THE CUSTOMER SUCCESS PLATFORM
SALES SERVICE MARKETING COMMUNITY ANALYTICS APPS

Secure Coding
External App Integrations

Astha Singhal
Product Security Engineer
salesforce.com

Tim Bach
Product Security Engineer
salesforce.com

dreamforce

Safe Harbor

Safe harbor statement under the Private Securities Litigation Reform Act of 1995:

This presentation may contain forward-looking statements that involve risks, uncertainties, and assumptions. If any such uncertainties materialize or if any of the assumptions proves incorrect, the results of salesforce.com, inc. could differ materially from the results expressed or implied by the forward-looking statements we make. All statements other than statements of historical fact could be deemed forward-looking, including any projections of product or service availability, subscriber growth, earnings, revenues, or other financial items and any statements regarding strategies or plans of management for future operations, statements of belief, any statements concerning new, planned, or upgraded services or technology developments and customer contracts or use of our services.

The risks and uncertainties referred to above include – but are not limited to – risks associated with developing and delivering new functionality for our service, new products and services, our new business model, our past operating losses, possible fluctuations in our operating results and rate of growth, interruptions or delays in our Web hosting, breach of our security measures, the outcome of any litigation, risks associated with completed and any possible mergers and acquisitions, the immature market in which we operate, our relatively limited operating history, our ability to expand, retain, and motivate our employees and manage our growth, new releases of our service and successful customer deployment, our limited history reselling non-salesforce.com products, and utilization and selling to larger enterprise customers. Further information on potential factors that could affect the financial results of salesforce.com, inc. is included in our annual report on Form 10-K for the most recent fiscal year and in our quarterly report on Form 10-Q for the most recent fiscal quarter. These documents and others containing important disclosures are available on the SEC Filings section of the Investor Information section of our Web site.

Any unreleased services or features referenced in this or other presentations, press releases or public statements are not currently available and may not be delivered on time or at all. Customers who purchase our services should make the purchase decisions based upon features that are currently available. Salesforce.com, inc. assumes no obligation and does not intend to update these forward-looking statements.

Astha Singhal

Product Security Engineer

Astha Singhal

- Working with product teams from design to implementation to help them build secure applications for our customers.
- Conduct penetration tests on Salesforce applications.
- Facilitating the security process via better security training and enabling self-service for product teams.
- Helping them understand security bugs and guiding through remediation of security issues.

Tim Bach

Product Security Engineer

Tim Bach

- Conduct security penetration tests on your products
  - (Sorry)
- Develop security automation tools to help partners prepare for security review and increase first-time-pass rate
- Develop proactive vulnerability detection and notification tools for partner products
  - Coming to you soon!
- Work with partners to guide the remediation process before resubmission

App Integrations

App Integrations

- Extend Salesforce functionality with external app integrations
- Need a way to map Salesforce user identity to these external systems
- Need a way to grant access to Salesforce data without breaking the Salesforce security model or trust in the Salesforce platform

Integration Methods

- API / OAuth
  - External services authenticate with Salesforce via OAuth and receive access tokens
  - Tokens must be treated with same sensitivity as a password
  - Utilize public-facing API's to share data with Salesforce instances
  - Developers can expose custom Apex REST endpoints

Integration Methods

- Connected App
  - Runs on the Salesforce app canvas
  - Does not have access to the Salesforce app DOM at any time
  - Authenticate via OAuth or SAML using Salesforce credentials
  - Easy way to integrate an external application into the Salesforce “skin”
  - The OAuth scope for the connected app determines the amount of access this app would have to your Salesforce data

Integration Methods

- Apex Callouts
  - Use Apex code to access external REST API's
  - Can be used to send data out or pull data in to/ from an external service
  - Actions must be initiated by a user action from within Salesforce

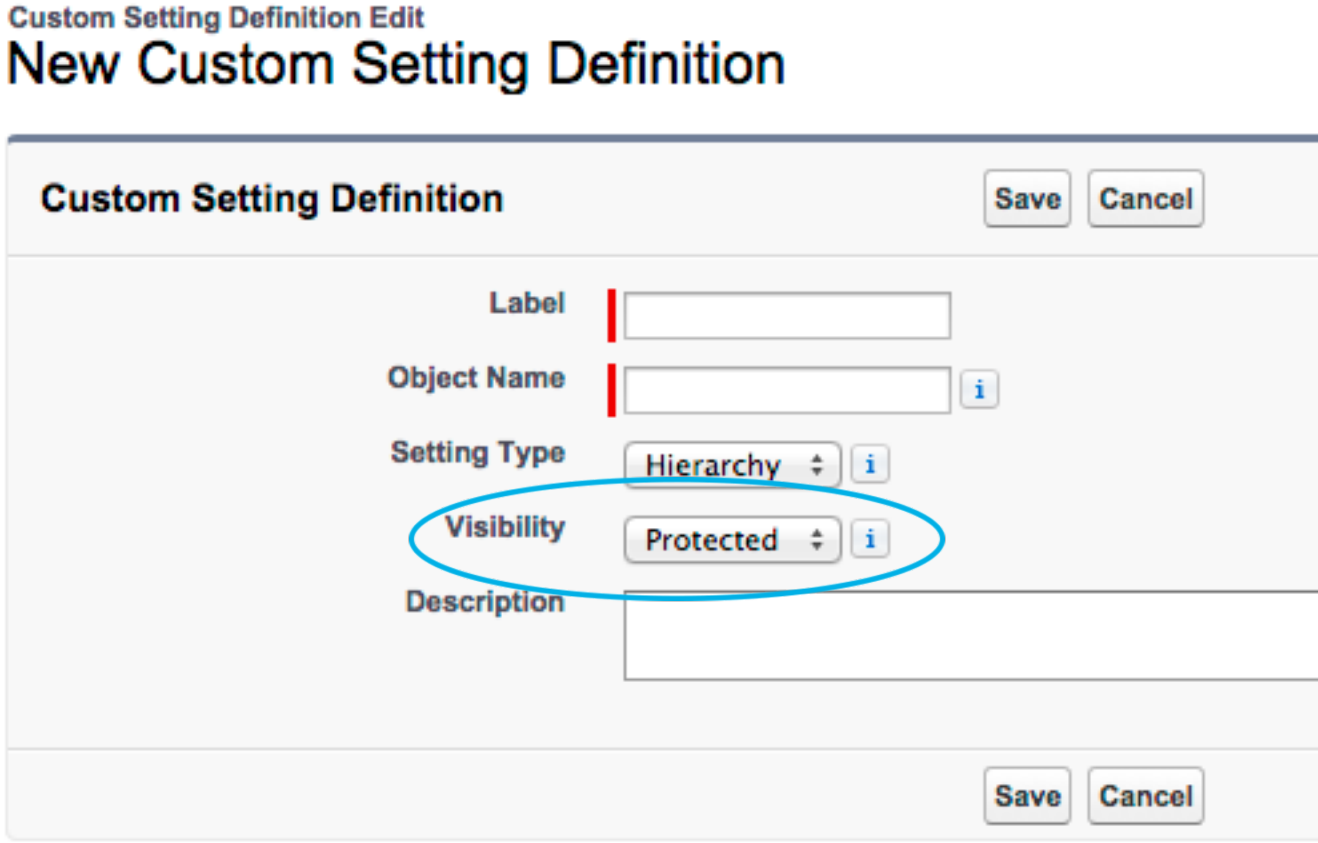
Authentication vs Authorization

- Authentication
  - Who are you?
  - Login, Session, UserContext
- Authorization
  - What are you allowed to see/do?
  - Access Control – Sharing models, profiles, user/license type, object access, field access
  - CRUD/FLS

Application logic flaws may be caused due to a misunderstanding of this distinction or when to enforce this distinction!

Secure Credential Storage

- Use Protected Custom Settings
  - Allows users to set a secret in the custom setting
  - Access the secret and perform the require operation, returning the end-result of the sensitive operation *only* back to the user.
  - Does not expose the set secret back to the user
  - This should be used to store credentials like API keys for external systems that you are integrating with.



Handling SFDC Credentials Externally

- When building an OAuth integration with Salesforce, make sure to only store the SFDC refresh token on the external site.
- Follow industry best practices for secure storage on the platform being used.
- Never store SFDC username and passwords on the external systems.
- Make sure to follow the principle of least privilege when defining OAuth scope.
- Never log SFDC access tokens on the external application.

Securing Your External Application

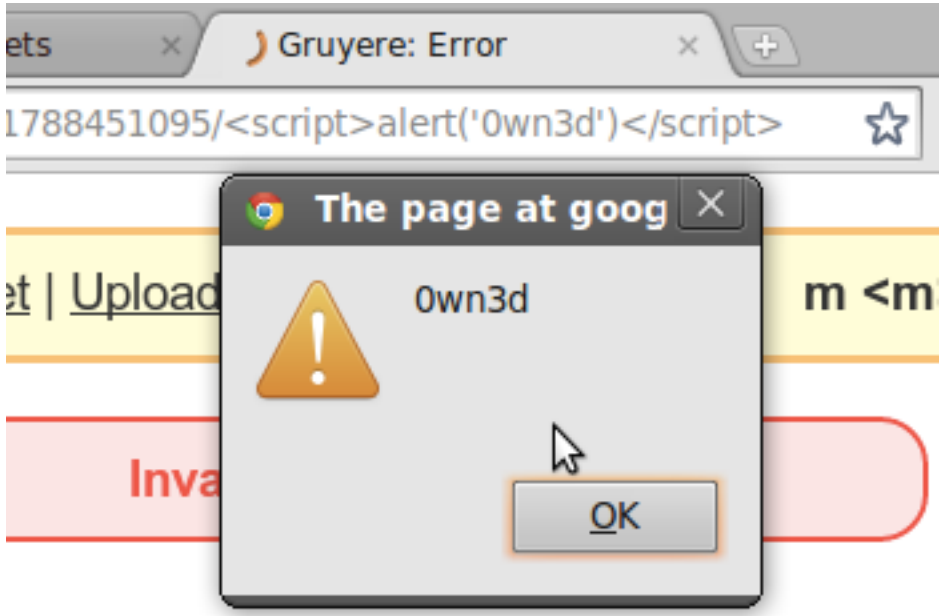
- In addition to following secure coding guidelines for the platform, you should make sure to secure your external application.
- When external integrations are built with Salesforce, we conduct an end-to-end review of the whole system.
- Web application vulnerabilities are pervasive.
- Now that your Salesforce app and data is interacting with these external applications, the security of the external applications becomes equally important.

Common Vulnerabilities

…and how to avoid them!

Cross Site Scripting (XSS)

- XSS attacks involve malicious scripts being injected into otherwise trusted web sites.
- XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a victim end user because the web application reflects user input without validating or encoding it.
- In their simplest form, XSS attacks occur when a user supplied input is directly reflected back on the web page without encoding it. It is caused due to poor separation between data and code context and user data is executed as code.



Cross Site Scripting (XSS)

- **Reflected XSS**
  - Reflected XSS occurs when a malicious input sent to the server as a part of the request is reflected back without proper encoding on the response page.

- **Stored XSS**
  - Stored XSS occurs when the user's malicious input is permanently stored on the vulnerable server. The malicious script gets executed in the victim's browser when they request the stored information from the server.

- **DOM XSS**
  - An XSS attack where the attack payload gets executed as a result of modifying the DOM environment in the victim's browser.

Cross Site Scripting (XSS)

XSS Context

```
<html>
  <head>
  </head>

  <body>
        Outwardly: dumbly, I shamble about, a thing that could never have been known as
        human, a thing whose shape is so alien a travesty that humanity becomes more obscene
        for the vague resemblance.

        <img src=”//userdefinedlocation” alt=”I think” onerror=”dothings”/>

        <script>
                name = “Nimdok“; eval(alert(name));
        </script>
  </body>
</html>
```

Cross Site Scripting (XSS)

XSS Context

```
<html>
  <head>
  </head>

  <body>
        Outwardly: dumbly, I shamble about, a thing that could never have been known as
        human, a thing whose shape is so alien a travesty that humanity becomes more obscene
        for the vague resemblance.

        <img src="//userdefinedlocation" alt="I think" onerror="dothings"/>

        <script>
                name = "Nimdok"; eval(alert(name));
        </script>
  </body>
</html>
```

Cross Site Scripting (XSS)

XSS Context

```
<html>
  <head>
  </head>

  <body>
        Outwardly dumbly, I shamble about, a thing that could never have been known as
        human, a thing whose shape is so alien a travesty that humanity becomes more obscene
        for the vague resemblance.

        <img src="//userdefinedlocation" alt="I think" onerror="dothings"/>

        <script>
                name = "Nimdok"; eval(alert(name));
        </script>
  </body>
</html>
```

DATA

CODE/CONTROL

HTML Context

HTML-URI Context

HTML-Attrib Context

JS Context

Cross Site Scripting (XSS)

Context Specific Injection

DATA | CODE/CONTROL

HTML Context

```
<body>
      Welcome <script>alert(document.cookie)</script>!
</body>
```

HTML-Attr Context

```
<img src="something" alt="" onload = "alert(document.cookie)" >
```

Javascript Context

```
<script>
      var name='bob haxor';alert(document.cookie);var hax='hah';alert('Hello '+name);
</script>
```

Cross Site Scripting (XSS)

Mitigation

- **Input filtering**
  The goal of input filtering is to constrain inputs to their expected format and to render any dangerous input harmless by removing dangerous characters. It is always safer to constrain inputs to known-good values than to try to filter dangerous characters. **Whitelist filtering** is always the preferred route for input filtering.

- **Output encoding**
  The most important defense against XSS. Output encoding refers to rewriting data such that it cannot “break out” of the structural context into which it is inserted. It thus prevents the treatment of user-supplied data as code. It tells your browser that you are trying to print the special character on the page and not execute it.

Salesforce Platform Protections

- The Force.com platform provides several defenses against XSS when writing code on the platform.
- When using standard Visualforce components to render HTML on a Visualforce page, the platform automatically encodes all the XSS-vulnerable characters in HTML context.
- User input needs to be encoded explicitly when custom javascript is written on the page. The same is true for merge fields in event handlers.
- Make sure to call the correct encoding function based on the context in which user input is reflected.
- Make sure to encode explicitly in Apex when default escaping is set to false.

Salesforce Platform Protections

```
<apex:outputText escape="false" value="Hello {!
$CurrentPage.parameters.userName}">
</apex:outputText>

var a = "{!JSENCODE($CurrentPage.parameters.userName)}";

<apex:column id="message" value="{!message}"
     onclick="displayPopup('{!message}')" />
```

XSS Prevention

- Look for `innerHTML` assignments. User input should not be assigned to an HTML element directly.(Assign innerText instead – use as data, not as code)
- Make sure dangerous coding patterns like `document.write` and `eval` are not being used.
- Look for HTML blocks being constructed via concatenation in backend controllers.
- Make sure to encode user input as close to the sink as possible. (Data might be getting reflected in multiple contexts).
- Make sure to always use security features provided by your platform as a part of the templating engine instead of building your own.
- During security review, your application will be tested with a mixture of Whitebox/ Static Analysis and Blackbox techniques

S(O)QL Injection

- S(O)QL injection is caused by naively concatenating user input with S(O)QL queries in your code
- A malicious user can exploit this vulnerability to perform arbitrary database operations and retrieve or modify data they should not have access to.

- Whenever possible, do not use dynamic queries
  - Bind variables (Platform)
  - Prepared statements (External)

- Otherwise, escape special characters in user input
  - Many database layer libraries include functions designed for preventing SQLI

- Whenever possible, use prebuilt APIs instead of trying to build your own.

Cross-Site Request Forgery (CSRF)

- Cookies are sent by the browser for all requests, including cross-domain requests.

- What this means is every time a request is made to salesforce.com from your browser, it would automatically include the cookies set for that domain.

Cross-Site Request Forgery (CSRF)

- An attacker can thus force a user to make legitimate requests with attacker supplied parameters to any site via something like this:

CSRF Prevention

- State changing web operations should be accompanied by a secret that isn't sent automatically like Cookies. These secrets are called CSRF tokens.
- The tokens should be unique per user per session. 128 bits (16 chars) of entropy.
- The server should validate the token upon each state changing request
- The Salesforce platform automatically adds CSRF protection to all VF form POST requests. Do not perform state changes on page loads (like in init functions).
- Custom headers with secrets are also an effective way to prevent CSRF because they are not automatically sent by your browser.

Testing for CSRF

- Identify state changing operations in a feature
- Intercept operation using a proxy like Burp or ZAP
- Identify the CSRF token in the request (Parameter or a Custom Header)
- Repeat the request without the token and the operation should fail on proper implementation
- Alter the value of the token and the operation should fail on proper implementation
- Re-using user A's CSRF token (by editing the request) on a different session(A terminates session and creates a new session Or user B's session) should result in failure

Remote Code Execution (RCE)

- Protect file upload functionality
  - User-uploaded content should be served from a different domain
  - File types & formats (extension or MIME) should be verified server-side
  - Upload directory should not be web-accessible or indexable
  - Files uploaded by a user should not have execution permissions
  - “Never trust user input” applies to user uploaded files as well

- Do not accept user input for file inclusion or import
  - User may be able to run an otherwise “safe” uploaded file by making the system think it is a necessary resource file
  - Malicious user may also be able to dump server and/or source code files via this attack

Thank You

Secure Development Sessions

Secure Coding: Field-level Security, CRUD, and Sharing
Monday, October 13 @ 11:00 a.m. - 11:40 a.m.

Secure Coding: Storing Secrets in Your Salesforce Instance
Monday, October 13 @ 2:00 p.m. - 2:40 p.m.

Building Secure Mobile Apps
Monday, October 13 @ 5:00 p.m. - 5:40 p.m.

Protect Your Data Against Malicious Scripts
Tuesday, October 14 @ 11:00 a.m. - 11:40 a.m.

Secure Coding: External App Integration
Wednesday, October 15 @ 9:00 a.m. - 9:40 a.m.

Secure Coding: SSL, SOAP, and REST
Thursday, October 16 @ 10:30 a.m. - 11:10 a.m.

dreamforce

Announcements:

Force.com Code Scanner now supports Salesforce1 and JavaScript! Try it here: http://bit.ly/SF1Scanner

Chimera Web App Scanner alpha nominations are open. Partners apply at: http://bit.ly/SFChimera

Live security office hours are available in the partner zone.

salesforce

Additional Resources

- Secure Coding Guidelines - https://developer.salesforce.com/page/Secure_Coding_Storing_Secrets
- Salesforce StackExchange - http://salesforce.stackexchange.com/questions/tagged/security
- Developer.Salesforce.com Security Forum - https://developer.salesforce.com/forums (full link hidden)
- Salesforce Security resources - https://developer.salesforce.com/page/Security
- Security Office Hours (Partners) - http://security.force.com/security/contact/ohours
- OWASP Top Ten - https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project