10 WEEKS TO 0 CRITICAL VULNERABILITIES

BROKEN AUTHENTICATION

Sherif Koussa

@skoussa

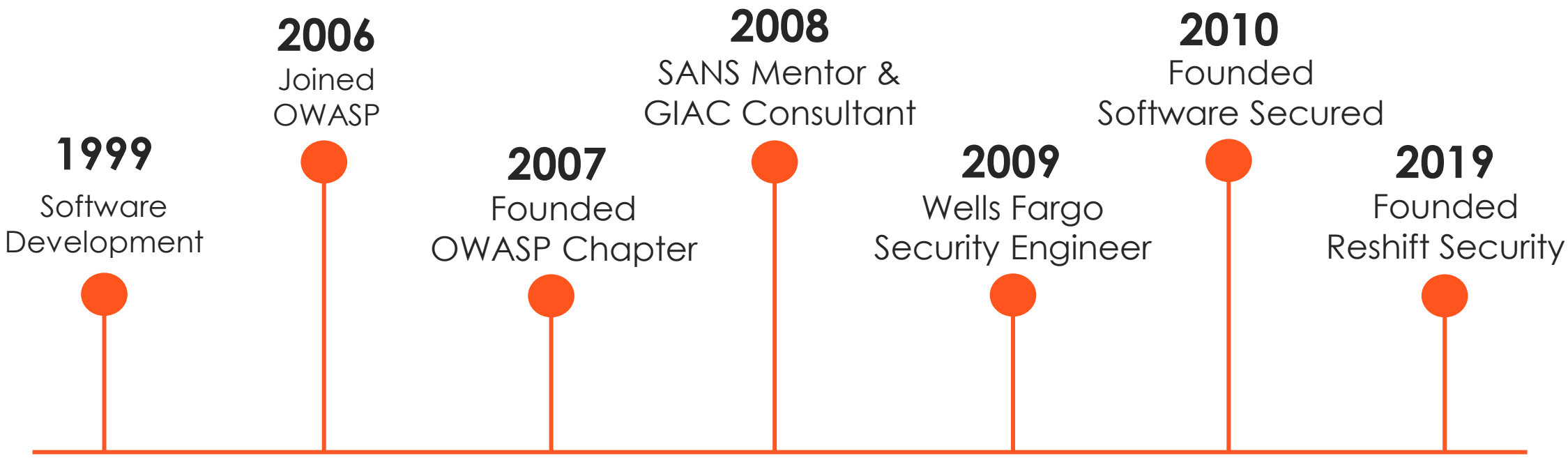
SoftwareSecured

HOUSEKEEPING

- Why we're all here
- Recording for internal training purposes
- Slides will be provided after the session
- Your mic will be muted, please use "Q&A" for any questions

SoftwareSecured

ABOUT ME



Certifications: GSSP-Java, GSSP-NET, GWAPT

SoftwareSecured

Reshift integrates with your modern software development pipeline to help your team find and fix vulnerabilities.

Penetration Testing as a Service company based out of Ottawa, Canada.

10 WEEK SCHEDULE

1. April 10th: Injection
2. **April 17th : Broken Authentication**
3. April 24th: Sensitive data Exposure
4. May 1st : External Entity Injection
5. May 8th : Broken Access Control
6. May 15th: Security Misconfiguration
7. May 22nd: Cross-site Scripting
8. May 29th: Insecure Deserialization
9. June 5th: Using Components with Known Vulnerabilities
10. June 12th: Insufficient Logging and Monitoring

BROKEN AUTHENTICATION

SESSION 2: AGENDA

1. What is Broken Authentication
2. Authentication process
3. Exercise: exploiting authentication issues
4. Mitigating authentication issues
5. Security principal: multi-factor authentication

WHAT IS BROKEN AUTHENTICATION?

- A result of an inadequately developed login page, authentication logic, backdoor, custom session management or authentication scheme.

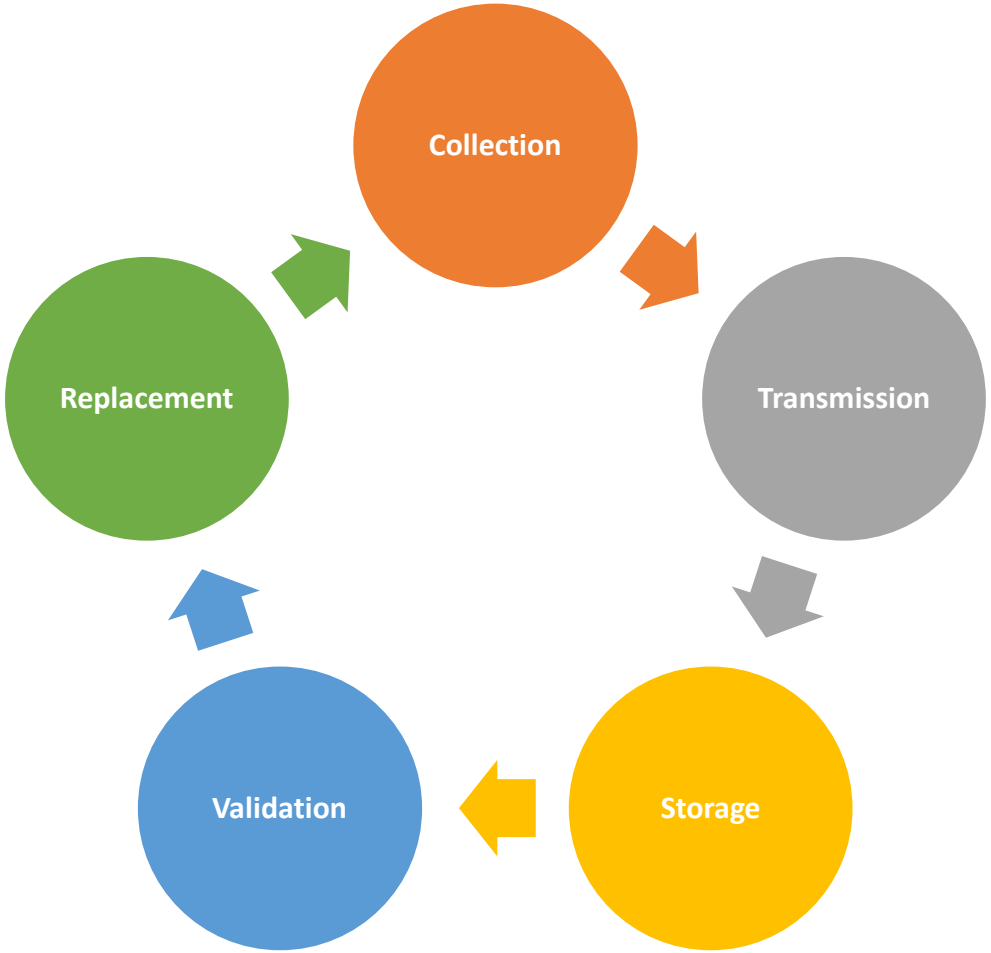
- Frequently have flaws in areas such as logout, password management, timeouts, remember me, secret question, account update, etc.

- **Exploitability: AVERAGE**
- **Prevalence: COMMON**
- **Detectability: AVERAGE**
- **Impact: SEVERE**

REAL COMPANY INJECTION ATTACKS

- **2020 PayPal Major Login Vulnerability**
- **2019 Instagram Vulnerability** – 1 Million Accounts Lost
- **2019 LastPass Vulnerability**

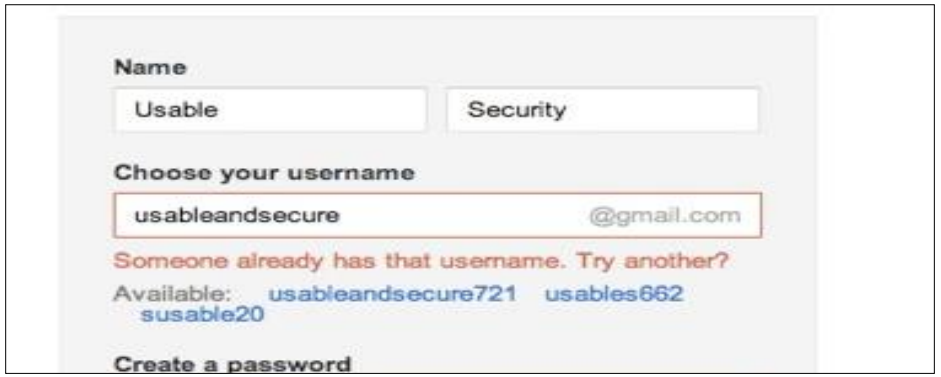
AUTHENTICATION PROCESS



AUTHENTICATION PROCESS | COLLECTION

- **Username Enumeration:**
  - Is an attack that aims at enumerating valid usernames that can be used later for a DoS attack or a Brute Force attack.

- **Caused By:**
  - Application trying to be too usable
  - Friendly error Message at Login



AUTHENTICATION PROCESS | COLLECTION

- **Brute Force Attacks/Dictionary Attacks:**

Attack that aims at guessing account credentials using brute force techniques, common passwords or dictionary words

- **Caused By:**
  - Very short passwords
  - Weak password policy.
  - Common dictionary words
  - 1337 Dictionary words
  - Pre/Appending predictable values to dictionary words

AUTHENTICATION PROCESS | TRANSMISSION

Consider the following example:

http://travelbargains.com/sale/destinations?sessionid=268544541&dest=Hawaii

- The application supports session URL rewriting
- URLs are shared, logged and emailed.
- This could allow other users to hijack the session

AUTHENTICATION PROCESS | TRANSMISSION

Consider the following example. What is the issue?

```
$.ajax({
   url: $(this).attr('action'),
   type: 'GET',
   data: $(this).serialize(),
   username: "user",
   password: "password",
   success: function(data){
   //var data = JSON.parse(jsondata);
     console.log(data);
   },
});
```

- Credentials are sent in the URL
- URLs are shared, logged and emailed.
- This could allow attackers to hijack users' accounts

AUTHENTICATION PROCESS | TRANSMISSION

- **Vulnerable Transmission of Credentials:** occurs when transmitting credentials to the server is done in an insecure manner leaving it vulnerable to Man-in-the-Middle attack or other attacks that would result in unauthorized access to users accounts
- **Caused By:**
  - Basic Authentication over HTTP: credentials are transmitted in encoded clear text over the network
  - Credentials in the URL: credentials are displayed in clear text and logged in log servers, proxy servers, log files, etc.
  - Credentials Stored in Cookies: credentials captured and replayed (Remember Me Functionality)

AUTHENTICATION PROCESS | STORAGE

- **Insecure Password Storage:** happens when user's passwords are stored in an insecure manner inside the database, this could lead to unauthorized access to users' account in case of a data breach.
- **Cause By:**
  - Storing passwords in plaintext format.
  - Storing passwords using reversible encryption algorithm.
  - Using weak hashing algorithm.
  - Lack of SALTs

AUTHENTICATION PROCESS | VALIDATION

Consider the following example. What is the issue?

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

  try {
    String username = request.getParameter("username");
    String password = request.getParameter("password");

    Boolean isFound = fetchUser (username, password);
    if (!isFound){
      response.sendRedirect("/login");
      log("Login Failed");
    }
  }
  catch (Exception ex)
  {
    log(ex.getMessage());
  }
  response.sendRedirect("/homepage");
  log("Login Succeeded");
}
```

AUTHENTICATION PROCESS | VALIDATION

- **Fail-open Login Mechanism:** Fail-open authentication is the situation when the user authentication fails but results in providing open access to authenticated and secure sections of the web application to the end user.
- **Logic Issues:** occurs when the application does not count for all the possible scenarios.

AUTHENTICATION PROCESS | VALIDATION

Consider the following example:

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws Exception {
    try
    {
      String username = request.getParameter("username");
      String password = request.getParameter("password");

      Boolean isFound = fetchUser(username, password);
      if (isFound || Boolean.getBoolean(request.getParameter("isAdmin")))
      {
        response.sendRedirect("/homepage");
        log("Login Succeeded");
      }
    }
    catch(Exception ex)
    {
      log(ex.getMessage());
    }
    response.sendRedirect("/login");
    log("Login Failed");
  }
```

Setting isAdmin to ”true” will grant access regardless of credentials!

AUTHENTICATION PROCESS | VALIDATION

Consider the following example:

```
String[] ids = {"334364642945","94985984723","98759390257"};
protected void doPost3(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

  try {
    String username = request.getParameter("username");
    String password = request.getParameter("password");

    Boolean isFound = fetchUser (username, password);
    if (isFound || Arrays.asList(ids).contains(username)){
      response.sendRedirect("/homepage");
      log("Login Succeeded");
    }
  }
  catch (Exception ex)
  {
    log(ex.getMessage());
  }
  response.sendRedirect("/login");
  log("Login Failed");
}
```

User granted unconditional access based on a hardcoded ID!

AUTHENTICATION PROCESS | VALIDATION

- **Backdoors:** a hidden way of authentication that bypasses all the standard application authentication mechanisms.
- **Caused By:**
  - Special Parameters
  - Hardcoded passwords or IDs
  - Secret URLs

AUTHENTICATION PROCESS | REPLACEMENT

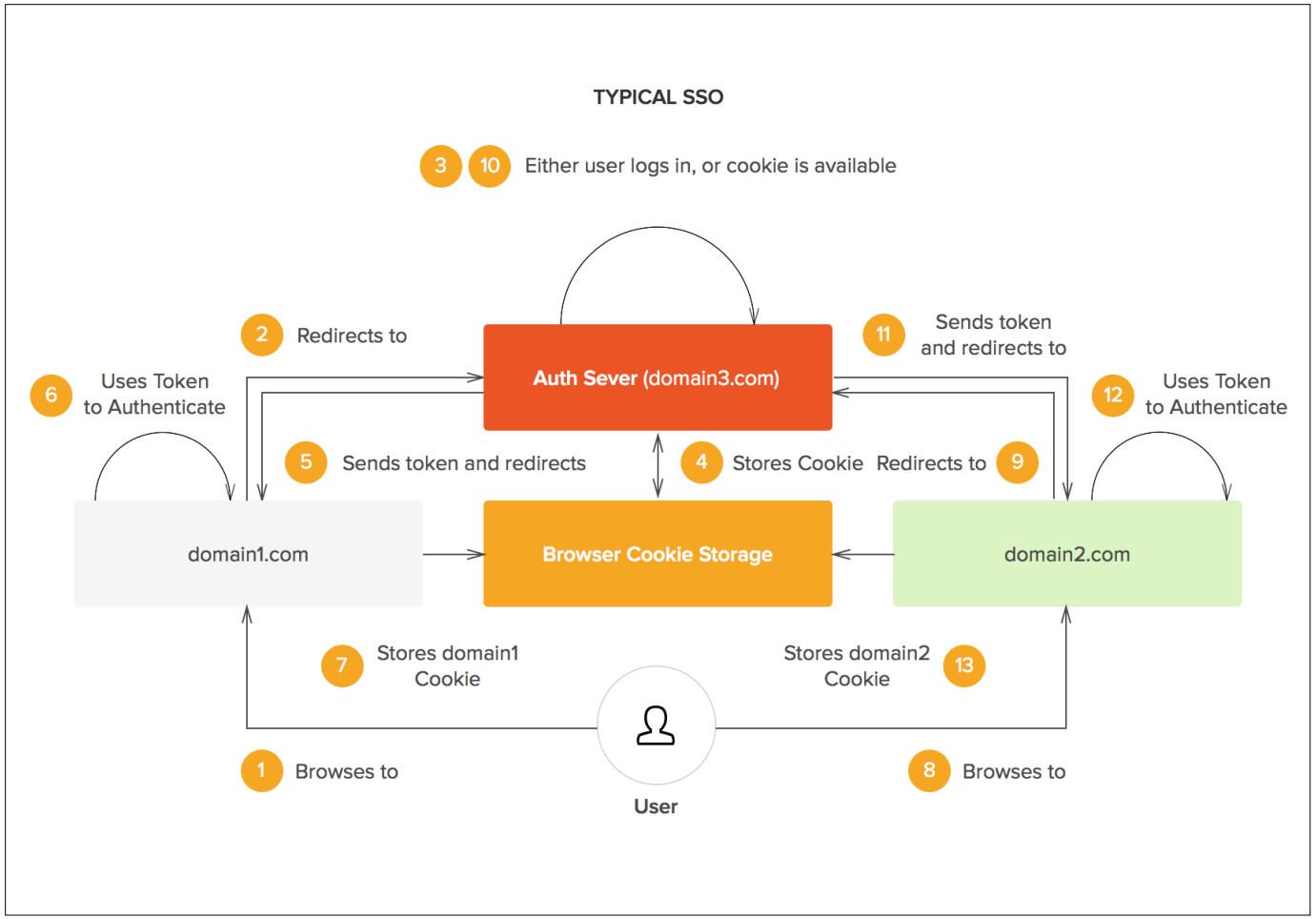
- **Insecure Change Password:** occurs when the application does not tighten security controls around the change password process.
- **Caused By:**
  - Failure to require the old password before accepting the new one.
  - Failure to enforce a strong password history policy.

AUTHENTICATION PROCESS | REPLACEMENT

- **Insecure Forgot Password Mechanism:** occurs when the application does not tighten security controls around the forgotten password process.
- **Caused By:**
  - Displaying the password after answering a security question that has a small set of answers (e.g. what's your favorite color)
  - Sending the username and password together in the same email.
  - Sending predictable reset token

AUTHENTICATION PROCESS | FEDERATION/SSO

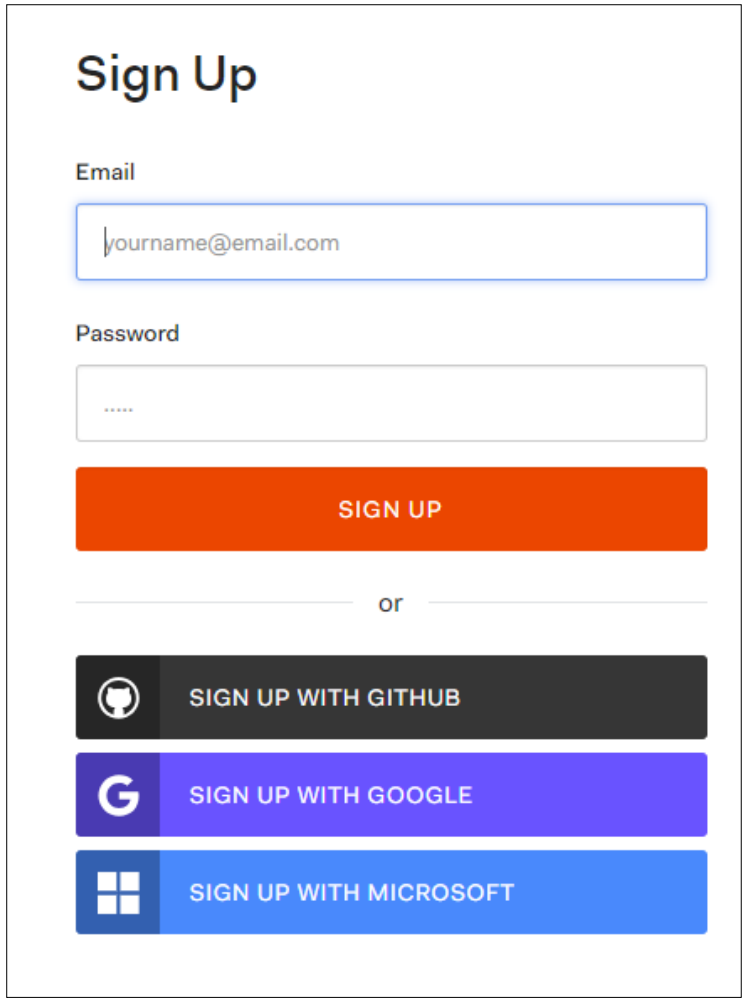
- Relies on third-party provider to authenticate a user
  - OpenId Connect
  - Auth0
  - Microsoft Account
- Representations
  - SAML (Security Assertion Markup Language)
  - JWT (JSON Web Token)
- Federation can handle Authentication, Authorization, User Attribute Exchange and User Management
- Allows a user to log in once and access all compatible sites



Typical SSO Flow

- Central Authentication Domain
- Shared sessions with other domains/applications

EXAMPLE FEDERATED PROVIDERS

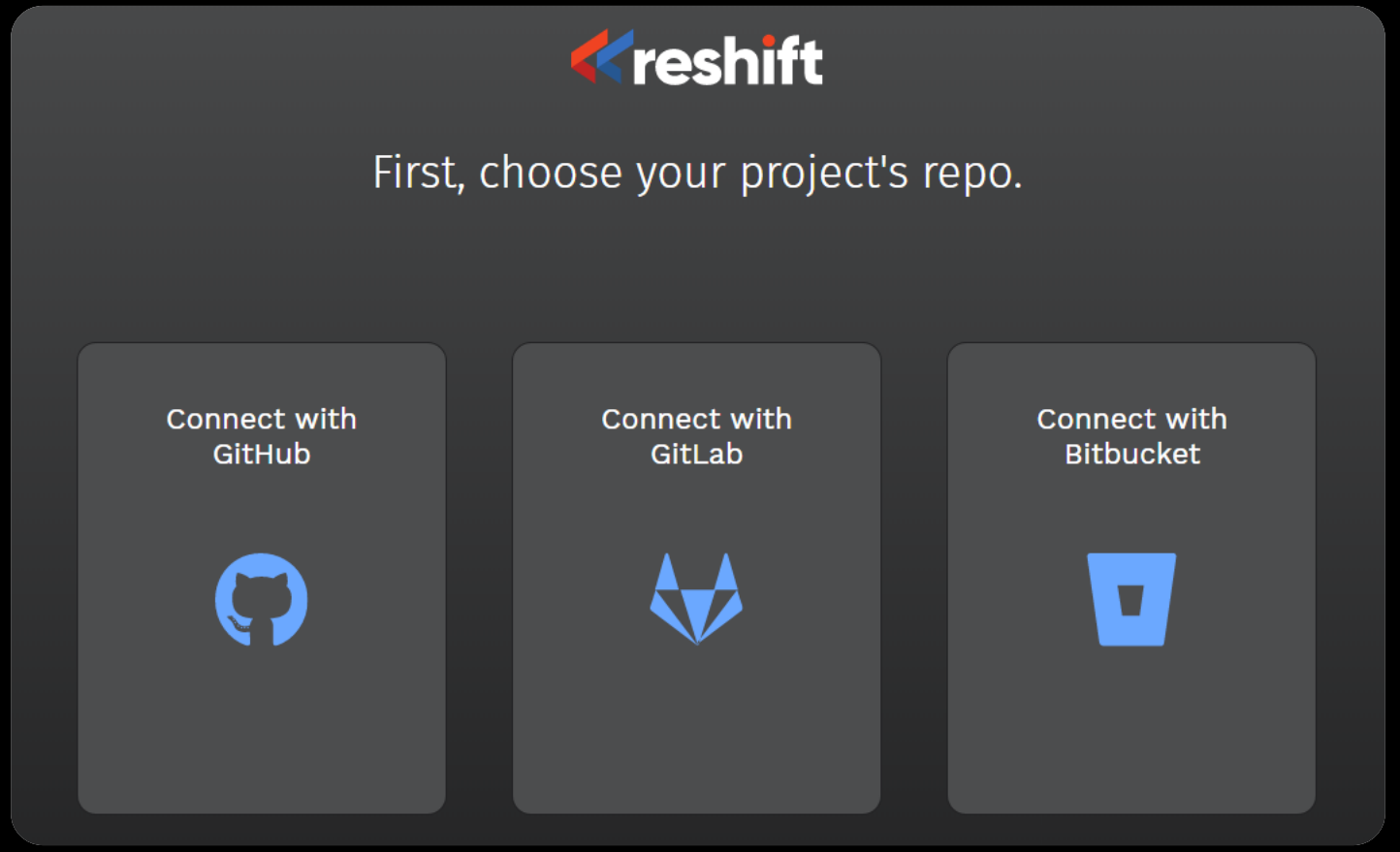


SoftwareSecured

VULNERABILITIES

- Many JWT implementations support “none” as signature algorithm
- JWT implementations lack definition of verification algorithm in function
- SAML implementations can have radically different usernames based on comment placement in an XML name assertion
- SAML, based on XML, is often affected by XML canonicalization

EXERCISE: INSECURE AUTHENTICATION



SoftwareSecured

MITIGATING AUTHENTICATION ISSUES | COLLECTION

- **Protect Against User Enumeration:**
  - Registration Pages: disable functionality upon the receipt of several request within a short period of time
  - Login Screens: display a generic invalid login message.
  - Forget Passwords: display a generic message regardless if the email was actually found

MITIGATING AUTHENTICATION ISSUES | TRANSMISSION

- **Protect Against Insecure Transmission of Credentials**
  - All authentication screens must be communicated over HTTPS.
  - Credentials should be submitted to the server using POST Requests only.
  - Avoid Remember Me functionalities.

MITIGATING AUTHENTICATION ISSUES | STORAGE

- **Use Strong Password Policy**
  - Enforce strong passwords that consists of minimum eight characters and consists of upper and lower case letters, digits and special characters.
  - Force users to change their passwords periodically.
  - Prevent users from reusing their last 6 passwords.

HOW WEAK ARE WEAK PASSWORDS?

https://password.kaspersky.com/

MITIGATING AUTHENTICATION ISSUES | STORAGE

- **Store Credentials Securely:**
  - Store passwords hashed and SALTed using a unique SALT per user in the database. Ensure SALTs have high entropy
  - Use adaptive hashing algorithms.

- **Prevent Against Brute Force Attacks:**
  - Enable account lockouts after a certain number of unsuccessful logins.
  - Leverage throttling to avoid denial-of-service attacks.
  - Use Captcha's to differentiate Humans from Robots

MITIGATING AUTHENTICATION ISSUES | REPLACEMENT

- **Prevent Account Recovery Abuse:**
  - Users should be asked to answer at least two security questions that do not have a limited set of answers.
  - A time sensitive unique link should be sent to the user's email, that redirect users to choosing a new password.

MITIGATING AUTHENTICATION ISSUES | SSO

- **Avoid SAML**
  - Leverage newer technologies like OpenID, Passport, Facebook Account
- **Blacklist algorithms and leverage encryption in JWT**
  - Most attacks rely on abusing data the attacker can control. This includes the algorithm for the token and verification signature.
- **Follow release notes and news on libraries used**
  - Authentication libraries that are well maintained react quickly to vulnerabilities
  - Responsible libraries inform their user community.

SECURITY PRINCIPAL | MULTI-FACTOR AUTHENTICATION

Multi-factor authentication involves the usage of two or more of the following

- Something You Know (e.g. password)
- Something You Have (e.g. ATM cad)
- Something You Are (e.g. fingerprint)

MULTI-FACTOR AUTHENTICATION ATTACKS

- SIM Swapping
- Revere Proxy Tools (e.g. Modlishka)
- Session fixation
- Man-in-the-endpoint Attacks

TOOLS

- Reshift security: great at finding weak crypto used in authentication scenarios
- Modlishka reverse proxy: a great tool to test multi-factor mechanisms.
- JWT.io: a great tool to debug your JWTs
- Kaspersky Password Checker: to test password strengths

RESOURCES:

- Comparing SAML vs OpenID vs oAuth: comparing the three protocols from a security perspective.
- OWASP Authentication Cheat Sheet
- OWASP Multifactor Authentication Cheat Sheet
- Forgot Password Cheat Sheet
- Cryptographic Best Practices

10 WEEKS TO 0 VULNERABILITIES PROGRAM

1. April 10th: Injection
2. April 17th : Broken Authentication
3. **April 24th: Sensitive Data Exposure**
4. May 1st : External Entity Injection
5. May 8th : Broken Access Control
6. May 15th: Security Misconfiguration
7. May 22nd: Cross-site Scripting
8. May 29th: Insecure Deserialization
9. June 5th: Using Components with Known Vulnerabilities
10. June 12th: Insufficient Logging and Monitoring

Sign up for next week's session:

Register

SoftwareSecured

SoftwareSecured

THANK YOU ☺

sherif@softwaresecured.com
@skoussa