan nilai "*on*" pada parameter "*ssl_stapling"* dan menambahkan nilai "*on"* pada parameter "*ssl_stapling_verify"* di website *configuration block*.

3.2.3 *Secure HTTP Headers*

Melakukan konfigurasi pada *HTTP Response Header* merupakan hal yang sangat penting. Dengan menerapkan *Secure HTTP Header*, *web server* dapat terlindungi dari serangan seperti *clickjacking, Cross-Site Scripting* (XSS) *dan Multipurpose Internet Mail Extension* attack (MIME). Agar dapat menggunakan *header* yang diajukan, maka *module rewrite* dan *headers* harus diaktifkan. Untuk Apache *web server* dapat menggunakan *command* "*a2enmod rewrite*" dan "*a2enmod headers"*, sedangkan untuk Nginx *web server* dapat langsung mendeklarasikan *header* yang ingin digunakan.

1. HTTP Strict Transport Security
   Konfigurasi pada HSTS akan menggunakan parameter "*max-age"* dengan nilai minimal "31536000" detik, parameter "*includeSubDomains*" digunakan agar peraturan HSTS diterapkan pada seluruh *subdomain* dan parameter "*preload"* digunakan agar *web browser* seperti *Chrome*, *Internet Explorer* dan *Mozilla* menyantumkan *web server* pada *list* HSTS.

2. X-XSS-Protection
   Konfigurasi *X-XSS-Protection* yang digunakan adalah parameter "*1;mode=block"*. Dengan mendeklarasikan "*1;mode=block"*, *web browser* mengaktifkan *filter* dan jika pada suatu halaman ditemukan adanya serangan berupa XSS, maka parameter tersebut memerintahkan *browser* untuk tidak memuat *page* tersebut.
3. X-Frame-Options
   X-Frame-Options menerima 3 nilai parameter yaitu "*SAMEORIGIN", "DENY" dan "ALLOW-FROM".* Parameter *X-Frame-Options* yang digunakan adalah parameter "*sameorigin".*
4. X-Content-Type-Options
   Parameter yang digunakan pada *X-Content-Type-Options header* ini adalah "*nosniff"*. Dengan menggunakan parameter "*nosniff"*, *web browser* dapat mencegah *client browser* untuk melakukan *sniffing* pada asset yang sudah ditentukan oleh *web server.*
5. Content Security Policy
   Content Security Policy header yang digunakan adalah *script-src* dengan *unsafe-eval* dan *unsafe-inline*, *style-src*, *img-src* dan *font-src.*

3.2.4 *DH Parameter dan ECDH Parameter*

Dengan melakukan implementasi parameter *Diffie-Hellman* dengan ukuran *key* yang besar dapat mencegah serangan *logjam*. DH parameter yang akan digunakan mempunyai ukuran *bit* sebesar 4096 [3]. Lalu, untuk *Elliptic Curve Diffie Hellman*, *Curves* yang akan digunakan adalah 384-*bit curves* atau 521-*bit curve*.

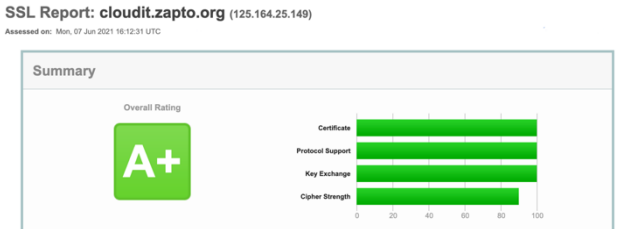
4. HASIL PENGUJIAN

Pengujian ini dilakukan dengan tujuan mengetahui tingkat keamanan konfigurasi SSL yang diajukan. Selain itu, pengujian juga dilakukan kepada konfigurasi SSL yang digunakan oleh beberapa web server serta memberikan saran sesuai konfigurasi SSL yang telah diajukan. Pengujian konfigurasi SSL menggunakan program bernama SSLyze dan Qualys SSL Labs.

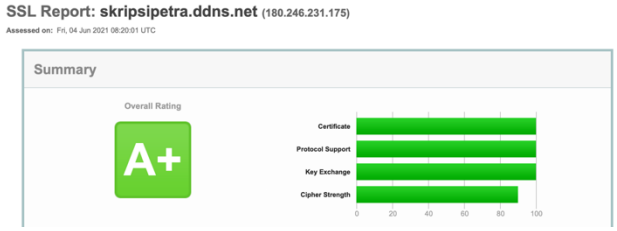
Pengujian konfigurasi SSL dilakukan dengan perintah *sslyze --regular --http_headers* agar SSLyze dapat melakukan scanning pada konfigurasi SSL.

Tabel 7. Hasil pengujian SSLyze terhadap konfigurasi SSL yang diajukan

	Apache	**Nginx**
Certificate Algorithm	Elliptic Curve , RSA	
SSL 2.0	FALSE	
SSL 3.0	FALSE	
TLS 1.0	FALSE	
TLS 1.1	FALSE	
TLS 1.2	TRUE	
TLS 1.3	TRUE	
Forward Secrecy	Supported	
RC4	Not Supported	
OCSP Stapling dan OCSP Must Staple	Yes	
HeartBleed	Not Vulnerable	
Session Renegotiation	Secure Renegotiation Supported	
Compression	Disabled	
Downgrade Attack	Not Vulnerable	
ROBOT	Not Vulnerable	
EC Key Exchange	secp384r1, secp521r1	
Session Resumption	Session ID Supported	
Secure HTTP Headers	Yes	



Gambar 2. Hasil pengujian konfigurasi SSL yang diajukan pada Apache menggunakan Qualys SSL Labs



Gambar 3. Hasil pengujian konfigurasi SSL yang diajukan pada Nginx menggunakan Qualys SSL Labs

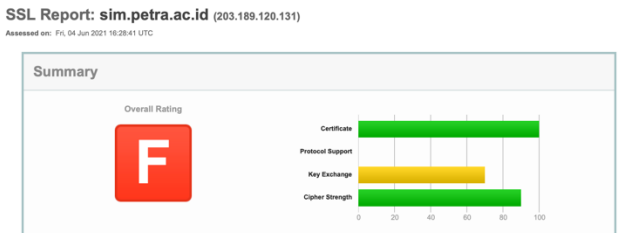
Gambar 2 dan Gambar 3 merupakan hasil pengujian konfigurasi SSL yang diajukan menggunakan Qualys SSL Labs sedangkan untuk Tabel 7 merupakan pengujian menggunakan SSLyze. Hasil tersebut menunjukkan bahwa konfigurasi yang diajukan mempunyai tingkat keamanan yang baik. Pada Tabel 7 menjelaskan bahwa konfigurasi SSL yang diajukan tidak lagi support TLS 1.0 dan TLS 1.1 serta memberikan support terhadap TLS 1.3, menggunakan OCSP Stapling dan Secure HTTP Headers, serta tidak rentan terhadap serangan yang tidak diinginkan.

Selain pengujian dilakukan pada konfigurasi SSL yang diajukan, pengujian juga akan dilakukan pada konfigurasi SSL milik website yang mempunyai nama domain "*sim.petra.ac.id*". Pengujian konfigurasi SSL dilakukan dengan perintah dan flag yang sama dengan pengujian pertama.

Tabel 8. Hasil pengujian SSLyze terhadap konfigurasi SSL "*sim.petra.ac.id*"

	Konfigurasi SSL "*sim.petra.ac.id*"
Certificate Algorithm	RSA
SSL 2.0	FALSE

SSL 3.0	TRUE
TLS 1.0	TRUE
TLS 1.1	TRUE
TLS 1.2	TRUE
TLS 1.3	FALSE
Forward Secrecy	Supported
RC4	Insecure, Supported
OpensSSL CCS Injection	Vulnerable
OCSP Stapling dan OCSP Must Staple	No
HeartBleed	Not Vulnerable
Session Renegotiation	Secure Renegotiation Supported
Compression	Vulnerable
Downgrade Attack	Vulnerable
ROBOT	Not Vulnerable
EC Key Exchange	prime256v1
Session Resumption	Session ID dan TLS Tickets Supported
Secure HTTP Headers	No



Gambar 4. Hasil pengujian konfigurasi SSL "*sim.petra.ac.id*" menggunakan Qualys SSL Labs

Hasil pengujian kedua menggunakan konfigurasi SSL dari "*sim.petra.ac.id*". Tabel 8 merupakan hasil pengujian menggunakan. Sedangkan untuk Gambar 4 merupakan hasil pengujian menggunakan Qualys SSL Labs. Pada Tabel 8 dan Gambar 4 menunjukan bahwa konfigurasi SSL yang dipakai oleh "*sim.petra.ac.id*" memiliki masalah keamanan yang cukup serius. Masalah tersebut dapat dijabarkan sebagai berikut:

1. Web server support SSL 3.0, TLS 1.0, TLS1.1 dan tidak support TLS 1.3 [10].
2. Penggunaan RC4 dan 3DES sebagai algoritma *encryption* [8].
3. Rentan terhadap serangan OpenSSL CCS, CRIME dan Protocol Downgrade
4. Penggunaan Diffie-Hellman key exchange kurang dari 4096-*bit*.
5. Penggunaan Cipher Block Chaining dengan key exchange dan bulk encryption yang lemah [4].
6. Tidak mengimplementasikan *Secure HTTP Headers* [9].
7. Tidak mengimplementasikan *OCSP Stapling* [5].

Dengan masalah keamanan tersebut, maka "*sim.petra.ac.id*" dianjurkan untuk mengubah konfigurasi SSL agar dapat meningkatkan keamanan dan kecepatan web server dengan menerapkan konfigurasi SSL sesuai dengan yang diajukan pada Tabel 9.

Tabel 9. Konfigurasi SSL yang diajukan untuk "*sim.petra.ac.id*"

Certificate Algorithm	*Elliptic Curve* dan *RSA*
TLS 1.0 (*Optional*)	ECDHE-ECDSA-AES256-SHA ECDHE-ECDSA-AES128-SHA ECDHE-RSA-AES256-SHA ECDHE-RSA-AES128-SHA DHE-RSA-AES256-SHA
TLS 1.1 (*Optional*)	
TLS 1.2	ECDHE-ECDSA- CHACHA20-POLY1305 ECDHE-ECDSA-AES256-GCM-SHA384 ECDHE-ECDSA-AES128-GCM-SHA256 ECDHE-ECDSA-ARIA256-GCM-SHA384 ECDHE-ECDSA-ARIA128-GCM-SHA256 ECDHE-RSA-CHACHA20-POLY1305 ECDHE-RSA-ARIA256-GCM-SHA384 ECDHE-RSA-ARIA128-GCM-SHA256 ECDHE-RSA- AES256-GCM-SHA384 ECDHE-RSA-AES256-SHA384 ECDHE-RSA-AES128-GCM-SHA256
TLS 1.3	TLS_CHACHA20_POLY1305_SHA256 TLS_AES_256_GCM_SHA384 TLS_AES_128_GCM_SHA256
Elliptic Curve Key Exchange	secp384r1, secp521r1
Strict-Transport-Security Header	*Max Age* : 31536000 *Include Subdomains : True* *Preload : True*
X-XSS-Protection	*1; mode=block*
X-Frame-Options	*SAMEORIGIN*
X-Content-Type-Options	*nosniff*

Sesuai saran yang diberikan, dapat dilihat bahwa versi *protocol* TLS yang digunakan lebih mengutamakan TLS 1.2 dan TLS 1.3. Sedangkan untuk penggunaan *protocol* TLS 1.0 dan TLS 1.1 merupakan hal yang opsional. Jika ingin menggunakan *protocol* TLS 1.0 dan TLS 1.1 maka *cipher suite* yang digunakan harus sesuai dengan Tabel 9. Untuk *protocol* TLS 1.2 dan TLS 1.3 menggunakan algoritma *cipher suite* yang sesuai dengan konfigurasi SSL yang diajukan pada Tabel 5 dan Tabel 6. Selain menggunakan *cipher suite* yang tepat, menggunakan DH parameter 4096-*bit* dan ECDH *curve secp384r1* atau *secp521r1* dapat membantu meningkatkan keamanan *web server*. Agar *web server* dapat bekerja lebih optimal dan aman maka penggunaan *OCSP Stapling, OCSP Must Staple* dan *Secure HTTP Headers* merupakan hal yang dianjurkan.

5. KESIMPULAN

Berdasarkan pengujian yang telah dilakukan, dapat dilihat bahwa masih terdapat banyak web server menggunakan konfigurasi SSL dengan tingkat keamanan yang kurang baik. Dengan menggunakan konfigurasi SSL yang telah disarankan maka web server akan terjaga dari masalah keamanan yang ada. Keunggulan konfigurasi SSL ini yaitu mempunyai tingkat keamanan yang lebih baik serta membuat kinerja server lebih efisien dan optimal. Namun, kelemahan konfigurasi SSL yang disarankan terletak pada support terhadap beberapa legacy browser, dimana pada umunya legacy browser menggunakan TLS 1.0 dan sebelumnya untuk melakukan koneksi HTTPS. Kemudian, saran untuk penelitian ke depan yakni seiring dengan berkembangnya zaman, sudah dapat dipastikan bahwa protokol keamanan web akan selalu berkembang sehingga diperlukan penelitian lebih lanjut terhadap protokol di masa yang akan datang.

6. DAFTAR PUSTAKA

[1] Ahmed Ansari, J. and Najera-Gutierrez, G. Web Penetration Testing with Kali Linux - Third Edition | Packt. *Packt*, 2018. https://www.packtpub.com/product/web-penetration-testing-with-kali-linux-third-edition/9781788623377.

[2] Ali, A. Comparison and Evaluation of Digital Signature Schemes Employed in NDN Network. *International Journal of Embedded Systems and Applications 5*, 2 (2015), 15-29.

[3] Brehm, T. How to protect your Debian or Ubuntu Server against the Logjam attack. *HowtoForge*. https://www.howtoforge.com/tutorial/how-to-protect-your-debian-and-ubuntu-server-against-the-logjam-attack/.

[4] Coders Conquer Security: Share & Learn Series - Padding Oracle. *Securecodewarrior.com*, 2019. https://www.securecodewarrior.com/blog/coders-conquer-security-share-learn-series-padding-oracle.

[5] Crane, C. Everything You Need to Know About OCSP, OCSP Stapling & OCSP Must-Staple - Hashed Out by The SSL Store™. *Hashed Out by The SSL Store™*, 2020. https://www.thesslstore.com/blog/ocsp-ocsp-stapling-ocsp-must-staple/.

[6] Greenberg, A. Hacker Lexicon: What Is Perfect Forward Secrecy?. *Wired*, 2016. https://www.wired.com/2016/11/what-is-perfect-forward-secrecy/.

[7] Hagenlocher, P. Performance of Message Authentication Codes for Secure Ethernet. *net.in.tum.de*, 2018. https://www.net.in.tum.de/fileadmin/TUM/NET/NET-2018-11-1/NET-2018-11-1_04.pdf.

[8] Kaduk, B. and Short, M. Deprecate 3DES and RC4 in Kerberos. *Tools.ietf.org*, 2017. https://tools.ietf.org/id/draft-ietf-curdle-des-des-des-die-die-die-01.html.

[9] Kumar, R. 12 security headers you should use to prevent Vulnerabilities. *Rajesh Kumar*, 2019. https://www.rsupernova.com/12-security-headers-you-should-use-to-prevent-vulnerabilities/.

[10] Moriarty, K. and Farell, S. Deprecating TLSv1.0 and TLSv1.1. *Tools.ietf.org*, 2020. https://tools.ietf.org/id/draft-ietf-tls-oldversions-deprecate-06.html.

[11] Nohe, P. TLS 1.3 Update: Everything you need to know. *Hashed Out by The SSL Store™*, 2019. https://www.thesslstore.com/blog/tls-1-3-everything-possibly-needed-know.

[12] Satapathy, A. and Livingston, J. A Comprehensive Survey on SSL/ TLS and their Vulnerabilities. *International Journal of Computer Applications 153*, 5 (2016), 31-38.