Hardening Sistem Informasi XYZ Menggunakan *Framework* OWASP

Fadila Ahmad Sya'bani[1], Fayruz Rahma[2]
[1,2]*Jurusan Informatika*
Fakultas Teknologi Industri, Universitas Islam Indonesia
Yogyakarta, Indonesia
[1]18523107@students.uii.ac.id, [2]fayruz.rahma@uii.ac.id

***Abstrak*— Aplikasi berbasis web merupakan platform yang telah digunakan secara luas di berbagai bidang. Salah satu implementasi dari aplikasi berbasis web adalah sistem informasi akademik. Namun, dengan banyaknya penggunaan web pengelolaan informasi, terdapat sebagian orang yang memanfaatkan celah keamanan untuk aktivitas ilegal. Untuk menghindari hal tersebut terjadi, diperlukan suatu upaya pengerasan atau penguatan keamanan agar keamanan aplikasi web dan *server* lebih baik. Makalah ini memaparkan proses penelitian mengenai *hardening* aplikasi berbasis web menggunakan metode dari *Open Web Application Security Project* (OWASP). Hal tersebut meliputi proses *hardening* dengan pembuatan *security checklist* dari OWASP Top 10 2021, OWASP WSTG – *Stable*: *Phase 3 During Development*, dan implementasi pengerasan keamanan pada sisi aplikasi dan *server* yang digunakan. Hasil pada penelitian ini menunjukkan bahwa terdapat beberapa celah keamanan yang ditemukan seperti SQL *Injection*, XSS, *vulnerable package*, dan *misconfiguration* yang kemudian berhasil dimitigasi. Proses pengerasan keamanan dilakukan dengan menambahkan fungsi dan melakukan pembaruan *package*/*libary* yang digunakan oleh aplikasi. Aplikasi berbasis web yang telah melalui tahap *hardening* ini lalu dilakukan pengujian dan memberikan hasil bahwa proses uji keamanan telah berhasil.**

Keywords— Hardening, OWASP, Security, Vulnerability, Website

I. PENDAHULUAN

Program Studi ABC sedang mengembangkan sistem informasi akademik XYZ berbasis web yang akan digunakan sebagai sarana penunjang kegiatan kuliah. Penggunaan web yang semakin luas mengakibatkan aplikasi berbasis web menjadi salah satu sasaran serangan kriminal siber. Salah satu penyebab terjadinya serangan adalah adanya celah keamanan pada *source code* atau konfigurasi yang digunakan. Kerentanan tersebut dapat dieksploitasi dengan cara melakukan injeksi kode program, pelanggaran *business logic*, dan memanfaatkan celah manajemen sesi aplikasi [1]. Dampak yang dapat terjadi adalah kebocoran data, pencurian identitas, penipuan, dan lain-lain. Maka dari itu, diperlukan suatu upaya untuk mencegah celah keamanan tersebut terjadi.

Hardening keamanan adalah serangkaian proses mengamankan sistem dengan *patching* celah keamanan dan menonaktifkan layanan yang tidak diperlukan [2]. Proses *hardening* diperlukan oleh Sistem Informasi XYZ yang sedang dikembangkan karena cara tersebut dapat mengurangi *attack-surface* yang dapat dieksploitasi oleh penyerang.

Open Web Application Security Project (OWASP) merupakan suatu badan nonprofit yang bertujuan untuk membantu meningkatkan keamanan sistem informasi berbasis web. Proyek, *tools,* dan dokumentasi *open-source* yang telah dipublikasikan oleh OWASP telah digunakan oleh pengembang perangkat lunak dan para ahli keamanan sebagai rujukan dan platform dalam mengamankan sistem [3].

The Web Security Testing Guide atau WSTG versi stabil adalah sebuah panduan komprehensif pengujian keamanan perangkat lunak berbasis web yang merupakan salah satu proyek oleh OWASP. Kerangka kerja ini berisi tentang tahapan-tahapan pengujian keamanan pada setiap *Software Development Life Cycle* (SDLC) yang sedang dilakukan. WSTG telah tersedia dalam beberapa versi, salah satu versi yang telah dirilis adalah versi stabil. Dengan sistem informasi XYZ berbasis web yang sedang dikembangkan, fase WSTG yang sesuai dan akan digunakan adalah fase *During Development* [4].

Proyek lain yang telah dipublikasikan oleh *Open Web Application Security Project* adalah OWASP Top 10. Publikasi ini merupakan *awareness-document* untuk pengembang dan pemerhati keamanan aplikasi web. OWASP Top 10 berisi tentang sepuluh celah keamanan kritis paling populer yang dapat mengancam aplikasi berbasis web. Dokumen ini diterbitkan dalam rentang empat tahun sekali dengan rilisan 2021 sebagai publikasi paling baru [5].

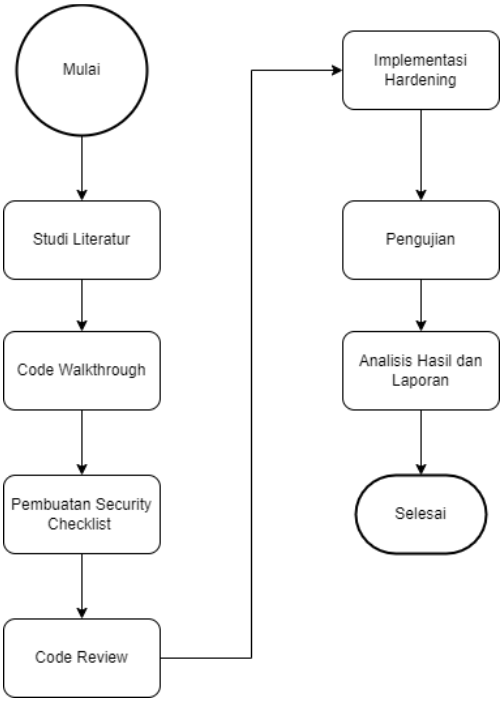
Penilitian ilmiah mengenai keamanan siber dengan topik aplikasi web telah cukup banyak dilakukan. Secara umum, pengamanan sistem dapat ditangani dua bagian/tim, yaitu *Red Team* yang berfokus dalam penyerangan dan *Blue Team* yang berfokus pada pertahanan [6]. Proses penelitian ini akan menggunakan dua pendekatan tersebut, yakni *Blue Team* pada saat pengerasan keamanan dan *Red Team* pada tahap pengujian. Maka dari itu, referensi tambahan diperlukan sebagai dasar dalam penelitian.

II. METODE

Penelitian *Hardening* Sistem web dilakukan dengan melalui beberapa tahap sistematis. Secara garis besar, alur penelitian dapat dilihat dalam diagram alur pada Gambar 1.

A. Studi Literatur

Pada tahap ini, studi literatur dilakukan dengan tujuan untuk mencari dan mengumpulkan makalah-makalah ilmiah yang akan digunakan untuk mendukung penelitian *hardening* web. Selanjutnya, dilakukan penyeleksian informasi atas literatur ilmiah yang dikumpulkan berdasarkan beberapa kategori yang telah ditentukan. Hasil akhir dari tahap ini berupa referensi penelitian-penelitian ilmiah.



Gambar 1 Diagram Alur Penelitian

B. Code Walkthrough

Penulis bersama dengan tim pengembang melakukan *code walkthrough* dengan melihat kode secara umum di mana pengembang menjelaskan logika dan aliran kode yang diimplementasikan. Tujuan dari kegiatan ini adalah untuk mendapatkan pemahaman umum tentang kode, aliran, *layout*, dan struktur kode yang membentuk aplikasi [4]. *Code Walktrough* dilakukan dengan beberapa media antara lain: pertemuan langsung melalui platform daring maupun luring dan dapat melalui pesan teks.

C. Pembuatan Security Checklist

Daftar pemeriksaan celah keamanan dibuat berdasarkan OWASP Top 10 2021 yang berisi mengenai sepuluh kerentanan keamanan kritis populer yang dapat terjadi. C*hecklist* tersebut berisi mengenai deskripsi celah keamanan, cara mengatasi kerentanan, dan contoh serangan yang dapat terjadi.

D. Code Review

Dengan landasan pengetahuan mengenai struktur kode dan alasan di balik penggunaan kode tersebut, penulis dapat melakukan *code review* terhadap kode sumber yang sebenarnya untuk mencari celah keamanan. Pemeriksaan *source code* dilakukan dengan menerapkan *checklist* keamanan yang telah dibuat terhadap kode aplikasi dan *server* [4].

E. Implementasi Hardening

Pada tahap implementasi pengerasan keamanan, terdapat dua hal yang menjadi fokus utama, yaitu *hardening* pada sisi aplikasi web dan *server*. Pada aplikasi web sistem informasi XYZ, dilakukan tinjauan kode aplikasi untuk memastikan keamanan aplikasi pada bagian *frontend* maupun *backend*. Apabila terdapat kode yang rawan menimbulkan celah keamanan, akan dilakukan perbaikan kode.

Fokus lain implementasi *hardening* terdapat pada sisi *server*. Di sini akan dilakukan pemeriksaan keamanan sistem operasi dan jaringan. Apabila terdapat konfigurasi yang dijalankan pada sistem operasi dan jaringan yang digunakan dapat menimbulkan kerentanan keamanan, dilakukan perbaikan dengan mengganti konfigurasi tersebut dengan keamanan yang lebih baik.

Hal-hal teknis yang dilakukan selama proses implementasi dapat menggunakan panduan yang telah tersedia pada *security checklist* maupun informasi yang tersedia pada sumber resmi kerangka kerja, bahasa, ataupun perangkat lunak yang digunakan untuk membangun aplikasi.

F. Pengujian

Tahap terakhir yang dilakukan setelah implementasi *hardening* adalah melakukan pengujian. Tujuan dari dilakukannya pengujian keamanan adalah untuk mengetahui tingkat keberhasilan dari proses implementasi *hardening*. Dalam *cybersecurity*, pengujian keamanan dapat berupa kegiatan *penetration testing*, yaitu menguji kerentanan yang ada dengan mengeksploitasi celah keamanan tersebut. Penguji dapat mengeksploitasi kerentanan dengan melakukan serangan terhadap aplikasi dengan metode *scripting* ataupun menggunakan *tools* yang telah tersedia luas di internet. Hasil dari pengujian dapat digunakan sebagai acuan tingkat keamanan aplikasi.

III. HASIL DAN PEMBAHASAN

Pada bab ini, akan dijelaskan mengenai hasil yang diperoleh sesuai dengan metode yang digunakan dan penjelasan tentang hasil yang didapatkan.

A. Studi Literatur

a. Sumber Literatur

Tahap pengumpulan makalah penelitian ilmiah dilakukan dengan pencarian dari beberapa sumber di internet. Web yang digunakan antara lain: Google, Google Scholar, Research Gate, dan Science Direct.

b. Klasifikasi Literatur

Terdapat sepuluh literatur ilmiah dan akan dimasukkan dalam tiga kategori, yaitu: *Offensive*, *Defensive*, dan *Other*. Makalah akan dimasukkan dalam kategori *other* apabila tidak membahas mengenai serangan ataupun pengamanan sistem. Hasil klasifikasi literatur dapat dilihat pada Tabel 1.

Tabel 1. PENGGOLONGAN LITERATUR ILMIAH

Offensive	*Defensive*	*Other*
[7], [8], [9], [10]	[11], [2], [12], [13], [14]	[15]

B. Code Walkthrough.

Aplikasi web versi terbaru saat ini dalam masa pengembangan dan akan memasuki fase *deployment* dalam waktu dekat. Kegiatan yang dilakukan berupa penambahan fitur, pengujian fitur, perbaikan kode, dan melakukan konfigurasi *server*.

Berdasarkan aktivitas *code walkthrough*, dapat diperoleh informasi mengenai web aplikasi yang dibuat dan *server* yang digunakan. Rincian informasi tersebut:

a. Aplikasi *backend*:
   - Aplikasi bernama XYZ-API
   - Bahasa utama yang digunakan adalah Go.
   - *Native*. Aplikasi tidak dibangun menggunakan *framework* seperti GORM, Laravel, Codeigniter, dan lain-lain.
   - Paradigma aplikasi menggunakan *Model, View, Controller* (MVC).
   - Menggunakan JSON Web Token (JWT) untuk autentikasi.

b. Aplikasi *frontend:*
   - Aplikasi bernama XYZ-FE
   - Bahasa utama yang digunakan adalah Javascript.
   - Menggunakan NodeJS sebagai Javascript *runtime*.
   - *Node Package Manager* (NPM) dan Yarn sebagai manajemen paket.
   - Menggunakan kerangka kerja NextJS.

c. *Server*
   - OS: Ubuntu 18.04.6 LTS (Bionic Beaver)
   - RAM: 1.8GB
   - Swap: 2GB
   - Processor: Intel(R) Xeon(R)
   - CPU *core*: 1
   - Web *server*: Nginx
   - DBMS: Postgresql
   - Menggunakan Pm2 sebagai *process manager* untuk NodeJS.

C. *Pembuatan Security Checklist*

Pada tahap pembuatan *security checklist*, OWASP Top 10 2021 digunakan sebagai acuan. Tabel 2 merupakan konten dari OWASP Top 10 2021.

Tabel 2. OWASP TOP 10 2021

Rank	***Vulnerability***
1	*Broken Access Control*
2	*Cryptographic Failures*
3	*Injection*
4	*Insecure Design*
5	*Security Misconfiguration*
6	*Vulnerable and Outdated Components*
7	*Identification and Authentication Failures*
8	*Software and Data Integrity Failures*
9	*Security Logging and Monitoring Failures*
10	*Server-Side Request Forgery* (SSRF)

Pada masing-masing penjelasan celah keamanan yang dipaparkan OWASP, terdapat tujuh subkonten yang berisi tentang:

1) *Factors*
2) *Overview*
3) *Description*
4) *How to Prevent*
5) *Example Attack Scenarios*
6) *References*
7) *List of Mapped CWEs*

Dari tujuh subkonten yang tersedia, penulis memilih tiga subkonten yang akan dijadikan sebagai *security checklist* dan akan digunakan pada *code review*. Subkonten *Description* dipilih karena bagian tersebut menyediakan informasi tentang deskripsi celah keamanan dan penyebab umum terjadinya kerentanan. Kedua, pemilihan *How to Prevent* didasarkan pada panduan teknis yang dapat membantu dalam tahap implementasi *hardening*. Poin *Example Attack Scenarios* dipilih karena subkonten tersebut memberikan informasi mengenai metode yang dapat digunakan oleh penyerang untuk mengeksploitasi celah keamanan yang bersangkutan.

Selain informasi yang diberikan oleh OWASP Top 10 2021, penulis juga menambahkan keterangan tambahan berupa catatan. Hal tersebut digunakan sebagai rujukan dalam melakukan proses pengerasan keamanan. Informasi tambahan tersebut berasal dari berbagai sumber internet baik resmi maupun nonresmi. Pada bagian ini juga terdapat hasil dari *Code Review* yang berisi mengenai celah-celah keamanan yang telah ditemukan.

D. *Code Review*

a. XYZ-API

- *Injection*

  XYZ-API telah menerapkan penggunaan parameter dalam setiap eksekusi fungsi SQL [16]. Pada saat sebuah *statement* sql dijalankan, sql *package* mengubah sql *statement* menjadi *prepared statement* atau *statement* yang ber-parameter lalu dikirimkan bersamaan dengan parameter secara terpisah. Pada postgresql digunakan tanda dollar ($) sebagai parameter, seperti yang ditunjukkan pada Gambar 2.

```
SQL := `INSERT INTO
research_students(
    student_id, title, category_id
)
VALUES ($1, $2, $3)
RETURNING id`
```

Gambar 2 *SQL Prepared Statement*

- *Vulnerable and Outdated Components*

  Untuk mengecek ketersediaan pembaruan pada dependensi yang digunakan pada *Application Programming Interface (API),* dapat digunakan perintah terminal:

```
go list -u -f '{{if (and (not
(or .Main .Indirect))
.Update)}}{{.Path}}:
{{.Version}} ->
{{.Update.Version}}{{end}}' -
m all 2> /dev/null
```

Perintah *go list* akan menampilkan daftar dependensi yang digunakan. *Flag* -u digunakan untuk menampilkan dependensi yang telah tersedia pembaharuan. *Flag* -f digunakan untuk melakukan format pada *output* yang dikeluarkan. *Flag* -m menyebabkan Go melakukan *list* terhadap modul dibandingkan dengan *package* [17]. Hasil pada Gambar 3 memperlihatkan terdapat tiga dependensi yang dapat diperbarui.

```
$ go list -u -f '{{if (and (not (or .Main .Indirect)) .Updat
h}}: {{.Version}} -> {{.Update.Version}}{{end}}' -m all 2> /
github.com/gin-gonic/gin: v1.7.4 -> v1.8.0
github.com/go-playground/validator/v10: v10.9.0 -> v10.11.0
github.com/lib/pq: v1.10.3 -> v1.10.6
```

Gambar 3 Hasil pengecekan *outdated package*

- *Software and Data Integrity Failures*
  Setiap proses impor modul, Go akan melakukan proses *hash* terhadap modul yang bersangkutan. Hasil dari *hash* tersebut akan disimpan dalam dokumen go.sum seperti yang ditunjukkan pada Gambar 4.

```
github.com/aymerick/douceur v0.2.0 h1:Mv+mAeH1Q+n9Fr+oyamOlAkUNPWPlA8PPGR0QAaYuPk=
github.com/aymerick/douceur v0.2.0/go.mod h1:wlT5vV2O3h55X9m7iVYN0TBM0NH/MmbLnd30/FjWUq4=
github.com/creack/pty v1.1.9/go.mod h1:oKZEueFk5CKHvIhNR5MUki03XCEU+Q6VDXinZuGJ33E=
github.com/davecgh/go-spew v1.1.0/go.mod h1:J7Y8YcW2NihsgmVo/mv3lAwl/skON4iLHjSsI+c5H38=
github.com/davecgh/go-spew v1.1.1 h1:vj9j/u1bqnvCEfJOwUhtlOARqs3+rkHYY13jYWTU97c=
github.com/davecgh/go-spew v1.1.1/go.mod h1:J7Y8YcW2NihsgmVo/mv3lAwl/skON4iLHjSsI+c5H38=
github.com/gin-contrib/sse v0.1.0 h1:Y/yl/+YNO8GZSjAhjMsSuLt29uWRFHdHYUb5lYOV9qE=
github.com/gin-contrib/sse v0.1.0/go.mod h1:RHrZQHXnP2xjPF+u1gW/2HnVO7nvIa9PG3Gm+fLHvGI=
github.com/gin-gonic/gin v1.7.4 h1:QmUZXrvJ9qZ3GfWvQ+2wnW/1ePrTEJqPKMYEU3lD/DM=
github.com/gin-gonic/gin v1.7.4/go.mod h1:jD2toBW3GZUr5UMcdrwQA10I7RuaFOl/SGeDjXkfUtY=
```

Gambar 4 Hash dari *package* yang digunakan

- *Security Logging and Monitoring Failures*
  Tim pengembang XYZ telah menambahkan fitur *logging* pada sisi API, seperti yang ditunjukkan pada Gambar 5. Fitur tersebut akan menuliskan aktivitas log seperti info, *warning*, dan eror. Hasil dari log akan dituliskan ke dalam berkas logs.log. seperti yang ditunjukkan pada Gambar 6.

```
package utils

import (
    "log"
    "os"
)

var (
    WarningLogger *log.Logger
    InfoLogger    *log.Logger
    ErrorLogger   *log.Logger
)

func NewLogger() {
    file, err := os.OpenFile("logs.log", os.O_APPEND|os.O_CREATE|os.O_WRONLY, 066
    PanicIfError(err)

    InfoLogger = log.New(file, "INFO: ", log.Ldate|log.Ltime|log.Lshortfile)
    WarningLogger = log.New(file, "WARNING: ", log.Ldate|log.Ltime|log.Lshortfil
    ErrorLogger = log.New(file, "ERROR: ", log.Ldate|log.Ltime|log.Lshortfile)
}
```

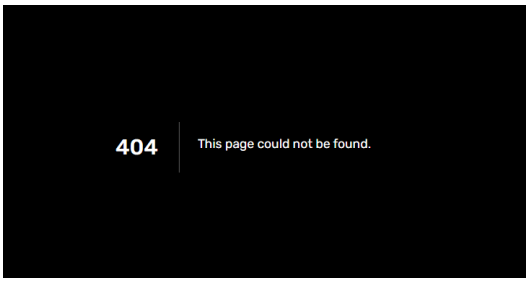
Gambar 5 Fungsi log

```
INFO: 2022/05/13 21:44:00 app.go:37: Env file successf
INFO: 2022/05/13 21:44:00 app.go:45: Database successf
INFO: 2022/05/13 21:44:00 app.go:47: Validator success
INFO: 2022/05/16 21:44:52 app.go:37: Env file successf
INFO: 2022/05/16 21:44:52 app.go:45: Database successf
INFO: 2022/05/16 21:44:52 app.go:47: Validator success
INFO: 2022/05/16 22:02:00 app.go:37: Env file successf
INFO: 2022/05/16 22:02:00 app.go:45: Database successf
INFO: 2022/05/16 22:02:00 app.go:47: Validator success
INFO: 2022/05/18 21:18:22 app.go:37: Env file successf
INFO: 2022/05/18 21:18:22 app.go:45: Database successf
INFO: 2022/05/18 21:18:22 app.go:47: Validator success
```

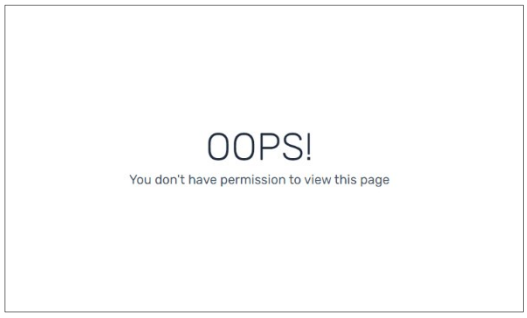
Gambar 6 Hasil fungsi log

b. XYZ-FE

- *Security Misconfiguration*
  XYZ-FE telah menerapkan *custom error handling*. Fungsi tersebut bertujuan untuk tidak memberikan pesan kesalahan yang terlalu informatif kepada *end-user*. Skenario yang dapat terjadi antara lain: halaman yang di-*request* tidak tersedia (Gambar 7) atau mengakses url dengan *role* yang berbeda. (Gambar 8).



Gambar 7 Halaman web tidak ditemukan



Gambar 8 Pesan *error* tidak mendapatkan izin

XYZ-FE belum menerapkan *http response header* tambahan, terutama yang berkaitan dengan keamanan. Konfigurasi *header* terdapat pada berkas next.config.js. yang ditampilkan pada Gambar 9.

```
JS next.config.js ×
JS next.config.js > ...
1  /** @type {import('next').NextConfig} */
2  module.exports = {
3    reactStrictMode: true,
4  }
```

Gambar 9 Berkas next.config.js

- *Vulnerable and Outdated Components*
  NPM telah mempunyai *built in* perintah yang dapat digunakan untuk melakukan audit

keamanan pada *packages* yang digunakan yaitu `npm audit`. Perintah tersebut akan mengirim-kan deskripsi dependensi ke *default registry* dan akan meminta laporan *known vulnerability*. Apabila ada kerentanan yang ditemukan, dampak kerentanan dan remediasi yang diperlukan akan dihitung [18].

Pada perintah `npm audit`, hasil yang didapatkan berupa tujuh celah keamanan dengan tiga tingkat bahaya: satu *vulnerability* berstatus *critical*, dua *vulnerability* dengan tingkat *high*, dan empat mempunyai tingkat *moderate*, seperti yang ditunjukkan pada Gambar 10 dan Gambar 11.

```
# npm audit report

minimist  <1.2.6
Severity: critical
Prototype Pollution in minimist - https://github.com/advisories/GHSA-xvch-5gv4-984h
fix available via `npm audit fix`
node_modules/minimist

nanoid  3.0.0 - 3.1.30
Severity: moderate
Exposure of Sensitive Information to an Unauthorized Actor in nanoid - https://github.com/
fix available via `npm audit fix`
node_modules/nanoid

next  9.0.6-canary.0 - 9.3.4-canary.0 || 10.0.0 - 12.0.11-canary.21
Severity: high
Improper CSP in Image Optimization API for Next.js versions between 10.0.0 and 12.1.0 - ht
Unexpected server crash in Next.js. - https://github.com/advisories/GHSA-25mp-g6fv-mqxx
```

Gambar 10 Hasil dari perintah npm audit

```
quill  <=1.3.7
Severity: moderate
Cross-site Scripting in quill - https://github.com/advisories/GHSA-4943-9vgg-gr5r
fix available via `npm audit fix --force`
Will install react-quill@0.0.2, which is a breaking change
node_modules/quill
  react-quill  >=0.0.3
  Depends on vulnerable versions of quill
  node_modules/react-quill

7 vulnerabilities (4 moderate, 2 high, 1 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force
```

Gambar 11 Hasil dari perintah npm audit (lanjutan)

- *Software and Data Integrity Failures*
  Untuk setiap *package* yang dipasang dan digunakan pada NextJS dengan manajer paket, npm akan mencatat nama, versi, *engine*, *resolved,* dependensi, dan *integrity* dari setiap *package* ke dalam berkas package-lock.json. Algoritma yang dipakai oleh npm untuk melakukan proses *hash* adalah SHA512, seperti yang ditunjukkan pada Gambar 12.

```
"node_modules/@babel/code-frame": {
  "version": "7.12.11",
  "resolved": "https://registry.npmjs.org/@babel/code-frame/-/code-frame-7.12.11.tgz",
  "integrity": "sha512-Zt1yodBx1UcyiePMSkWnU4hPqhwq7hGi2nFL1LeA3EUl+q2LQx16MISgJ0+z7dnmgv
  "dependencies": {
    "@babel/highlight": "^7.10.4"
  }
},
"node_modules/@babel/helper-validator-identifier": {
  "version": "7.16.7",
  "resolved": "https://registry.npmjs.org/@babel/helper-validator-identifier/-/helper-val
  "integrity": "sha512-hsEnFemeiW4D08A5gUAZxLBTXpZ39P+a+DGDsHw1yxqyQ/jzFEnxf5uTEGp+3bzAbH
  "engines": {
    "node": ">=6.9.0"
  }
},
```

Gambar 12 *Package* dan nilai *hash* pada file package-lock.js

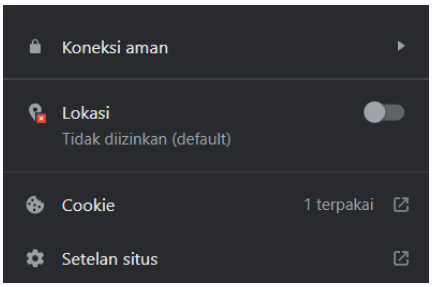
c. *Server*

a. Cryptographic Failures

Password yang digunakan oleh akun *server* XYZ telah di-*hash* menggunakan algoritma SHA512 dengan ditandai oleh 6. Hasil dari proses *hash* dapat dilihat dalam *file* /etc/shadow seperti yang ditunjukkan pada Gambar 13. Sertifikat SSL juga telah dipasang pada *server* XYZ dan telah digunakan pada saat situs tersebut diakses, seperti yang ditunjukkan pada Gambar 14.

```
informatics-sekawan:~$ sudo cat /etc/sha
$6$e.kKLfk6$FhBe8QGE8QZNLMupXeksptgMpfWm
```

Gambar 13 Nilai *hash* pada akun linux *server*



Gambar 14 Bukti penggunaan https pada XYZ

b. *Insecure Design*

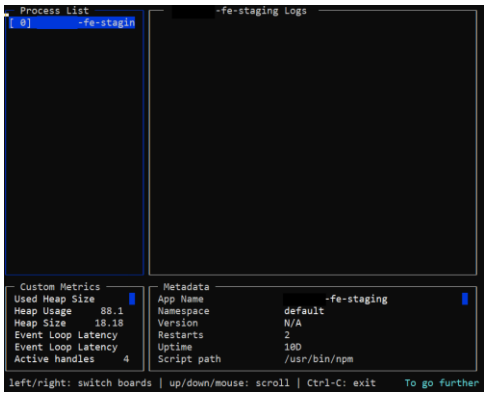
Salah satu komponen dari CIA Triad adalah *Availability.* Hal tersebut telah diimplementasikan dalam *server* XYZ dengan manajemen layanan. XYZ-API dijalankan pada *server* dengan membuat *systemd custom service* yang akan meng-*handle* aktivitas layanan, seperti yang ditunjukkan pada Gambar 15. Di sisi lain, XYZ-FE menggunakan *process manager* untuk NodeJS yaitu pm2, seperti yang ditunjukkan pada Gambar 16. Kedua hal tersebut diperlukan untuk mengelola dan menjaga aplikasi agar tetap *online*.

```
[Unit]
Description=sekawan-api
After=multi-user.target

[Service]
User=root
Group=root
ExecStart=/root/sekawan-api/sekawan-api

[Install]
WantedBy=multi-user.target
```

Gambar 15 Layanan XYZ-API pada systemd Linux



Gambar 16 Manajemen proses pm2

c. *Security Misconfiguration*

Akun yang digunakan pada *server* XYZ telah menerapkan *password* dan tidak menggunakan *password* bawaan/lemah dan telah diganti dengan kata sandi yang lebih kuat.

E. *Implementasi Hardening*

a. *XYZ-API*

- *Injection*

Salah satu celah keamanan injeksi adalah *Cross Site Scripting* (XSS). Celah tersebut memanfaatkan *form input* yang tidak divalidasi sehingga dapat digunakan untuk mengeksekusi *script* yang dijalankan [7].

```go
package utils

import (
    "github.com/microcosm-cc/bluemonday"
)

// This function will strip all html elements from user's i
func HTMLSanitize(input string) string {
    p := bluemonday.NewPolicy()

    // Allowed elements
    p.AllowElements("p")
    p.AllowElements("ol")
    p.AllowElements("li")

    sanitized := p.Sanitize(input)
    return sanitized
}
```

Gambar 17 Fungsi HTMLSanitizer

Pengerasan keamanan yang dilakukan adalah dengan melakukan sanitasi terhadap *input*. Proses sanitasi akan memodifikasi input yang diberikan dengan mengubah atau menghapus simbol atau karakter tertentu [11]. Apabila user memasukan *malicious tag* html, fungsi HTMLSanitizer (Gambar 17) akan menghapus *tag* html yang tidak sesuai dengan aturan yang telah ditetapkan pada kode.

```go
err := tx.QueryRow(SQL, researchStudent.StudentId,
    utils.HTMLSanitize(researchStudent.Title), researchStudent.Catego
utils.PanicIfError(err)
```

Gambar 18 Implementasi HTMLSanitizer pada eksekusi sql

Gambar 18 merupakan contoh penerapan fungsi HTMLSanitizer yang telah dibuat. Fungsi dimasukkan pada parameter yang terdapat di eksekusi sql dan akan melakukan sanitasi pada *argument* yang diberikan.

b. *XYZ-FE*

- *Security Misconfiguration*

Pengerasan keamanan dilakukan dengan menambahkan *custom http response header* yang berkaitan keamanan pada aplikasi *frontend* [19].

```js
module.exports = {
  reactStrictMode: true,
  async headers() {
    return [
      {
        // Apply these headers to all routes in your application.
        source: "/:path*",
        headers: securityHeaders,
      },
    ];
  },
};
```

Gambar 19 Implementasi custom http response header

Pada Gambar 19, headers() merupakan fungsi asinkron dengan menggunakan *array* yang menyimpan properti *source* dan *header* yang akan diimplementasi. *Headers* tersebut dapat dimasukkan ke dalam variabel bertipe *array* agar dapat menyimpan lebih dari satu *header*. Pada XYZ-FE, terdapat enam *security header* tambahan yang diimplementasi antara lain: *Strict-Transport-Security, X-XSS-Protection, X-Frame-Options, X-Content-Type-Options, Referrer-Policy,* dan *Content-Security-Policy.*

```js
const securityHeaders = [
  {
    key: "Strict-Transport-Security",
    value: "max-age=63072000; includeSubDomains; preload",
  },
  {
    key: "X-XSS-Protection",
    value: "1; mode=block",
  },
  {
    key: "X-Frame-Options",
    value: "SAMEORIGIN",
  },
  {
    key: "X-Content-Type-Options",
    value: "nosniff",
  },
  {
    key: "Referrer-Policy",
    value: "origin-when-cross-origin",
  },
  {
    key: "Content-Security-Policy",
    value: ContentSecurityPolicy.replace(/\s{2,}/g, " ").trim(),
  },
];
```

Gambar 20 *Custom http header* yang diterapkan

Pada *Content-Security-Policy* terdapat *directives* (Gambar 21) yang berfungsi untuk mendeskripsikan aturan-aturan yang diterapkan pada *header Content-Security-Policy* [20].

```
const ContentSecurityPolicy = `
  default-src 'self';
  script-src 'self' 'unsafe-eval';
  style-src 'self' 'unsafe-inline' fonts.googleapi
  font-src 'self' fonts.googleapis.com fonts.gstat
  img-src 'self' data: w3.org/svg/2000;
`;
```

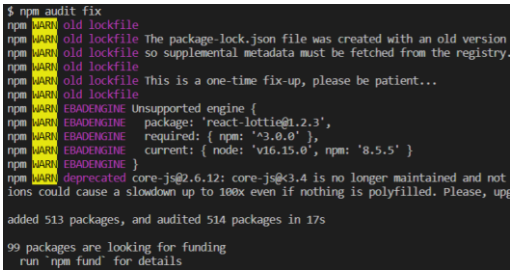
Gambar 21 Content Security Policy directives

- *Vulnerable and Outdated Components*

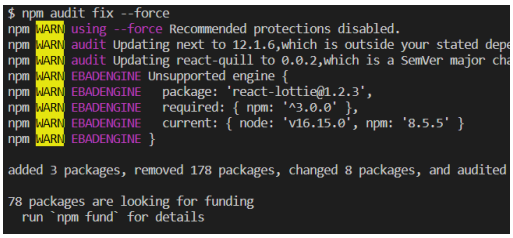
Package yang mempunyai kerentanan keamanan dan telah kadaluarsa diperbarui dengan versi yang lebih aman. Pembaruan dilakukan sesuai dengan instruksi dari hasil pemindaian menggunakan perintah `npm update`.

Terdapat dua perintah yang digunakan yaitu `npm audit fix` (Gambar 22) dan `npm audit`

`fix –force (Gambar 23)`. Penggunaan *tag* `–force` diperlukan apabila *vulnerability* terdapat hingga *root project* dan tidak dapat diperbarui tanpa mengubah dependensi yang digunakan.



Gambar 22 `Npm audit fix`

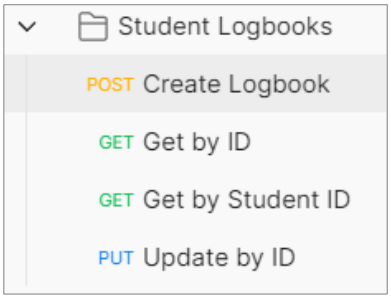


Gambar 23 `Npm audit fix --force`

F. *Pengujian*

a. *Broken Access Control*

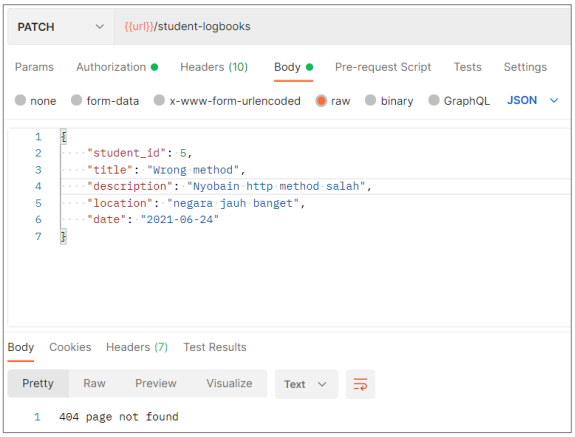
Pengujian dilakukan menggunakan *tools* Postman. Hal yang akan diuji adalah cara API dalam menangani http *method* yang salah. Salah satu API yang diuji berfungsi untuk melakukan *input* data *logbook.*



Gambar 24 Postman *http method* pengujian API

Terdapat tiga *http method* yang dapat digunakan yaitu *post, get,* dan *put*. Metode tersebut digunakan untuk melakukan proses *insert, get,* dan *update* data (Gambar 24).

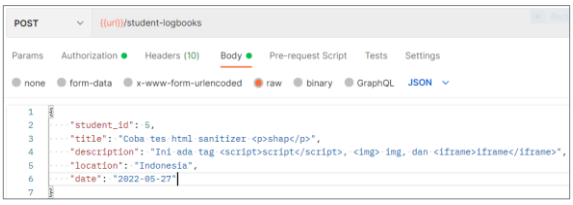
Hasil pengujian terlihat pada Gambar 25, API akan memberikan respon berupa "404 *page not found"* apabila metode yang digunakan tidak sesuai dengan yang diminta. Dengan begitu, hanya *http method* yang telah ditentukan saja yang dapat melakukan akses melalui API.



Gambar 25 Postman pengujian *wrong http method*

b. *Injection*

Pengujian injeksi dilakukan menggunakan Postman. *Tools* tersebut digunakan untuk melakukan *request* yang mana dalam pengujian ini adalah membuat *request* untuk *input* data logbook (Gambar 26). Namun, data yang diberikan mengandung *tag* html yang tidak diperbolehkan.



Gambar 26 Postman pembuatan *request* API

Pada Gambar 17, terdapat kode yang mengatur html *tag* yang dapat digunakan. Apabila terdapat *tag* lain yang masuk ke dalam *input* dan tidak sesuai dengan kriteria yang telah ditentukan, hal tersebut akan dihapus.

	nyoba xss	wa
tag ilang	Depanya ada tag img tapi dah ilang. tapi tag masih bisa	wa
Coba tes html sani	Ini ada tag , img, dan	In

Gambar 27 DBeaver hasil uji pada *database*

Hasil pada pengujian pada basis data dapat dilihat menggunakan *tools* DBeaver. Pada Gambar 27, terlihat bahwa *tag* <script>, <img>, dan <iframe> telah terhapus sedangkan *tag* , <ol>, dan <li> tidak dihapus. Hal tersebut menunjukan bahwa implementasi html *sanitizer* telah berhasil dan berjalan sesuai dengan aturan yang telah ditetapkan.

IV. KESIMPULAN

Penelitian yang telah dilakukan memberikan gambaran mengenai prosedur pengamanan sistem XYZ yang sedang dikembangkan. Hal itu dilakukan untuk memastikan dan meningkatkan keamanan sistem melalui proses pengerasan keamanan pada sisi *front-end, back-end,* dan *server* yang digunakan. Proses *security hardening* dilakukan menggunakan kerangka kerja OWASP WSTG versi stabil pada fase *During Development* dan OWASP Top 10 2021. Evaluasi keamanan dengan *code* review pada Sistem Informasi XYZ memperoleh hasil yang cukup baik. Pada API, celah

keamanan seperti SQL *Injection* telah diamankan, fitur *logging* sistem telah ditambahkan, dan sertifikat SSL telah digunakan. *Front-end* telah menerapkan *custom error* agar tidak menyebarkan informasi sistem. Dengan menggunakan OWASP Top 10 2021 sebagai *security checklist*, terdapat celah keamanan yang ditemukan dan berhasil diperbaiki dengan implementasi sanitasi *input* dan pembaruan *package*. Hasil proses pengerasan keamanan yang dilakukan masuk ke dalam tahap pengujian dan terbukti berhasil memperbaiki keamanan sistem XYZ. Penelitian mengenai *hardening* keamanan XYZ ini belum mencakup keseluruhan celah keamanan pada OWASP Top 10 2021 sehingga sebagai saran ke depan dapat dilakukan penelitian kembali pada tahap *deployment* dan tahap produksi menggunakan metode yang sama maupun metode lainnya.

V. DAFTAR PUSTAKA

[1] G. Deepa and P. S. Thilagam, "Securing web applications from injection and logic vulnerabilities: Approaches and challenges," *Information and Software Technology,* vol. 74, pp. 160-180, 27 February 2016.

[2] E. Barker, M. Smid and D. Branstad, "A Profile for U.S. Federal Cryptographic Key Management Systems," National Institute of Standards and Technology, Gaithersburg, 2015.

[3] McCamon, Mike; Blankenship, Harold; J, Lisa; Stock, Andrew van der; M, Rick; Rodriguez, Christopher; Ganne, Balakrishna Prasad; dawnaitken;, "About the OWASP Foundation," OWASP, 10 May 2022. [Online]. Available: https://github.com/OWASP/owasp.github.io/blob/main/pages/about.md. [Accessed 3 June 2022].

[4] wstgbot, "The Web Security Testing Framework," 3 December 2020. [Online]. Available: https://github.com/OWASP/www-project-web-security-testing-guide/blob/master/stable/3-The_OWASP_Testing_Framework/0-The_Web_Security_Testing_Framework.md.

[5] T. M. McCamon, H. Blankenship, A. van der Stock, B. Sulzbach, A. Falk and A. Morla, "OWASP Top Ten," 1 October 2021. [Online]. Available: https://github.com/OWASP/www-project-top-ten/blob/master/index.md.

[6] J. Firch, "Red Team VS Blue Team: What's The Difference? | Purplesec," Purplesec, 27 September 2020. [Online]. Available: https://purplesec.us/red-team-vs-blue-team-cyber-security/. [Accessed 6 July 2022].

[7] I. G. A. S. Sanjaya, G. . M. A. Sasmita and . D. M. S. Arsa, "Evaluasi Keamanan Website Lembaga X Melalui Penetration Testing Menggunakan Framework ISSAF," *Jurnal Ilmiah Merpati,* vol. 8, no. 2, pp. 113-124, 2020.

[8] S. Nagpure and S. Kurkure, "Vulnerability Assessment and Penetration Testing of Web Application," in *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)*, Pune, India, 2017.

[9] P. S. Shinde and S. B. Ardhapurkar, "Cyber Security Analysis using Vulnerability Assessment and Penetration Testing," in *World Conference on Futuristic Trends in Research and Innovation for Social Welfare (Startup Conclave)*, Coimbatore, 2016.

[10] B. Mburano and W. Si, "Evaluation of Web Vulnerability Scanners Based on OWASP Benchmark," in *IEEE International Conference on Systems Engineering*, Sydney, 2018.

[11] K. P. Alemi, "Input Validation and Input Sanitization for Web Applications," *Dissertation,* 2021.

[12] R. M. Nur, J. Na'am, G. W. Nurcahyo and S. Arlis, "Peningkatan Keamanan Website Menggunakan Metode XML dengan Framework," *Indonesian Journal of Computer Science,* vol. 8, no. 2, pp. 156-163, 2019.

[13] J. Salibindla, "Microservices API Security," *International Journal of Engineering Research & Technology,* vol. 07, no. 01, pp. 277-281, 2018.

[14] J. Singh and N. K. Chaudhary, "OAuth 2.0 : Architectural design augmentation for mitigation of common," *Journal of Information Security and Applications,* vol. 65, pp. 1-11, 2022.

[15] H. S. Lallie, L. A. Sheperd, J. R. Nurse, A. Erola, G. Epiphaniou, C. Maple and X. Bellekens, "Cyber security in the age of COVID-19: A timeline and analysis of cyber-crime and cyber-attacks during the pandemic," *Computers & Security,* vol. 105, no. 102248, pp. 1-20, June 2021.

[16] Google LLC, "Avoiding SQL injection risk," Google, [Online]. Available: https://go.dev/doc/database/sql-injection. [Accessed 13 April 2022].

[17] Google LLC, "Go Modules Reference - The Go Programming Language," [Online]. Available: https://go.dev/ref/mod#go-list-m. [Accessed 6 June 2022].

[18] G. I. E. Thomson, L. Karrys, R. Adorno, n. J. Hell and W. Chegham, "npm-audit | npm Docs," NPM, 15 April 2022. [Online]. Available: https://github.com/npm/cli/blob/latest/docs/content/commands/npm-audit.md. [Accessed 13 June 2022].

[19] S. S. Robbins, L. Robinson, J. Mendez, A. Patel, M. Hols, A. Puyou and A. Mittal, "Advanced Features: Security Headers," Vercel, 2 February 2022. [Online]. Available: https://github.com/vercel/next.js/blob/canary/docs/advanced-features/security-headers.md. [Accessed 10 June 2022].

[20] J.-Y. Perrier, w. M. Smith, H. Willee, F. Scholz, O. Ruikar and L. Medeiros, "Content Security Policy (CSP) - HTTP | MDN," Mozilla, 28 May 2022. [Online]. Available: https://github.com/mdn/content/blob/main/files/en-us/web/http/csp/index.md?plain=1. [Accessed 10 June 2022].