Ajax Security in Groupware

SEAA 2006, Cavtat, 29.8. – 1.9.2006

Institute for Information Processing and Microprocessor Technology (FIM)
Johannes Kepler University Linz, Austria

E-Mail: sonntag@fim.uni-linz.ac.at
http://www.fim.uni-linz.ac.at/staff/sonntag.htm
Introduction

- What is "Ajax"?
  - Promises for groupware
- Inherent security issues
  - Client-side data verification, cross-domain integration etc.
- Exacerbated security issues
  - Communication security, code injection, client-side code, ...
- Possible solutions
  - Implementation and design approaches
- Conclusions
  - Open problems
  - Recommendations
What is "Ajax"?

- **Ajax** = Asynchronous JavaScript and XML
  - Not a new technology, but new hype (?)
- **Basic idea**: Make the web more like an application
  - No form submit & page reload, but working on a single page
- **Implementation**:
  - Normal webpage enhanced with JavaScript
  - On some event, JavaScript generates an XML request
  - The request is sent to the server the page originates from
  - Answer calculated by server and sent back in XML
  - JavaScript decodes the response and displays it
- **Modifications and alternatives**:
  - XML replaced by other format, e.g. JSON or plain text
  - Applets, Flash, …
    - Locally installed plugins
Promises of Ajax for groupware

• Special advantages for groupware
  → No need for installing local applications (easy "distribution")
  → Both outgoing and incoming information possible at any time without interrupting the user
    » More timely alerts on actions by others
    » No need for submitting a webpage to send information
  → Reduced response times: Only sent what is exactly needed

• Mash-ups: Content from several servers on a single page
  → Integration of infos and interaction from several sources
    » Only for display and separate usage
    » Actual data exchange or interaction between the platforms still requires additional work; easier with Ajax (on same page!)
  → Actions in one platform can be used for notifications or personalization in all participating systems
Inherent security issues

- **Asynchronicity:** No longer 1 user = 1 server thread
  - Requests might be outdated when actually served
  - Responses might be sent to the wrong page
    - Result of long calculation; user changed static (=quick) page
  - JavaScript must cope with ANY information sent to it!

- **Server-side data verification:** Never trust the client
  - Typical task: Client-side form validation with server help
  - Server **must** revalidate everything (= validate twice)!
    - Groupware can use the intermediate verification for notifications!

- **Session tracking:** All Ajax calls should be within a session
  - Session ID (hidden field, link parameter) must be encoded by JavaScript and checked on server
    - Even when the Ajax call is session-independent
    - Otherwise they become public services anyone can use!
Inherent security issues

- Cross-domain integration (mash-ups):
  - All data must go through a single server: JavaScript cannot contact servers other than the one the page was loaded from
    - At least in theory: Numerous bugs exist which allow this!
      - Result: Any information from one of the participating servers can be sent to an arbitrary other server!

- Page validation & testing: Checking e.g. for WAI-conformity
  - How to check webpages changed dynamically on the client?
    - What if markup (not just data) is sent by the server?
  - Typically browsers allow no access to the "compound" page, only to the in-memory object representation (DOM, …)
  - Testing more difficult: Simulating user input? Verifying the response? DOM modification/inspection?

- Similar to distributed computing: More difficult to program
  - More bugs to be expected!
Exacerbated security issues

- Communication security: Server sends code executed on clients
  - Man-in-the-middle-attacks very attractive
  - Intermediate values are sent (e.g. password in wrong field)
- Multiple entry points: Each Ajax call is a separate script
  - Framework or strict security guidelines needed
  - Potential programming and maintenance problem
    » But: Each function clearly separated and smaller
- Code injection: Server can send anything
  - Clients cannot verify the code sent from the server
  - Problems with mash-ups: Code from third party servers cannot even be verified by the main (=proxy) server!
- Client-side code: JavaScript is very "bug-prone"
  - No type checking, weak objects, etc.!
- Program code accessible on client
**Possible solutions: Implementation**

- **Server-side data verification**
  - Revalidate everything validated through Ajax on the client
  - Verify the state: Is this Ajax call allowed for the page, the client currently is on?
    - Requires extended state infos on server, e.g. unique tokens for each "main" page, which must be passed back with Ajax calls!

- **Client-side input verification**
  - Clients should verify data from server too, if possible
    - But: Code cannot be verified
    - The verification code itself comes from the server too…
      - See communication security!
      - Therefore pointless against "attacks", only helps with errors

- **Timeouts on the client**
  - Avoid lock-ups on server problems
  - Reduces the danger of inconsistencies between Ajax calls
Possible solutions: Implementation

- Proxy server must inspect information passed on
  - When assembling a page from different sources, only pass on data, but never code, and inspect in thoroughly
    » Do not pass on encrypted or obfuscated code or data
    » Some code will be necessary: verify it and then host it locally!
  - Investigate interdependencies between services
    » They are running within the same security context on the client!

- Use an encrypted communication channel
  - Includes secure comm. with "base servers" for mash-ups!
  - Prevents tampering of (dynamically sent!) code
    » Encryption through JavaScript code on client is NOT secure!
    » Ensures that Ajax calls are encrypted too
  - Requires server verification: Encryption alone is insufficient
    » Depending on program "distribution", clients might need identification and verification too
Possible solutions: Design

- Restrict data and invocation models for Ajax
  - Use the GET model: Retrieve information from the server by Ajax, but don't change the server's state through it
    » Avoids problems because of asynchronicity!
    » Receiving notices: Server sends updates to display
      – No influence on actual "behaviour", e.g. form contents!
  - Sending notices as one-way only
    » The client informs the server, but doesn't expect an answer
      – Awareness features in groupware!
  - I.e. avoid "data loops"
    » Client 1 → Server → Client 1 or in reverse (S → C → S)
    » Careful with mash-ups: Loops over third-party servers possible!
    » Less problematic if the user is within the loop
      – But problems possible even then
Possible solutions: Design

- Keep the business logic on the server
  - Ajax: Transfer some (or all) logic to the client
    - This cannot be any business critical or security related logic!
      - The server cannot verify the result unless it recreates it
  - All the "base" data must be sent back and the server then calculates any results or consequences from it (again)
    - Danger of inconsistencies between what users see
      - Calculated by JavaScript
    - and what happens on the server!
      - Calculated by a different program in PHP, C++, ...
  - Alternative: The server trust the clients completely
    - Secure client identification in SSL through mutual authentication
Conclusions

- **Open problems:**
  - Page validation and testing
    - Test all parts separately
    - Use framework and strict guidelines for assembly
  - Code accessibility
    - Cannot really be avoided
    - Make sure no secret information is ever sent to the client!
  - Legal responsibility: Proxying has legal consequences….
    - Regarding both licenses and liability!

- **Recommendations:**
  - Ajax can greatly enhance the UI and its responsiveness
    - But use it for this **only**, not for distributing the program
  - Make sure the connection is secured
  - Drop the "A" in "Ajax" if possible: Avoid asynchronicity
    - Programming harder; additional problems, e.g. synchronization
Questions?

Thank you for your attention!