

TUGAS AKHIR SEMESTER GASAL

KEAMANAN PEMROGRAMAN LANJUTAN

Disusun oleh :

- Achmad Abdul Wafi (1615101198)
- Mauli Bayu Segoro (1615101234)
- Nur Annisa Kadarwati Febriyani (1615101247)
- Tio Hana Lolita Boru Lumban Tobing (1615101260)
- Tubagus Eiffel Rivaldo (1615101261)

BAB I. SKENARIO

A. Deskripsi Sistem

Sistem yang digunakan dalam praktikum Keamanan Pemrograman Lanjutan ini adalah sistem penjualan produk oleh Departemen Perniagaan. Sistem dibangun menggunakan bahasa pemrograman *Java* dengan berbasis *Desktop*. Aplikasi yang digunakan untuk membangun sistem adalah *Netbeans* dan *MySQL*. Sistem kemudian akan disebut sebagai Aplikasi Depniag.

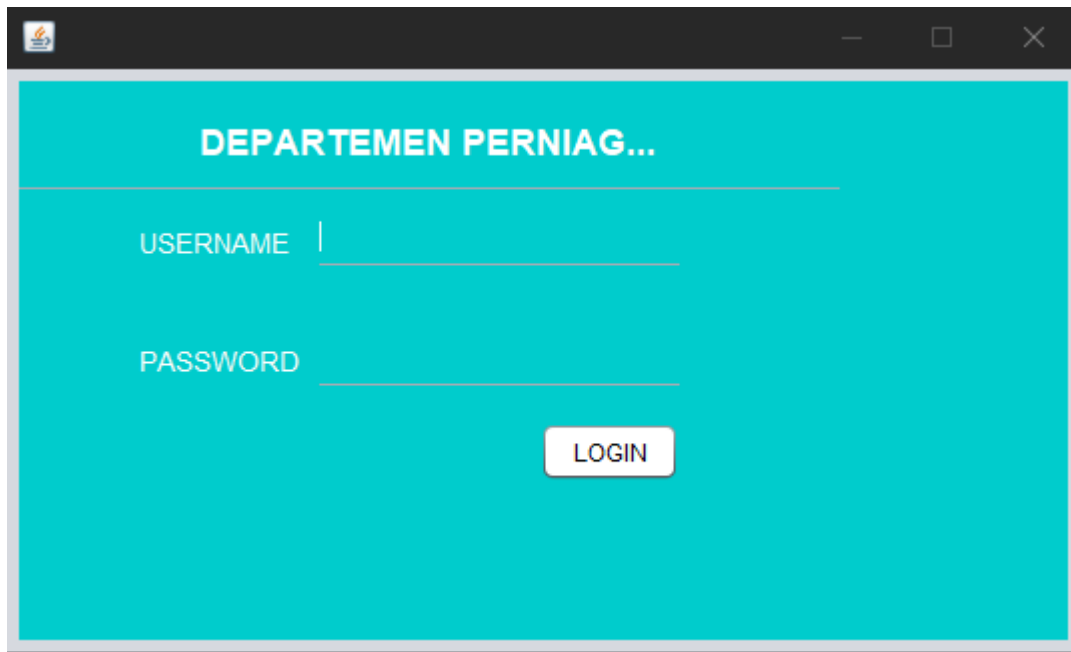
B. Tujuan Sistem

Aplikasi Depniag digunakan untuk :

1. Mempermudah proses jual-beli produk pada Departemen Perniagaan
2. Membantu kinerja Departemen Perniagaan dalam memonitor proses jual-beli produk

C. Model Aplikasi

Aplikasi Depniag terdiri dari dua halaman tampilan. Tampilan yang pertama merupakan halaman Login. Pengguna dapat menggunakan Aplikasi Depniag dengan terlebih dahulu melakukan *login* dengan memasukkan *username* dan *password* yang telah terdaftar dalam daftar pengguna pada basis data. Pengguna kemudian mengklik tombol LOGIN yang akan memeriksa *username* dan *password*. Apabila *username* atau *password* yang dimasukkan salah, aplikasi akan membuka *window* peringatan yang memberitahu bahwa *username* atau *password* salah. Apabila *username* dan *password* yang dimasukkan benar, aplikasi akan membuka tampilan kedua. Berikut adalah tampilan halaman *login* :



The image shows a web application window with a dark title bar. The main content area has a light blue background. At the top, the text 'DEPARTEMEN PERNIAG...' is displayed in bold. Below this, there are two input fields: 'USERNAME' and 'PASSWORD'. The 'PASSWORD' field has a small icon on the right side. A 'LOGIN' button is positioned below the password field.

Tampilan kedua adalah halaman *Order* atau Pemesanan. Pada halaman ini, pengguna diharuskan untuk mengisi beberapa *field* untuk membuat pemesanan. Pengguna harus mengisi Nama, No. Telp, Kelas, memilih Barang yang hendak dibeli, lalu mengisi Jumlah Barang yang dibeli. Aplikasi akan secara otomatis menghitung total harga yang harus dibayar pada *field* Total. Ketika kita memasukkan jumlah uang yang akan dibayar pada kolom Bayar, aplikasi akan menghitung kembalian secara otomatis pada *field* Kembalian. Setelah semua sudah terisi, kita mengklik tombol Hitung. Detail pemesanan akan masuk ke dalam tabel di bagian bawah dan di folder aplikasi secara otomatis akan membuat *file* struk atas pembelian produk tersebut dengan nama file [nama pembeli].txt. Pengguna juga dapat menghapus entri pada tabel dengan mengklik baris tabel lalu mengklik tombol Hapus. Berikut adalah tampilan halaman *Order* :

ORDER HERE

Nama Total Rp.

No. Telp Bayar

Kelas Kembalian Rp.

Barang Pilih ▼ Hitung

Jumlah Barang

Nomor ...	Nama	Nomor ...	Kelas	Jenis_b...	Jumlah	Harga	Tanggal
19	qwqeq...	2545078	7	Teh Ma...	1	3000	2020-0...
20	maulisds	098772...	10	Teh Ma...	3	9000	2020-0...
21	maulisds	098772...	10	Teh Ma...	3	9000	2020-0...
22	maulisds	098772...	10	Teh Ma...	3	9000	2020-0...
23	mauli	098772...	10	Teh Ma...	3	9000	2020-0...
24	asdsd	56789	10	Indomie	999998	-179497...	2020-0...
25	asa/aaaa	6789	12	Teh Ma...	2222222	-192326...	2020-0...

Wed Jan 22 11:11:41 ICT 2020 Hapus

BAB II. INSTALASI

REQUIREMENTS

Aplikasi yang akan diserang merupakan suatu aplikasi berbasis desktop dengan menggunakan bahasa pemrograman Java. Maka dari itu, ada beberapa hal yang perlu dipersiapkan, yaitu:

1. Java Runtime Environment (JRE)
2. NetBeans IDE

Aplikasi yang akan diserang terdapat pada: <https://github.com/mauli22/DepniagJava>

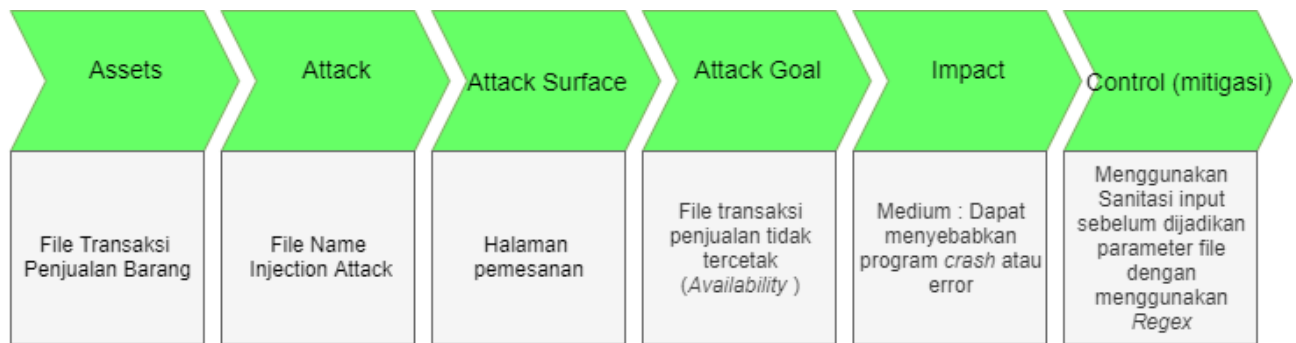
BAB III. SERANGAN

A. SERANGAN I

Serangan pertama yang dilakukan adalah menginjeksi nama *file* yang dimasukkan user pada *field input* Nama pada halaman Order. Berikut ialah rincian dari serangan :

- Tipe Serangan : *injection attack*

- *Threat Model* :



- *Attack Vector* : *field input* Nama pada halaman Order
- Eksekusi Serangan :

Ketika memasukkan nama pada *field* Nama, penyerang memasukkan karakter tengah input.

ORDER HERE

Nama

No. Telp

Kelas

Barang

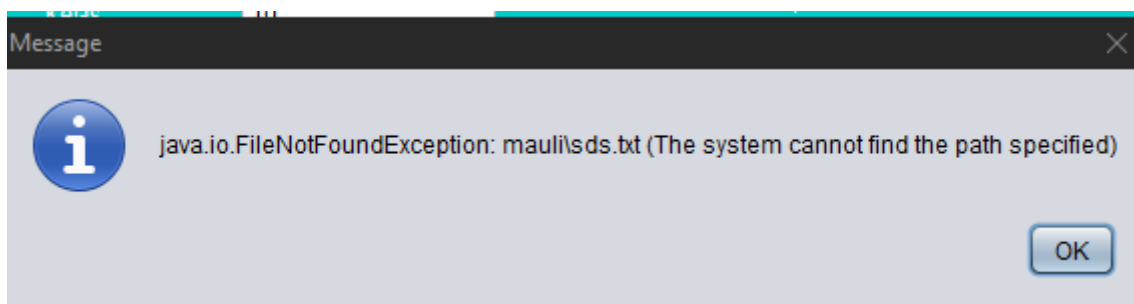
Jumlah Barang

Total Rp. 9000

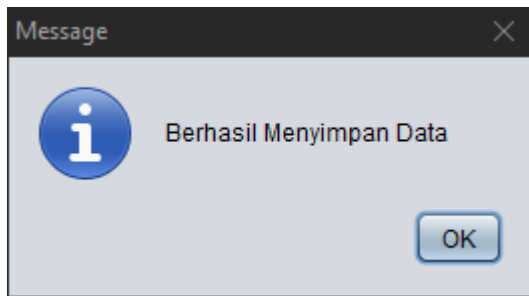
Bayar

Kembalian Rp. 11000

Saat tombol Hitung di klik sistem akan membuat *file* struk dengan nama *file* [*input field* Nama].txt . Karena penyerang memasukkan *payload* berupa **mauli\sds**, sistem akan membaca **mauli** sebagai direktori, dan **sds** sebagai nama file. Hak ini akan menyebabkan *error*, terutama apabila tidak ada direktori bernama **mauli**.



Pesanan tetap masuk ke dalam tabel daftar pesanan pada halaman Order. Namun, *file* struk tidak berhasil dibuat.



build	20/01/2020 10.49	File folder	
nbproject	08/07/2019 08.46	File folder	
src	07/07/2019 20.53	File folder	
test	07/07/2019 21.07	File folder	
build	07/07/2019 20.53	XML File	4 KB
manifest.mf	07/07/2019 20.53	MF File	1 KB

B. SERANGAN II

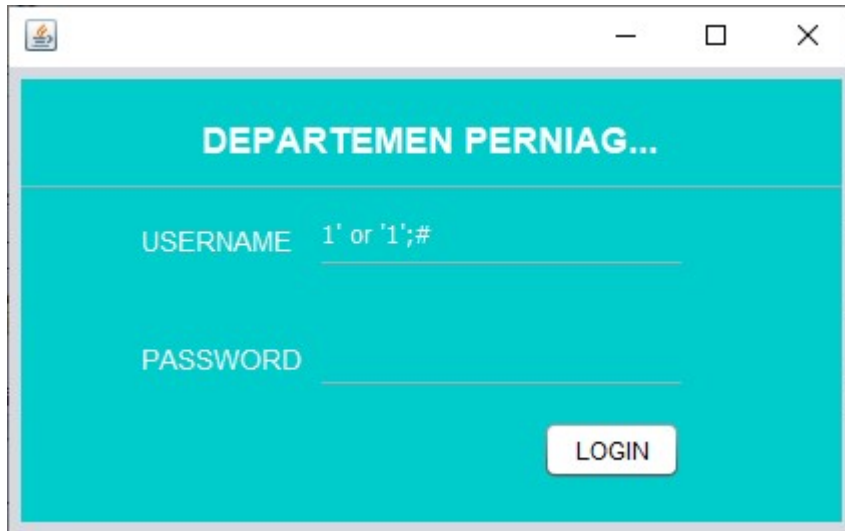
Serangan kedua yang dilakukan adalah serangan SQL *injection* yang dilakukan pada form *Login*. Berikut adalah rincian dari serangan yang dilakukan :

- Tipe Serangan : *injection attack*
- *Threat Model* :

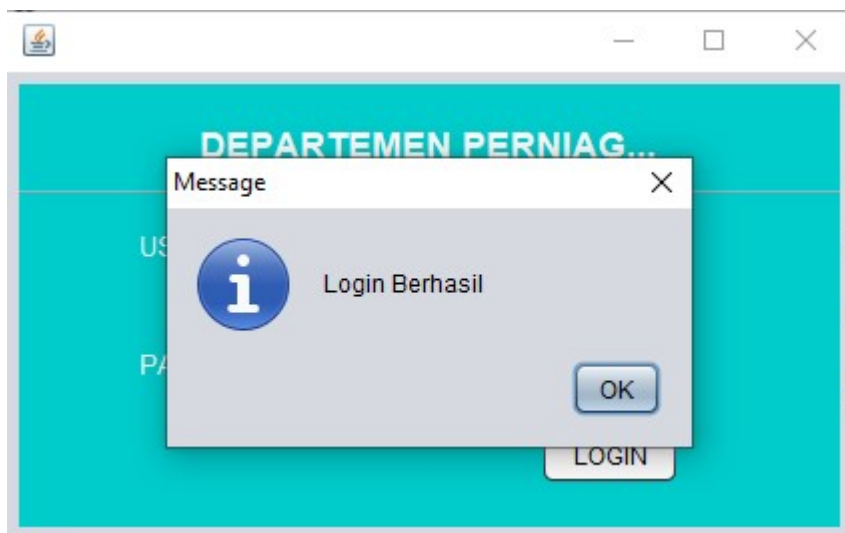
ASSET	ATTACK	ATTACK SURFACE	ATTACK GOAL	IMPACT	CONTROL (MITIGATION)
Data username dan password dari aplikasi	Injection attack	Interface aplikasi	User credential	High: dapat menyebabkan bocornya informasi rahasia	Menggunakan <i>class</i> PreparedStatement untuk pemrograman Java

- *Attack Vector* : form *Login*
- Eksekusi Serangan :

Pada saat melakukan *login*, penyerang memasukkan *payload* berupa `1' OR '1';#`



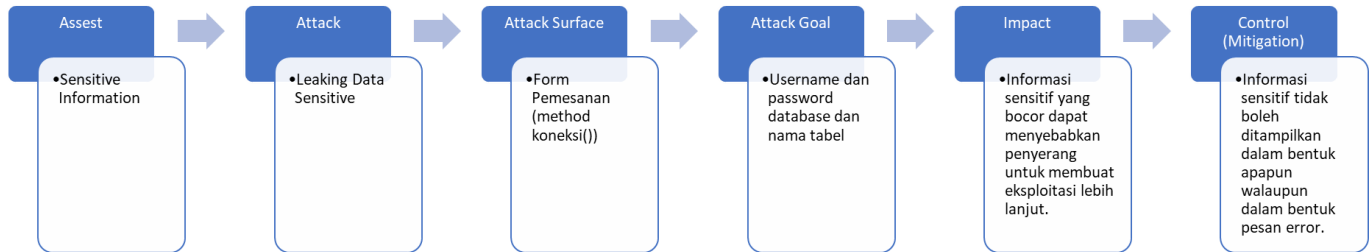
Hal ini menyebabkan commandSQL yang akan dieksekusi menjadi `SELECT * FROM tbl_login WHERE user= '1' OR '1';#`. Statement `SELECT` ini kan memilih *record* pengguna yang valid dalam `tbl_login`. *Password* tidak akan diperiksa kembali karna `user = '1'` akan dianggap valid, hal ini akan mengakibatkan semua *item* setelah *command* OR tidak dilihat kembali, dan tentunya kita dapat melakukan *bypass* pada form *login* yang ada.



C. SERANGAN III

Serangan ketiga yang dilakukan adalah keadaan dimana data sensitif ditulis secara jelas pada kode sumber (*hard-coded sensitive information*). Berikut ialah rincian serangan yang dilakukan :

- Tipe Serangan : *Leaking Sensitive Data*
- Threat Model :



- *Attack vector* : method `getKoneksi()` pada class `Koneksi`
- Eksekusi Serangan :

Penyerang dapat menggunakan command `java -c [NamaClass].class` dan mendapatkan data sensitif yang *hard-coded* pada kode sumber, seperti *username* dan *password* basis data.

```

public static java.sql.Connection getkoneksi();
Code:
  0: getstatic      #2                // Field con:Ljava/sql/Connection;
  3: ifnonnull      45
  6: new            #3                // class com/mysql/jdbc/Driver
  9: dup
 10: invokespecial  #4                // Method com/mysql/jdbc/Driver."<init>":()V
 13: invokestatic   #5                // Method java/sql/DriverManager.registerDriver:(Ljava/sql/Driver;)V
 16: ldc            #6                // String jdbc:mysql://localhost/db_depniag
 18: ldc            #7                // String root
 20: ldc            #8                // String
 22: invokestatic   #9                // Method java/sql/DriverManager.getConnection:(Ljava/lang/String;Ljava/lang
String;Ljava/lang/String;)Ljava/sql/Connection;
 25: putstatic      #2                // Field con:Ljava/sql/Connection;
 28: getstatic      #10               // Field java/lang/System.out:Ljava/io/PrintStream;
 31: ldc            #11               // String Berhasil
 33: invokevirtual #12               // Method java/io/PrintStream.println:(Ljava/lang/String;)V
 36: goto          45
 39: astore_0
 40: aload_0
 41: invokevirtual #14               // Method java/lang/Exception.printStackTrace:()[Ljava/lang/StackTraceElement;
 44: pop
 45: getstatic      #2                // Field con:Ljava/sql/Connection;
 48: areturn
Exception table:
   from    to  target type
    6      36    39    Class java/lang/Exception
  
```

D. SERANGAN IV

Serangan keempat yang dapat dilakukan ialah *integer overflow* pada form *Order*. Berikut ialah rincian serangannya :

- Tipe Serangan : *overflow (integer)*
- *Threat Model* :



- *Attack Vector* : *field* Jumlah Barang pada *method* `JumlahKeyReleased()`
- Eksekusi serangan :

Method `jumlahKeyReleased()` digunakan untuk menghitung total belanja yang harus dibayarkan dengan melakukan perkalian antara harga barang dan jumlah barang yang dimasukkan melalui *form* Order. *Payload* berupa angka besar 9663677 dimasukkan pada *field* Jumlah Barang.

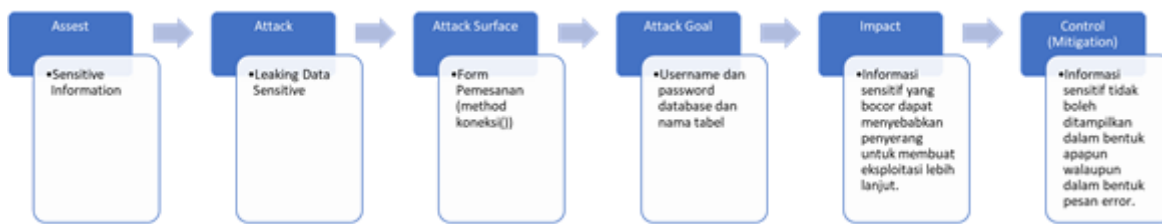
Nomor ...	Nama	Nomor ...	Kelas	Jenis_b...	Jumlah	Harga	Tanggal
15	randu	0369777	ipi	Beng-b...	4	6000	2020-0...
16				Pop Mie	2	10000	2020-0...
17	dodo	8192	2A	Teh Ma...	2	6000	2020-0...
18	alia	56	2a	Indomie	1000	2500000	2020-0...
19				Pop Mie	10000	5000000	2020-0...
20				Beng-b...	100000	200000	2020-0...
21				Beng-b...	100000	-147483...	2020-0...

Jika angka tersebut dimasukkan maka operasi yang dilakukan adalah: `int total = (int) ((total * harga_barang)); int total = (int) ((9663677 * 2000));` dengan hasil total seharusnya ialah 19327354000. Aplikasi menampilkan hasil yang tidak sesuai yakni -2147482480 karena hasil perkalian melebihi batasan dari *integer* yang akan menyebabkan hasil perhitungan yang salah.

E. SERANGAN V

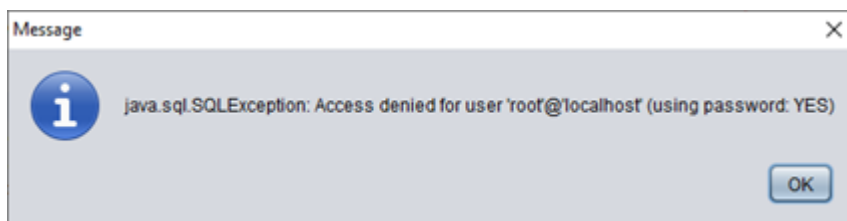
Serangan kelima yang dapat dilakukan ialah serangan dimana data sensitif ditampilkan secara *plain text*. Berikut ialah rincian serangan :

- Tipe Serangan : *Leaking Sensitive Data*
- *Threat Model* :



- *Attack vector* : method `getKoneksi()`
- Eksekusi Serangan :

Ketika menghubungkan aplikasi dengan basis data, *password* untuk basis data yang salah dimasukkan. Karena adanya *exception*, aplikasi akan memunculkan *window* pesan *error* yang berisikan variabel *e*. Namun, pesan *error* tersebut menampilkan data sensitif seperti *user* basis data dan apakah basis data menggunakan *password* atau tidak.



BAB IV. MITIGASI

A. TIPE KERAWANAN

Berikut ini merupakan kerawanan yang menyebabkan serangan pada BAB III dapat dilakukan :

1. SERANGAN I : *FileName Injection*

Berikut adalah potongan kode sumber yang masih rawan :

```

try (BufferedWriter bw = new BufferedWriter(
    new FileWriter(namafile+".txt",true))) {
    //write to file
    bw.write("-----Report Pembelian DEPNIAG "+ dtf.format(now)+"-----");
    bw.newLine();
    bw.write("Nama = "+namafile);
    bw.newLine();
}
  
```

Kerawanan ini dicatat pada CWE-116 : *Improper Encoding or Escaping of Output* . CWE tersebut menjelaskan aplikasi yang menggunakan *form input* yang disediakan, maka penyerang dapat memasukkan karakter khusus yang akan menyebabkan data ditafsirkan sebagai kontrol atau *meta data*. Akibatnya komponen yang menerima *output* akan melakukan operasi yang salah.

2. SERANGAN 2: *SQL Injection*

Kerawanan terhadap *SQL Injection* muncul ketika *query SQL* yang diinginkan penyerang eksekusi dari *query* dapat diubah menjadi *query* yang diinginkan penyerang. Eksekusi dari *query* yang telah diubah ini dapat mengakibatkan kebocoran informasi atau modifikasi data. Pada potongan kode berikut sangat dimungkinkan terjadinya *SQL Injection* karena *statement SQL* `st` menerima argumen *input* yang tidak disanitasi. Berikut adalah potongan kode sumber yang masih rawan :

```
rs = st.executeQuery("select * from tbl_login where " +
"user='"+user.getText()+"' and pass='"+String.valueOf(pass.getPassword())+"'");
```

Kerawanan ini masuk dalam CWE-89: *Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')*. Dimana CWE ini menjelaskan tentang kelemahan pada *Software* yang membangun semua atau sebagian *activity* dari perintah SQL untuk digunakan sebagai *input* yang dipengaruhi secara eksternal, tetapi tidak disanitasi atau secara salah menetralkan elemen khusus yang dapat memodifikasi perintah SQL yang dimaksud ketika dikirim ke aplikasi. Tanpa adanya sanitasi *query SQL* yang diinputkan *user*, seorang *malicious user* dapat memanfaatkan hal tersebut untuk mengubah logika *query* untuk mem-*bypass* pemeriksaan keamanan, atau untuk memasukkan statement tambahan yang dapat memodifikasi *back-end database*. *SQL Injection* sudah menjadi masalah umum pada aplikasi yang disertai dengan *database*. *Flaw* ini dapat dideteksi dengan mudah, dan mudah juga dieksploitasi.

3. SERANGAN III : *Leaking sensitive data*

Kerawanan ini merupakan masalah terkait pengungkapan atau menampilkan informasi yang sensitif seperti alamat IP, *password*, atau kunci enkripsi. Pada Aplikasi Depniag, *method* `getKoneksi()` pada *class* `Koneksi` mencoba menghubungkan basis data. Siapapun yang memiliki akses ke *file class* dapat mengkompilasi dan menemukan informasi sensitif. Berikut adalah potongan kode sumber yang masih rawan :

```
con = DriverManager.getConnection("jdbc:mysql://localhost/db_depniag", "root",
"");
```

Kerawanan ini masuk dalam CWE-116: *Improper Encoding or Escaping of Output*. CWE tersebut menjelaskan aplikasi yang menggunakan form input yang disediakan, maka penyerang dapat memasukkan karakter khusus yang akan menyebabkan data ditafsirkan sebagai kontrol atau *metadata*. Akibatnya komponen yang menerima *output* akan melakukan operasi yang salah.

4. SERANGAN IV : *Integer Overflow*

Kerawanan ini merupakan masalah yang timbul akibat tidak adanya penanganan pada saat operasi aritmetik dilakukan menghasilkan nilai melebihi rentang nilai dari tipe data *integer*. Masalah ini timbul karena pada dasarnya pada bahasa pemrograman *Java*, operator *integer* bawaan tidak menunjukkan *overflow* atau *underflow*. Berikut adalah potongan kode sumber yang rawan :

```
private void jumlahKeyReleased(java.awt.event.KeyEvent evt) {
    if (jBarang.getSelectedIndex() == 0) {
        JOptionPane.showMessageDialog(null, "Isi data!");
    } else if (jBarang.getSelectedIndex() == 1) {
        harga_barang = 2000;
    } else if (jBarang.getSelectedIndex() == 2) {
        harga_barang = 5000;
    } else if (jBarang.getSelectedIndex() == 3) {
        harga_barang = 2500;
    } else if (jBarang.getSelectedIndex() == 4) {
        harga_barang = 3000;
    }

    int totall = 0;
    totall = Integer.parseInt(String.valueOf(jumlah.getText()));
    int total = (int) ((totall * harga_barang));
    totall.setText(String.valueOf(total));
}
```

Kode diatas memungkinkan terjadinya *integer overflow* karena tidak dilakukannya pencegahan terhadap kondisi operasi yang memiliki hasil diluar batasan *integer*. Kerawanan ini masuk ke dalam CWE 682 : *Incorrect Calculation* dan CWE-190 : *Integer Overflow of Wraparound*. CWE 682 CWE-682 menjelaskan bahwa kerawanan ini terjadi pada aplikasi yang melakukan perhitungan yang menghasilkan hasil yang salah atau tidak sengaja yang akan digunakan selanjutnya pada keputusan keamanan yang kritis atau *manajemen resource*. Akibat dari kerawanan ini ialah pengalokasian resource yang salah, peberian privilege yang salah atau pebandingan nilai yang salah bahkan dapat menyebabkan kesalahan yang lebih besar yakni mekanisme keamanan yang gagal atau eksekusi kode arbitrer. Kerawanan ini dapat menimbulkan *Denial of Service* , *crash*, dan *integer overflow*. Kemudian untuk CWE-190 menjelaskan terkait terjadi akibat logika dalam aplikasi mengasumsikan bahwa nilai yang dihasilkan akan selalu lebih besar dari nilai aslinya. Lebih lanjut *integer overflow* terjadi ketika nilai *integer* ditambahkan ke nilai yang terlalu besar untuk disimpan. Hal ini dipicu dari *input* yang dimasukkan pengguna. Kerawanan ini dapat mengakibatkan DoS, *crash*, *execute unauthorized code or commands*, dan *bypass protection mechanism*.

5. SERANGAN V : *Leaking Sensitive Data*

Kerawanan ini merupakan masalah terkait pengungkapan atau menampilkan pesan *error* terhadap sesuatu yang seharusnya tidak perlu ditampilkan karena mengandung informasi yang sensitif. Pada Aplikasi Depniag, *methodKoneksi* mencoba menghubungkn aplikasi dengan basis data. Ketika nama basis data, nama pengguna, dan *password* pengguna. Namun ketika salah, maka akan menampilkan pesan *error* yang menampilkan data sensitif yang tersimpan pada variabel. Berikut adalah potongan kode sumber yang masih rawan :

```
private void koneksi() {
    try {
        Class.forName("com.mysql.jdbc.Driver");
```

```

        con = (Connection) DriverManager.getConnection
        ("jdbc:mysql://localhost/db_depniag", "root", "e");
        st = (Statement) con.createStatement();
    } catch (ClassNotFoundException | SQLException e) {
        JOptionPane.showMessageDialog(null, e);
    }
}

```

Kerawanan ini masuk dalam CWE-209, CWE-497, dan CWE-600. CWE-209 : *Information Exposure Through an Error Message* merupakan kerawanan dimana *software* atau aplikasi mengeluarkan suatu pesan *error* yang mengandung informasi sensitif seperti *environment*, *users*, atau data terkait. Selanjutnya, CWE-497 : *Exposure of System Data to an Unauthorized Control Sphere* merupakan kerawanan *software* atau aplikasi yang mengekspos *data system* atau informasi *debug* yang membantu penyerang mempelajari mengenai *system* dan membentuk rencana serangan. Yang terakhir yakni CWE-600: *** Servlet (class yang digunakan untuk menerima request dan memberi respon melalui protokol http, xml, html, file dan sebagainya) tidak menangkap semua pengecualian yang dapat mengungkapkan informasi debugging yang sensitif.*

B. MITIGASI

Berikut merupakan langkah-langkah yang dapat dilakukan untuk memperbaiki aplikasi dan menghindari serangan pada Bab III :

1. SERANGAN I : IDS05-J. *Use a subset of ASCII for file and path names.*

Untuk menghindari serangan tersebut, perlu digunakan *Regex* sebagai parameter pada *input field* Nama. Berikut adalah potongan kode sumber yang telah diperbaiki :

```

public boolean cetakStruk(String namafile){
    Pattern pattern = Pattern.compile("[^A-Za-z0-9%&+,.:=_]");
    Matcher matcher = pattern.matcher(namafile);
    if (matcher.find()) {
        // filename contains bad chars, handle error
        JOptionPane.showMessageDialog(null, "Nama tidak boleh mengandung unsur
hara !");
        return false;
    } else {
        try (BufferedWriter bw = new BufferedWriter(new
FileWriter(namafile+".txt",true)))
        {
            //write to file
            bw.write("-----Report Pembelian DEPNIAG "+ dtf.format(now)+"-----");
            bw.newLine();
            bw.write("Nama = "+namafile);
            bw.newLine();

```

2. Serangan II : IDS00-J. *Sanitize untrusted data passed accross a trust boundary*

Data yang diinputkan pengguna, seperti *parameter username* dan *password*, harus selalu dianggap sebagai suatu data yang *untrusted* dan *tainted*. Solusi yang dapat dilakukan untuk mencegah kerawanan *SQL Injection* pada aplikasi ini adalah dengan menggunakan *class PreparedStatement()* untuk memeriksa tipe *input*. Berikut adalah potongan kode sumber yang telah dipakai :

```
private void btn_loginActionPerformed(java.awt.event.ActionEvent evt){
    try{
        String username = user.getText();
        String passwd = String.valueOf(pass.getPassword());
        String query = "SELECT * FROM tbl_login WHERE user = ? AND pass = ?";

        PreparedStatement stmt = con.prepareStatement(query);
        stmt.setString(1, username);
        stmt.setString(2, passwd);
        rs = stmt.executeQuery();
    }
}
```

3. SERANGAN III : MSC03-J. *Never hardcode sensitive information.*

Langkah untuk mencegah kerawanan ini adalah menaruh data rahasia pada suatu *file* dan *file* baru akan dibaca saat program berjalan. Berikut adalah potongan kode sumber yang telah diperbaiki :

```
Properties properties = new Properties();
properties.load(new FileInputStream(new File("c:\\credentials.properties")));

String user = properties.getProperty("username");
String pass = properties.getProperty("password");
//...
```

4. SERANGAN IV : NUM00-J. *Detect or prevent integer overflow*

Solusi yang dilakukan untuk memitigasi terjadinya serangan ini ialah melakukan pencegahan dengan melakukan pengecekan *input* untuk setiap operasi aritmetik untuk mencegah munculnya *overflow*. Berikut adalah potongan kode sumber yang telah diperbaiki :

```
private void jumlahKeyReleased(java.awt.event.KeyEvent evt) {
    if (jBarang.getSelectedIndex() == 0) {
        JOptionPane.showMessageDialog(null, "Isi data!");
    } else if (jBarang.getSelectedIndex() == 1) {
        harga_barang = 2000;
    } else if (jBarang.getSelectedIndex() == 2) {
        harga_barang = 5000;
    } else if (jBarang.getSelectedIndex() == 3) {
        harga_barang = 2500;
    } else if (jBarang.getSelectedIndex() == 4) {
        harga_barang = 1000;
    }
}
```

```

        harga_barang = 3000;
    }

    int totall = 0;
    totall = Integer.parseInt(String.valueOf(jumlah.getText()));
    int total = safeMultiply(totall, harga_barang);
    totall.setText(String.valueOf(total));
}

static final int safeMultiply(int totall, int harga_barang) throws
ArithmeticException{
    if(harga_barang>0 ? totall>Integer.MAX_VALUE/harga_barang ||
    totall<Integer.MIN_VALUE/harga_barang : (harga_barang<-1 ?
    totall>Integer.MIN_VALUE/harga_barang ||
    totall<Integer.MAX_VALUE /harga_barang : harga_barang ==
    == Integer.MIN_VALUE)){
        throw new ArithmeticException("Integer Overflow");
    }
    return totall * harga_barang;
}

```

Kode sumber diatas menunjukkan penambahan *method* `safeMultiply()` untuk memastikan setiap *input* yang dihitung akan menghasilkan *output* nilai yang benar.

5. SERANGAN V : ERR01-J. *Do not allow exceptions to expose sensitive information*

Untuk menghindari *window* pesan *error* dari *exception* menampilkan data sensitif kita perlu menentukan pesan apa yang akan ditampilkan pada *window* tersebut. Pesan yang akan ditampilkan adalah *404 Database Not Found*. Berikut adalah potongan kode sumber yang telah diperbaiki :

```

private void koneksi(){
    try {
        Class.forName("com.mysql.jdbc.Driver");
        con = (Connection) DriverManager.getConnection
            ("jdbc:mysql://localhost/db_depniag", "root", "e");
        st = (Statement) con.createStatement();
    } catch (ClassNotFoundException | SQLException e) {
        JOptionPane.showMessageDialog(null, "404 Database Not Found !");
    }
}

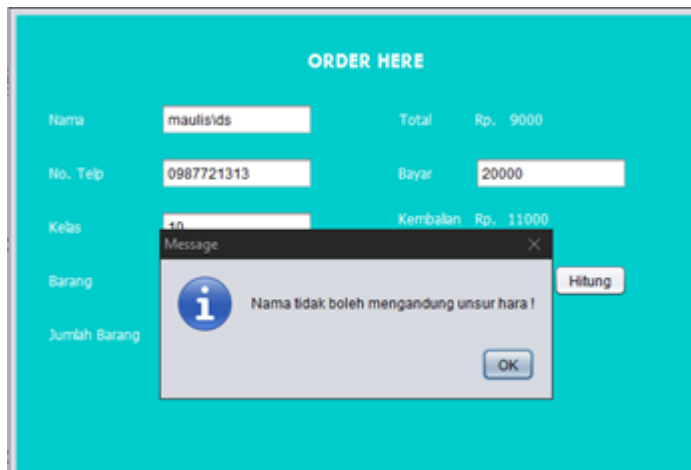
```

C. PENGUJIAN

Berikut ini adalah hasil ketika dilakukan serangan ulang pada aplikasi yang telah diberikan langkah mitigasi :

1. SERANGAN I

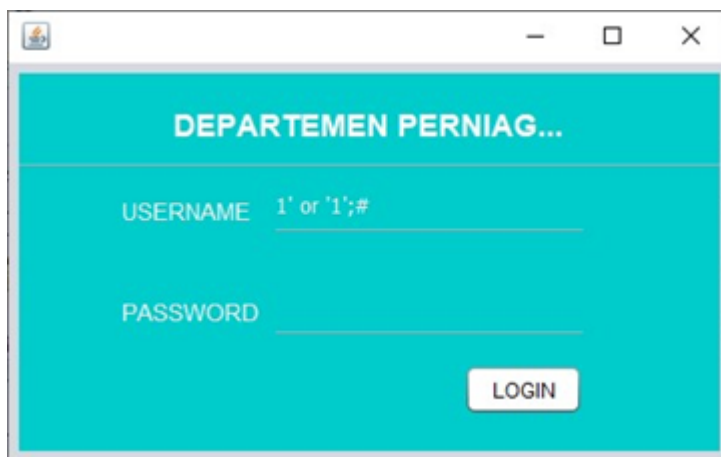
Serangan kembali dilakukan pada bagian *field* Nama pada halaman Order yang sudah diperbaiki .



The screenshot shows a web form titled "ORDER HERE" with a teal background. It contains input fields for "Nama" (filled with "maulisids"), "No. Telp" (filled with "0987721313"), "Kelas" (filled with "4n"), "Total" (Rp. 9000), "Bayar" (20000), and "Kembalian" (Rp. 11000). There are also fields for "Barang" and "Jumlah Barang". A "Hitung" button is visible. A modal message box is displayed in the center, stating "Nama tidak boleh mengandung unsur hara !" with an "OK" button.

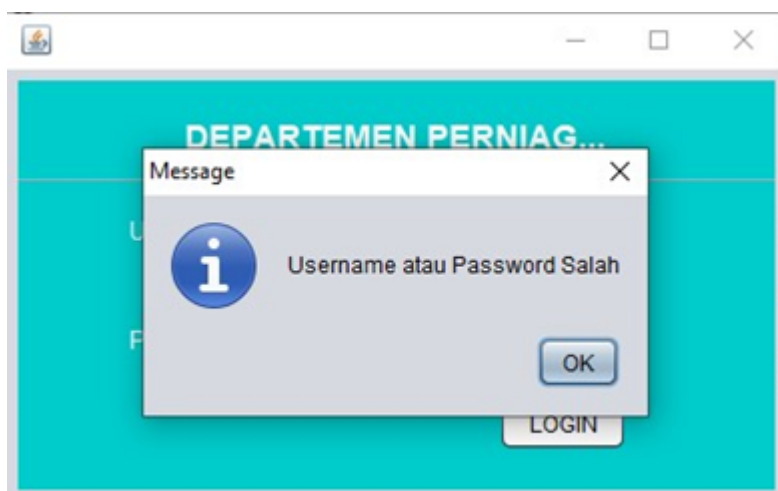
2. SERANGAN II

Dilakukan serangan *SQL Injection* kembali pada *form Login* Aplikasi Depniag yang sudah diperbaiki :



The screenshot shows a web form titled "DEPARTEMEN PERNIAG..." with a teal background. It has input fields for "USERNAME" and "PASSWORD". The "USERNAME" field contains the payload "1' or '1';#". A "LOGIN" button is at the bottom right.

Namun serangan gagal karena telah dilakukan pengamanan dan *query SQL* tidak secara langsung dieksekusi.



The screenshot shows the same login form as before, but with a modal message box displayed. The message box states "Username atau Password Salah" (Username or Password Wrong) and has an "OK" button. The "LOGIN" button is still visible in the background.

3. SERANGAN III

Walaupun penyerang dapat mendekompilasi kode sumber dari Aplikasi Depniag , penyerang tidak akan bisa mendapatkan *file* rahasia yang berisikan data sensitif.

ORDER HERE

Nama	<input type="text" value="maulisds"/>	Total	Rp. 9000
No. Telp	<input type="text" value="0987721313"/>	Bayar	<input type="text" value="20000"/>
Kelas	<input type="text" value="10"/>	Kembalian	Rp. 11000
Barang			
Jumlah Barang			



Nama tidak boleh mengandung unsur hara !

OK

Hitung

Ketika dilakukan serangan kembali pada Aplikasi Depniag yang sudah diperbaiki, hasil perhitungan Aplikasi yang ditampilkan sudah benar.

ORDER HERE

Nama

No. Telp

Kelas

Barang Beng-beng ▼

Jumlah Barang

Total Rp. 1932734000

Bayar

Kembalian Rp. Uang Kurang!

Hitung

Cetak

Nomor ...	Nama	Nomor ...	Kelas	Jenis_b...	Jumlah	Harga	Tanggal
16	dodo	8192	2A	Pop mie	2	10000	2020-0...
17	alia	56	2a	Teh Ma...	2	6000	2020-0...
18				Indomie	1000	2500000	2020-0...
19				Pop Mie	10000	500000...	2020-0...
20				Beng-b...	100000	200000...	2020-0...
21				Beng-b...	100000...	-147483...	2020-0...
22				Beng-b...	9663677	-214748...	2020-0...

Hapus

5. SERANGAN V

Telah dilakukan serangan ulang pada Aplikasi Depniag yang sudah diperbaiki. Ketika penyerang memasukkan *password* yang salah untuk basis data, *window* pesan *error* yang keluar akan menampilkan pesan yang telah ditentukan sebelumnya.

