International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Series X
Supplement 21
(01/2014)

SERIES X: DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY

ITU-T X.1143 – Supplement on security framework for web mashup services

ITU-T X-series Recommendations – Supplement 21

ITU-T

ITU-T X-SERIES RECOMMENDATIONS

DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY

PUBLIC DATA NETWORKS	X.1–X.199
OPEN SYSTEMS INTERCONNECTION	X.200–X.299
INTERWORKING BETWEEN NETWORKS	X.300–X.399
MESSAGE HANDLING SYSTEMS	X.400–X.499
DIRECTORY	X.500–X.599
OSI NETWORKING AND SYSTEM ASPECTS	X.600–X.699
OSI MANAGEMENT	X.700–X.799
SECURITY	X.800–X.849
OSI APPLICATIONS	X.850–X.899
OPEN DISTRIBUTED PROCESSING	X.900–X.999
INFORMATION AND NETWORK SECURITY	
General security aspects	X.1000–X.1029
Network security	X.1030–X.1049
Security management	X.1050–X.1069
Telebiometrics	X.1080–X.1099
SECURE APPLICATIONS AND SERVICES	
Multicast security	X.1100–X.1109
Home network security	X.1110–X.1119
Mobile security	X.1120–X.1139
Web security	X.1140–X.1149
Security protocols	X.1150–X.1159
Peer-to-peer security	X.1160–X.1169
Networked ID security	X.1170–X.1179
IPTV security	X.1180–X.1199
CYBERSPACE SECURITY	
Cybersecurity	X.1200–X.1229
Countering spam	X.1230–X.1249
Identity management	X.1250–X.1279
SECURE APPLICATIONS AND SERVICES	
Emergency communications	X.1300–X.1309
Ubiquitous sensor network security	X.1310–X.1339
CYBERSECURITY INFORMATION EXCHANGE	
Overview of cybersecurity	X.1500–X.1519
Vulnerability/state exchange	X.1520–X.1539
Event/incident/heuristics exchange	X.1540–X.1549
Exchange of policies	X.1550–X.1559
Heuristics and information request	X.1560–X.1569
Identification and discovery	X.1570–X.1579
Assured exchange	X.1580–X.1589
CLOUD COMPUTING SECURITY	
Overview of cloud computing security	X.1600–X.1601
Cloud computing security design	X.1602–X.1639
Cloud computing security best practices and guidelines	X.1640–X.1659
Cloud computing security implementation	X.1660–X.1679
Other cloud computing security	X.1680–X.1699

For further details, please refer to the list of ITU-T Recommendations.

Supplement 21 to ITU-T X-series Recommendations

ITU-T X.1143 – Supplement on security framework for web mashup services

Summary

Supplement 21 to the ITU-T X-series Recommendations describes the security framework for web mashup services and also describes web mashup types and a reference architecture. Security principles and measures for secure web mashup services are provided for mitigating security threats and addressing security challenges for the web mashup services.

History

Edition	Recommendation	Approval	Study Group	Unique ID[*]
1.0	ITU-T X Suppl. 21	2014-01-24	17	11.1002/1000/12155

[*] To access the Recommendation, type the URL http://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID. For example, http://handle.itu.int/11.1002/1000/11830-en.

Table of Contents

Supplement 21 to ITU-T X-series Recommendations

ITU-T X.1143 – Supplement on security framework for web mashup services

1 Scope

This Supplement addresses the security framework for web mashup services including the following items:

– Overview of mashup web services;

– security principles and measures for secure web mashup.

2 References

None.

3 Definitions

3.1 Terms defined elsewhere

This Supplement uses the following terms defined elsewhere:

3.1.1 access control [b-ITU-T X.800]: The prevention of unauthorized use of a resource, including the prevention of use of a resource in an unauthorized manner.

3.1.2 authorization [b-ITU-T X.800]: The granting of rights, which includes the granting of access based on access rights.

3.1.3 availability [b-ITU-T X.800]: The property of being accessible and useable upon demand by an authorized entity.

3.1.4 confidentiality [b-ITU-T X.800]: The property that information is not made available or disclosed to unauthorized individuals, entities, or processes.

3.1.5 data integrity [b-ITU-T X.800]: The property that data has not been altered or destroyed in an unauthorized manner.

3.1.6 data origin authentication [b-ITU-T X.800]: The corroboration that the source of data received is as claimed.

3.1.7 hyper text markup language (HTML) [b-ITU-T M.3030]: A system of coding information from a wide range of domains (e.g. text, graphics, database query results) for display by World Wide Web browsers. Certain special codes, called tags, are embedded in the document so that the browser can be told how to render the information.

3.1.8 origin [b-IETF RFC 6454]: The origin of a URI is the value computed by the algorithm of RFC 6454's section 4. Two URIs are part of the same origin if they have the same scheme, host, and port.

3.1.9 repudiation [b-ITU-T X.800]: Denial by one of the entities involved in a communication of having participated in all or part of the communication.

3.1.10 privacy [b-ITU-T X.800]: The right of individuals to control or influence what information related to them may be collected and stored and by whom and to whom that information may be disclosed.

3.2 Terms defined in this supplement

This Supplement defines the following terms:

3.2.1 authentication: A process used to achieve sufficient confidence in the binding between the entity and the presented identity.

NOTE – Use of the term authentication in a web-based service context is taken to mean entity authentication.

3.2.2 javascript object notation (JSON): A lightweight, text-based, language-independent data interchange format.

3.2.3 mashup: A web application that combines content (data and code) or services from multiple origins to create a new service.

3.2.4 screen scraping: Screen scraping is the use of manual or automatic means to harvest content from a website.

NOTE – Under normal circumstances, a legacy application is either replaced by a new program or brought up to date by rewriting the source code. In some cases, it is desirable to continue using a legacy application but the lack of availability of source code, programmers or documentation makes it impossible to rewrite or update the application. In such a case, the only way to continue using the legacy application may be to write screen scraping software to translate it into a more up-to-date user interface.

3.2.5 web 2.0: Web technology and applications that facilitate participatory information sharing, interoperability, user-centred design and collaboration on the world wide web (WWW).

4 Abbreviations and acronyms

This Supplement uses the following abbreviations and acronyms:

AJAX/Ajax	Asynchronous Javascript and XML
API	Application Programming Interface
CSRF	Cross-Site Request Forgery
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
iframe	Inline Frame
JSON	Javascript Object Notation
JSON-RPC	Javascript Object Notation-Remote Procedure Call
KML	Keyhole Markup Language
PC	Personal Computer
REST	Representational State Transfer
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
SOP	Same-Origin Policy
SQL	Structured Query Language
UI	User Interface
URI	Uniform Resource Identifier

WWW	World Wide Web
XHR	XMLHttpRequest
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
XML-RPC	Extensible Markup Language-Remote Procedure Call
XSS	Cross-Site Scripting

5 Conventions

None.

6 Overview of web mashup services

6.1 Web mashup types and style

In web 2.0, composite services are called mashups. A mashup is a web application that combines data or functionality from two or more sources to create new services. Data used in mashups is typically sourced from a third party via a public interface, an application programming interface (API) and screen scraping. The main characteristics of a mashup are combination, visualization and aggregation to make existing data more useful for personal and professional use. This means that mashup technically provides sharing public data, common user interface (UI) to data and new, interesting and valuable, data by aggregation.

There are many types of mashups, such as presentation mashup, client-side data mashup, client-side software mashup, server-side software mashup and server-side data mashup. Figure 1 shows the mashup types.

- The presentation mashup is where information and layout is retrieved from and either remixed or just placed next to each other.
- The client-side data mashup takes information from remote web services, feeds or even just plain Hypertext Markup Language (HTML) and combines it with data from another source.
- The client-side software mashup is where a code is integrated in the browser to result in a distinct new capability.
- The server-side software mashup is where software is recombined on the server since web services can use more easily other web services where there are less security restrictions and cross-domain issues.
- The server-side data mashup uses relatively powerful mechanisms to join or mashup data from databases on the server side.

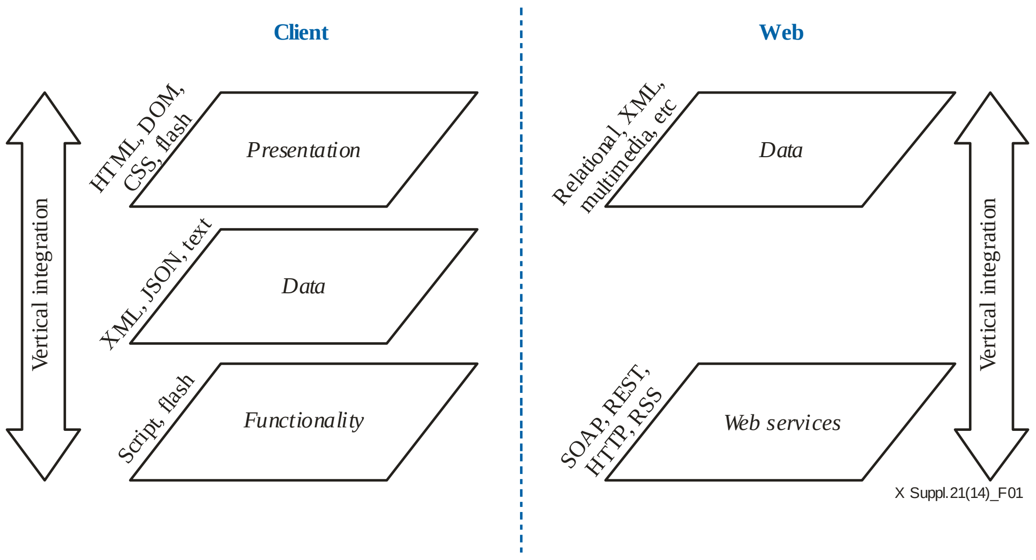

Figure 1 – Web mashup types

The structure of a mashup is divided into three layers:

- Presentation/user interaction: This is the user interface of mashups. The technologies which are used are HTML/Extensible Hypertext Markup Language (XHTML), cascading style sheets (CSS), script, asynchronous javascript and XML (Ajax) [b-W3C CSS], [b-AJAX].
- Web services: The products functionality can be accessed using the API services. The technologies used are XMLHttpRequest (XHR), Extensible Markup Language (XML) – remote procedure call (RPC), javascript object notation (JSON)-RPC, simple object access protocol (SOAP) and representational state transfer (REST) [b-W3C SOAP], [b-REST].
- Data: Handling the data such as sending, storing and receiving. The technologies used are XML, JSON and Keyhole Markup Language (KML) [b-W3C XML], [b-JSON], [b-OGC KML].

Web mashup security is based on the same-origin policy (SOP). The SOP states that scripts from an origin should not be able to access content from other origins. This prevents scripts from spoofing data, cookie credentials from other origins. According to SOP, loading components from different origins causes them to be separated. Because of these mechanisms, there are two styles of mashups: web-based and server-based. Whereas web-based mashups typically use the user's web browser to combine and reformat the data, server-based mashups analyse and reformat the data on a remote server and transmit the data to the user's browser in its final form.

Figure 2 shows styles of mashup application.

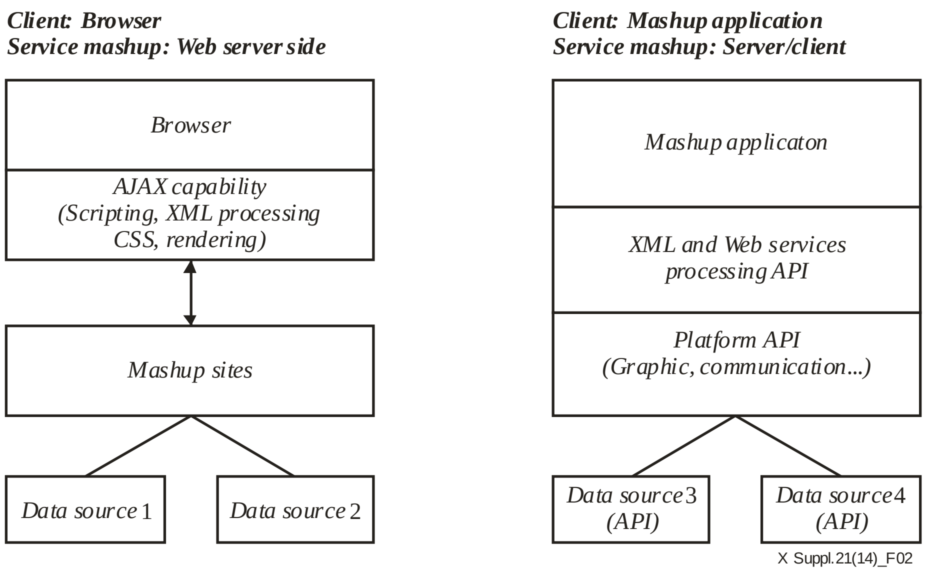

Figure 2 – Styles of mashup application

6.2 Web mashup reference architecture

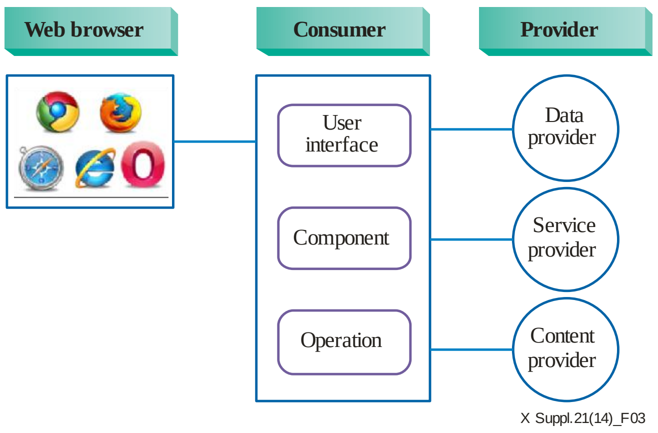

Figure 3 – Mashup reference architecture

Before the emergence of mashup services, a user could access and obtain the result from the only single service provider. As users' requests have become more complicated, the convergence of services is needed. However, the same-origin policy (SOP) prevents access to most methods and properties across the different websites. Web mashup developers have to overcome this SOP policy to merge the data and the operations from different sites. The mashup service architecture provides the consumer with the means to handle such requests. The consumer can control and resolve access to the data of the other sites. In addition, the consumer is supposed to be dealing with other security services. Figure 3 shows the web mashup reference architecture which is comprised of the web browser, the consumer and the service provider.

6.2.1 Web browser

End users access the web mashup services through the web browsers in their personal computers (PCs) or mobile phones.

6.2.2 Consumer

The mashup services are hosted in the mashup consumer. The mashup consumer can be implemented according to the style of mashup applications: web-based and server-based. The mashup consumer usually provides mashup services such as mixing data, managing access control and communication between consumers and providers. There are three sublayers in a mashup consumer.

6.2.2.1 User interface sublayer

End users may gather different mashup components such as widgets-a software application comprising portable code intended for one or more different software platforms into the presentation layer of web mashup browser. Various mashup components could be integrated into a new mashup web service.

6.2.2.2 Component sublayer

This sublayer mainly classifies and binds related mashup data and functions into a specific component. The name and composition of components depend on the programming languages and mashup consumers.

6.2.2.3 Operation sublayer

In the mashup operation sublayer, developers query the data and do some data manipulation such as data aggregation, data intersection, data cache, etc. Designers may also abstract the mashup APIs used by the mashup consumer.

6.2.3 Provider

The mashup data and services come from content/API providers. The dynamic mashup data may come from devices such as PCs, mobile phones, etc.

7 Security architecture of web mashup service

Early development efforts for the web were directed towards presenting static HTML pages to a browser client. This simplified model limited the types of interactions offered to a user. It was not long before developers and businesses realized that a more dynamic model of interaction was possible and desired by users of the websites.

A dynamic interaction with website users started with the introduction of server-side scripting languages that could create custom pages based on the input from a given user or input resulting from changes in the business data. At the present time, users could interface with a website in an interactive manner. This interactive means of interchange between the clients and the servers garnered an exponentially significant amount of momentum in a very short time. This chaotic development environment created a breeding ground for security vulnerabilities and holes. The era of mashups has emerged and is creating another round of sidesteps and hacks that are leading to more security problems.

A mashup illustrates the manner in which security vulnerabilities can multiply quickly. The wide open integration possibilities make it imperative to ensure that the data and functionality are not open to hacker attempts and other forms of intrusion. The intrinsic openness of a mashup environment and the inability to predict exactly how components of a mashup infrastructure will be used in the future imply the need to address security at every aspect of the development life cycle. A mashup environment uses components and UI artefacts developed externally. This means that the external components should be checked and aggregated with other components of a given mashup.

Mashups involve a man-in-the-middle problem originally. While web services using SOAP as a transport can provide end-to-end security services, typical web 2.0 applications use the simpler REST-based communication approach that seems more vulnerable. In particular, web clients (browsers) do not typically implement web services security. As a result, best practice is to delegate full rights to the mashup server (man-in-the-middle) and hope that the user's rights to data and services are not abused. This entails the end user providing the mashup server with their security credentials for the back-end services in a way whereby they can be exploited.

In Figure 4, end-client mixes the contents from different domains involving its own domain. Through APIs, the end user can aggregate the news feed server and load content from blog that are not ensured. The end-client just uses those contents but does not know whether the contents from the third-parties are sanitized or not. Because of this vulnerability, mashup behaviours and operations should be monitored and ensured.

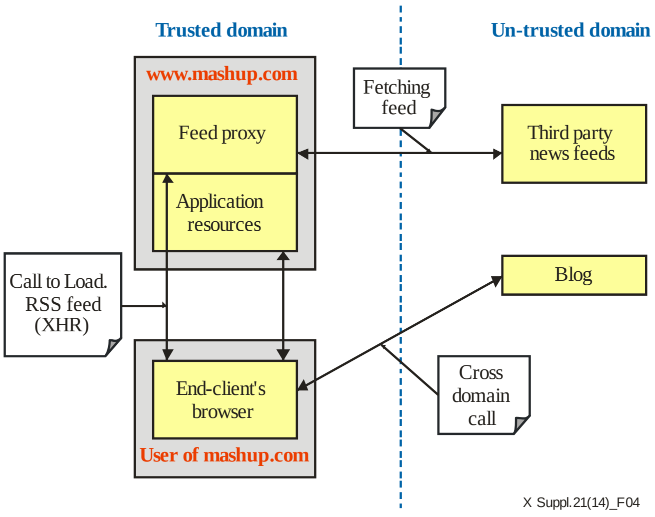

Figure 4 – Vulnerability structure of web mashup services

7.1 Web mashup security principles

A web mashup is a web application that integrates contents from different providers to create a new service. Cross-origin interaction within the browser is currently regulated by the so-called same-origin policy (SOP). SOP classifies documents based on their origins. Documents from the same origin may freely access each other's content, while such access is disallowed for documents of different origins.

Unfortunately, the SOP mechanism turns out to be problematic for mashup security. First, origin tracking in SOP is only partial and allows content from different sources to coexist under the same origin. For example, an HTML tag with a source attribute can load content from some other origin and integrate it in the current document. Once integrated, such content is considered to be of the same origin as the integrating document. This means that the content is accessible to scripts in other documents from the same origin.

Of particular concern is the document inclusion via script tags. When a script tag is used to load script code from a different origin, the loaded script is integrated into the document and thereby can freely interact with it. For the same reasons, interaction between different components loaded in this fashion is unrestricted.

The problem of script-tag inclusion for mashup applications is that the integrator must trust the third parties to protect its secrets and not to override the trusted data with the untrusted. The security of the integrator no longer relies only upon itself, but also on the security of the third parties whose scripts are included.

So far, these issues have been resolved using the inline frame (iframe) tag. The iframe tag borrows a part of the integrator's window space to display another document. Since the integrated content is loaded in a separate document, the SOP applies, and the sensitive information of the integrator is protected.

However, this also severely reduces the possibilities for interaction between the documents. A number of techniques for secure communication between documents have been proposed to bypass the restrictions, but, due to script's dynamic nature, ensuring confidentiality has proved to be complicated.

Because web mashups mix content and services from several domains into an application, there are many stakeholders such as users, web developers, mashup programmers and mashup composers. These stakeholders raise security and technical requirements on mashups. First, content from different components needs to be separated, but secure and controlled interaction is needed. Further, the composer wants to integrate a mix of these components easily. The separation, interaction techniques suffer from restrictions imposed by the same-origin policy. These limitations have driven the development of new techniques. Cross-domain communication has found its way. Cross-origin resource sharing allows controlled cross-domain interactions. These kinds of separation and interactions for mashups have led to four security requirement principles [b-SECMASH]:

– Separation of components: Components need to be separated from each other to ensure the following security properties:
  • DOM: Ensures that the component's part of the document object model (DOM) tree [b-W3C DOM tree] is separated from the other components.
  • Script: Ensures that the component's scripts cannot be influenced by other components.
  • Applicable in the same domain: Ensures that the separation techniques can also be applied to different components belonging to the same domain.
– Interaction: Regardless of their separation, a component requires interaction with other components and the host page. This interaction is subject to the following requirements:
  • Confidentiality: Ensures that sensitive information cannot be stolen from interactions between components.
  • Integrity: Ensures that the contents of an interaction cannot be modified without the knowledge of the interacting components.
  • Mutual authentication: Ensures that the interacting components can establish with whom they are interacting.
– Communication: Components need to be able to communicate with the mashup provider as well as with other parties. This requires the following properties:
  • Cross-domain: Components should be able to communicate with other origins than the origin to which they belong.
  • Authentication: A service receiving messages should be able to identify the origin of the message.
– Behavioural control: Control over the specific behaviour of the components is needed to selectively allow or disallow specific functionality.

7.2 Measures for secure web mashup services

Once organizations have established firm and effective policies and promoted the use of these standards within their systems, implementation details for securing mashup components and processes must be addressed [b-MASHUPS]. The following clauses show implementation specifics for securing the principal parts of a mashup infrastructure.

Filtering input data

Many intrusion vulnerabilities such as structured query language (SQL) injection, cross-site request forgery (CSRF) and cross-site scripting (XSS) [b-ITU-T X-Supp.17] are recommended to be prevented with an input filtering technique. Filtering input data is the foundation for securing a mashup application. A mashup server-side filtering technique and a client-side mashup filtering one could complement each other in the manner they process input data. Because client-side filtering process can be circumvented quite easily, a comprehensive and complementary server-side filtering provides another crucial component for protecting data and processes.

For filtering input data effectively, the following items are recommended:

- Define a list of finite values to which input data should be constrained;
- Validate input data types, data lengths, data ranges and data formats;
- Use regular expressions at the client and at the server to facilitate a consistent validation model;
- Sanitize input data for invalid characters.

Precaution against cross-site request forgery

The same-origin policy does not prevent requests from a third-party but it only prevents requests to a third-party. Therefore, the same-origin policy does not protect against cross-site request attacks. Most authentication mechanisms including cookies, username/password and certificates are vulnerable to CSRF attacks since each mechanism authenticates session not among browsers and servers but between a browser and the server.

The mashups are recommended to confirm that the mashup page is authenticated and allows each request to be performed. The mashup assumes a request is a normal authenticated request from the mashup page and performs the process as usual. The response is then transmitted unknowingly to the third-party site.

Defending on-demand script

A <script> tag has its accompanying script source that is embedded in an HTML page. One reason for using on-demand script is to bypass the same-origin policy and retrieve content from multiple sites. This mechanism is typically exploited by mashups by retrieving <script> snippets from a server after the page has been loaded.

On-demand script is often employed using AJAX and calls to a server via the XHR object. A response from the server can be formatted as a script. When the browser receives the response, it evaluates it and the script is executed. Any actions specified in the script affecting UI components are seen as the script is executed and the DOM is manipulated.

On-demand script has some obvious security vulnerabilities. Mainly, since the same-origin policy is bypassed and embedded scripts are executed as they are encountered, malicious code from external domains have a dangerous degree of access to data and processes available to the page in which the scripts are embedded. Specifically, scripts from external sites can access cookies associated with the hosting page, and scripts are executed immediately as they are evaluated and there is no chance to validate the scripts for potential security threats. Defending against on-demand security vulnerabilities involves constraining on-demand script to a hidden iframe. The hidden iframe communicates with the main page to alter UI components on the page. In this manner, scripts can be parsed and evaluated prior to the execution, thereby allowing a mashup to validate the script before execution.

Defending iframes

An inline frame places another HTML document in a frame. Unlike an object element, an inline frame can be the "target" frame for links defined by other elements and it can be selected by the user agent as the focus for printing, viewing its source, and so on. iframes is a good technique for isolating potentially untrusted content within a browser page, since content placed inside an iframe cannot manipulate the DOM or other browser components residing outside the iframe. However, iframes can be hidden and often are hidden to use as communication channels within a browser document. When a main document loads and evaluates iframe, the hidden iframe can retrieve the data following the fragment iframes or between containing documents and child iframes. There is a security vulnerability using the "srcURL" data-passing mechanism. If content snippet is embedded in the page from an external site and the snippet contains malicious code, the iframe can be compromised.

To defend iframe fragment-identifier data passing attack, the following is recommended:

– Verify the domain modifying fragment-identifiers to ensure that data is accepted from the white-listed domains;
– Filter script embedded in the fragment-identifier data;
– Filter embedded iframes in the fragment-identifier data.

Parsing JSON data

JSON appropriately indicates that JSON data is actually an integral part of the script programming language. This means that JSON data can be used, as is, in a script function or statement. The *eval()* function can be used to evaluate/interpret JSON data.

When the JSON data is interpreted, any valid script instructions embedded in the JSON data are executed. This mechanism is useful for receiving data responses from a server using the XHR object and used in a mashup page. However, this mechanism also presents some significant security vulnerabilities.

When JSON data is dynamically loaded as with an XHR response, it can be easily interpreted and converted into standard script. Any executable script embedded within the JSON data is executed immediately as it is interpreted script's *eval()* function is a common mechanism used to interpret JSON data dynamically. If the data is retrieved from an attacker site and contains a malicious script, sensitive data can be stolen and used, and the attacker can execute any code within the mashup page. Proper parsing of the JSON data on the client is recommended to resolve these embedded holes in the JSON data.

Authentication and authorization

Authentication and authorization are complex issues in a mashup environment since many requests can be transmitted to several different services, many of which may require authentication. The basic communication pattern for mashups has a client authenticating to a mashup server, which in turn authenticates to one or more data sources. When the mashup server and data sources are in the same security domain, the mashup server can reuse the authentication credentials to authenticate the data source. The result is unrestricted delegation.

OAuth provides a method for clients to access server resources on behalf of a resource owner. It also provides a process for the end-users to authorize third-party access to their server resources without sharing their credentials (typically, a username and password pair) using user-agent redirections [b-IETF RFC 5849], [b-IETF RFC 6749]. Although OAuth provides a good start, this relies on many browser redirects which make it more prone to phishing attack. It is recommended to specify a framework for rights delegation, identity management, authentication and user interaction.

Security policies

To achieve cross-domain communication, cross-domain resource sharing (CORS) is designed to extend the SOP to allow safe, controlled cross-domain communication [b-SECMASH]. The CORS standard works by adding new Hypertext Markup Language (HTTP) headers that allow servers to serve resources to the permitted origin domains [b-W3C CORS]. CORS allows a remote server to indicate whether the given origin has access to its resources or not, a decision which is enforced by the browser. CORS is not an answer for every cross-domain call. For instance, if a user wants to build a feed reader and access the feeds on different domains and the servers will not implement CORS, so the user will need to build a proxy to provide this. Also, CORS cannot point out the exact component within the same origin.

A fine-grained control over component behaviour in a mashup is recommended for the communication and behaviour control among cross origins. The enforcement of fine-grained security polices for script in a mashup browser is recommended to control the script behaviour [b-SEFGSP], [b-OVFGS], [b-LBA].

The measures of fine-grained security policies for script are recommended as follows:

- Control behaviour of script to execute a function and access data;
- Prevent unauthorized leaking between origins;
- Mediate access over shared objects in script environment;
- Evaluate an authentication/authorization decision on the contents.

The fine-grained security policies also include the traditional security policy functions (basic access controls [allow, deny, inapplicable and indeterminate], support distributed policies and domain independent) and the above measures are additional and focused on controlling the script behaviours and cross – origin communications. The last measure is recommended to protect the content because a web mashup browser gets the content by API after getting permission from the policy server to access the content. At that time there is a problem that the browser simultaneously gets the naive content and permission data (permitted or not permitted) even in case of negative authorization. And then the browser takes decision on the content to access it or not. The naive content is still there in the platform. Web video is presently popular in the World Wide Web (WWW) environment. The contents like file, code, streaming, and video are delivered to the end-user/browser and anyone can re-deliver these to others, there is no restriction to handle those in the web mechanisms. The last measure is a complementary improvement to mitigate unlimited replication and intellectual property rights (IPR) infringement. It includes user authentication/authorization to the mixed content among origins and performs the content encryption/decryption [b-W3C EME].

Bibliography

[b-ITU-T M.3030]	Recommendation ITU-T M.3030 (2002), *Telecommunications Markup Language (tML) framework.*
[b-ITU-T X.800]	Recommendation ITU-T X.800 (1991), *Security architecture for Open Systems Interconnection for CCITT Applications.*
[b-ITU-T X-Supp.17]	ITU-T X-series Recommendations – Supplement 17 (2012), *ITU-T X.1143 – Supplement on threats and security objectives for enhanced web-based telecommunication services.*
[b-IETF RFC 5849]	IETF RFC 5849 (2010), *The OAuth 1.0 Protocol* <http://www.ietf.org/rfc/rfc5849.txt>
[b-IETF RFC 6454]	IETF RFC 6454 (2011), *The Web Origin Concept* <http://www.ietf.org/rfc/rfc6454.txt>
[b-IETF RFC 6749]	IETF RFC 6749 (2012), *The OAuth 2.0 Authorization Framework* <http://www.ietf.org/rfc/rfc6749.txt>
[b-OGC KML]	OGC standard, *Keyhole Markup Language (KML)* <http://www.opengeospatial.org/standards/kml>
[b-W3C CORS]	W3C Proposed Recommendation (2013), *Cross-Origin Resource Sharing* <http://www.w3.org/TR/cors/>
[b-W3C CSS]	W3C Cascading Style Sheets (home page) <http://www.w3.org/Style/CSS/>
[b-W3C DOM tree]	Document Object Model (DOM) tree <http://www.w3schools.com/js/js_htmldom.asp>
[b-W3C EME]	W3C Editor's Draft (2013), *Encrypted Media Extensions* <https://dvcs.w3.org/hg/html-media/raw-file/tip/encrypted-media/encrypted-media.html>
[b-W3C SOAP]	W3C (2007), *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition).* <http://www.w3.org/TR/soap12-part1/>
[b-W3C XML]	W3C Recommendation (26 November 2008), *Extensible Markup Language (XML) 1.0 (Fifth Edition).* <http://www.w3.org/TR/2008/REC-xml-20081126/>
[b-AJAX]	Open Ajax alliance <http://www.openajax.org/>
[b-JSON]	JSON (JavaScript Object Notation), based on: ECMA (1999), *ECMAScript Language Specification*, Standard ECMA-262, 3rd edition. <http://www.json.org>
[b-LBA]	J. Magazinius, A. Askarov, and A. Sabelfeld. *A lattice-based approach to mashup security*. In Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, pages 15-23, 2010. <http://www.cse.chalmers.se/~andrei/asiaccs10.pdf>
[b-MASHUPS]	Hanson, Jeffrey J. (2009), *MASHUPS Strategies for the Modern Enterprise*, Addison-Wesley.
[b-OVFGS]	L.A. Meyerovich, A.P. Felt, and M.S. Miller. *Object views: Fine-grained sharing in browsers*. In Proceedings of the 19th international conference on World wide web, pages 721-730, 2010. <http://www.cs.berkeley.edu/~afelt/views-www-2010.pdf>

[b-REST]	Fielding, R.T. (2000), *Representational State Transfer (REST), In: Architectural Styles and the Design of Network-based Software Architectures* [Dissertation], Irvine, University of California, Irvine. <http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>
[b-SECMASH]	Philippe De Ryck, Maarten Decat, Lieven Desmet, Frank Piessens, and Wouter Joosen, *Security of web mashups: a Survey*. <https://lirias.kuleuven.be/bitstream/123456789/317390/1/paper.pdf/>
[b-SEFGSP]	B. Livshits and L. Meyerovich. *Conscript: Specifying and enforcing Fine-grained security policies for javascript in the browser*. Technical report, Microsoft Research, 2009. <http://research.microsoft.com/pubs/120969/paper.pdf>

SERIES OF ITU-T RECOMMENDATIONS

Printed in Switzerland
Geneva, 2014