Abstract
SAFE Scripting Method is a model for allowing application interactivity in email while simultaneously eliminating security vulnerabilities associated with client-side scripting.

Introduction
SAFE (Server-side Asynchronous Framework Execution) Scripting Method has only two intended objectives:

1) This model provides a method to allow behavior, or event-oriented, execution of programmatic application code across email. Such code will be referred to as script.

2) This model seeks to provide an alternative to client-side scripting of world wide web (WWW) documents free of security vulnerabilities with cross-site scripting (XSS) and cross-site request forgery (CSRF).

Each of these objectives is mutually necessary for the functional existence of the other. The SAFE model refers to the model of steps necessary to achieve the SAFE Scripting Method as explained below.

Requirements
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [34].

Three externally defined requirements are necessary for the implementation of this proposed model. One additional requirement is necessary and will be further expanded later in this document. The external requirements MUST NOT be defined by this document.

1) The first requirement is a standardized and well understood document structure definition, such as a markup language, that conforms to the conventions of the standardized Document Object Model (DOM) and accurately describes data intended for transmission across SMTP while simultaneously representing sufficient current common practices of representing or describing data intended for distribution as email or SMTP, which MUST NOT be (X)HTML. This requirement shall be arbitrarily referred to as ‘markup’ for the remainder of this document.

2) The second requirement is a standardized and widely adopted transmission scheme that reflects the primitive model defined by RFC 5321 while simultaneously respecting the rules and constraints defined by the document structure noted in Requirement 1, markup. This requirement shall be arbitrarily referred to as ‘protocol’ for the remainder of this document and MAY be representative of RFC 5598.
3) The third requirement is a certificate authority granting organization that is entirely external to any organization providing Requirement 1 (markup) or Requirement 2 (protocol). This requirement shall be arbitrarily referred to as CA, short for certificate authority, for the remainder of this document.

4) The third requirement is the standardization and adoption of a new programming language object, XMLSmtpPush object, which is intended to offer comparable functionality to the XMLHttpRequest object available on the WWW.

Elaboration of intended objectives

Allowing script execution in email is absolutely necessary to provide a minimal expectation of user experience and interaction, as well as promoting the advancement of data and commerce exchange. Behavior and event oriented scripting in email must not be compared with similar functionality available on WWW. On the WWW such scripting executes locally to the end user, which allows a degree of freedom and rapid programmatic response not available to this model at the cost of frequent security violations.

In this model scripts are executed on an email server, a mediating agent of the data transmission that is never an end point of that transmission. Execution occurs in response to an event, specific expected data, a defined document feature, or other programmatic condition. This means the code executed by any script language must be pushed back to the initiated user as either a new document or as an asynchronous alteration to an established document.

The reason why scripts must be executed on the server and not the client is to eliminate security vulnerabilities associated with script execution. Understandably, this opens new and more difficult to detect security vulnerabilities where compromises violate expectations of privacy. Email is inherently private and it is the liability of the distant server’s owner to ensure data that passes through that mail server remains private through best current security practices.

Models established by RFC 5321
SAFE Model

The SAFE model exists to allow transmission interactivity through execution of script opposed to programmatic interactivity that exists only through use of client-side programming language execution in WWW. In other words, the SAFE Model exists to provide interaction using dynamic communications that do not result in new or separate content documents that would appear to be additional emails. The mail clients MUST be expected to execute changes to the DOM of an establish document but MUST NOT process any other instructions with regard to this model or transmission interaction. This is secure because changes to the DOM only allow for the deletion, addition, or alteration to text or markup code data to a document only. The SAFE model expects that all communications will be encrypted as a result of assymmetric key exchange.

SAFE - Step One

In the SAFE Model the first step is to establish authorization between the two communicating clients to ensure the distant client is not a forged identity or spoofed address. If authorization is not established encryption could be exist with a fraudulent identity. A CA, as defined in Requirement 3, is necessary to ensure the integrity of the authorization process.
The distant client MUST respond with a digital certificate signed with a digital signature. Upon certificate receipt the initiating client MUST contact the issuing CA to receive a key to decrypt the digital signature. Since this key can only decrypt signatures signed with a private key stored at the distant user authorization is verified. This CA MUST NOT be on the same domain as either client address or any email server mediating in the transmission in order to provide an uncompromised validation of trust. This process SHOULD be transparent to the user of the originating client.

SAFE - Step Two
The second step is for both communicating clients to share public keys and engage in asymmetric encryption only after authorization is verified using Step One. If the initiating client inserts their public key in the first communication to the distant client, using a convention provided by the markup language for describing public keys, and the distant client responds with their public key, using the same markup convention, this step is unnecessary as both clients already have each other’s public key. Otherwise additional communications are required to exchange public keys between the initiating client and distant client.

SAFE - Step Three
The third step is for the initiating client to engage in communication to the distant client, such as sending an email. When the distant client is ready to engage script execution it must prompt its local mail server to initiate Step Four. This communication between the distant client and its local mail server MAY occur using any transmission protocol even if that protocol is not affiliated with email or SMTP and even if that method of communication is proprietary. When communicating to the distant mail server the distant client MUST include the public key value of the initiating client.
SAFE - Step Four
The forth step is for the distant client’s email server to send a public key exchange request to the initial client. The distant mail server must have a public/private key pair that is entirely unique and not affiliated with the key pair used by the distant client. This separate key pair is necessary, because it notifies the initial client that it will be engaging in communication with a different entity and it prevents accidental or involuntary entry into a model of script execution. This additional notification provides the initiating user a final opportunity to disallow use of the SAFE Model for any reason. Only the distant mail server needs to send a message with its key value since it was provide the initiating client’s public key in Step Three.

SAFE - Step Five
The fifth step is for the initial client and distant email server to engage in encrypted communication. At this point communication between the initiating client and the distant mail server MAY occur as many times as the distant mail server sends a response to the initiating client or as many times as the initiating client communicates to the distant mail server only.

SAFE - Step Six
The sixth step is to close communications between the initiating client and the distant mail server and then send a notification to the distant client. This communication from the distant mail server to the distant client verifies that SAFE Model communication has occurred and properly terminated. The termination communication MUST contain the final communication that is intended to be delivered to the distant client.
Once the SAFE Model has terminated the private data stored on the distant mail server MUST be destroyed or purged. If an interruption to communication occurs, so that a proper termination is never generated, the distant server MUST provide a defined timeout by which all stored communication will be automatically destroyed or purged. The initiating client SHOULD be provided with the opportunity to terminate the SAFE Model at every possible opportunity.

SAFE - Step Seven
The distant client MUST send an automated response email to the initiating client once the distant client verifies termination of the SAFE Model. If the distant client fails to send a response within a given timeout, specified by the initiating client’s user agent from the time of most prior response, the SAFE Model MUST be presumed to have failed. If the SAFE Model has failed the initiating client MAY start over in order to initiate a new SAFE Model instance or MAY communicate directly to the distant client without use of the SAFE Model.

SAFE - Summary
The end result of the SAFE Model is that communication may be encrypted between two distant clients and a separate simultaneous application session may be established between an independent third-party which is already present in the transmission path of the two clients. This effectively means, given SMTP derived protocols as a push transmission, the initiating client will be communicating directly to the distant client while such communications may be intercepted and automatically responded to by the distant server. The distant server, however, MUST be a mediating agent of communication and MUST NOT be a destination of communication.

Additional security constraints applied to SAFE Model
* All script, or programmatic execution, that is to occur during the SAFE Model MUST occur only from code that resides locally on the distant server. Local is defined as the single operating system on which the single specific mail server executes or resides. Scripts or other programmatic code MUST NOT request, refer to, or source in any code that is not stored local to the executing server.

* Any code specified to execute during the SAFE Model MUST reside as a new instance of the original code in a single ‘sand-boxed’ location so that other code is not executed by mistake and code is not open to security compromises as a result availability to a data transmission.

* Communications that arise between the distant server and initial client in the SAFE Model MUST be considered temporary and private. The distant server MUST NOT pass communication received or generated from the SAFE Model to the distant client until except for the final terminating communication of the SAFE Model to the distant client.
It is RECOMMENDED that a separate email address be specified for a distant client that is present to receive maintenance and status data from the distant server. This is intended to ensure that there is no collusion of maintenance data and user communication. Maintenance data and other gathered data, such as statistics, MUST NOT be reported or generated with uniquely identifiable user information, content, or header data. Such data MAY be generated for internal reporting to the owner of the distant mail server and distant client if it cannot be matched against uniquely identifiable user data.

* Communications sent from the distant server MUST be sent with the same MIME content type as the data received by the distant server from the initiating client. If this results in incompatibilities at the distant mail server the distant mail server MUST properly terminate the SAFE Model, but send an error report to the distant client instead of the intended communication.

* The CA MUST NOT reside within the same domain as either the initiating client, the initiating client’s direct mail server, the distant server, or the distant client.

XMLSmtpPush object definition
This programmatic object exists to allow script code to open a transmission path from a point of execution to a known addressable point independent of the communication details specified in either the transmission header information or document specifications. This transmission path MUST be a unidirectional path from the agent of execution, typically the agent of execution would be the distant server of the SAFE Model, to the addressable provoking entity, which would typically be the initial client of the SAFE Model. This represents a fire and forget method of transport where the agent of execution sends data once and expects to never receive either a success or failure transmission status. The XMLSmtpPush object MUST NOT open a transmission to any destination other than the provoking entity.

The XMLSmtpPush object MUST retrieve data from the most recent document header specific in the markup language, if available, or if that is not available then from the packet header of the transmission protocol. If the header data is not understood, not defined, or not well-formed the data MAY be transported with RFC 5322 conformant headers. Data transmitted by this object using RFC 5322 conformant headers SHOULD NOT expect successful or accurate interaction with the intended document.

Despite its name, the XMLSmtpPush object is not intended for limited implementation across the SMTP protocol only. The object is intended for functional operation across any protocol that conforms to the defined protocol requirements of RFC 5322, or compatible protocols, and its primitive models.
XMLSmtpPush object methods

 abort()
   The abort() method stops the current request by closing the
   transmission path before any data is sent if the transmission path is
   opened and returns a value of "false". If a transmission path is not
   opened a value of "false" is returned. This method is required to
   occur at least once per open method. If this method does not occur
   to close an open method an error MUST be thrown.

getHeaderDatagram()
   The getHeaderDatagram() method returns the header names and values in
   a JSON object or multidimensional array. If the document header
   cannot be detected or understood a value of null MUST be returned.

getHeader("headername")
   This method returns the string value of a single specified header if
   the specified header exists, otherwise this method returns a value
   of null.

open()
   The open method opens an asynchronous transmission path from the
   executing agent to the provoking entity. The open method MUST NOT
   send data.

send(variable1,variable2,...)
   The send method MUST send data to the provoking entity. The send
   method MUST NOT open or prepare a transmission path, which is the
   function of the open method. This means a send method MUST execute
   only after an open method has executed. If a send method attempts to
   execute without a prior instantiated open method an error MUST be
   thrown. This granular strict functionality between the open and send
   methods is intended to provide script authors greater control of
   various different data sets to be sent while simultaneously reducing
   complexity derived from such granularity.

   The send method expects to receive instructions for altering a
   document and the new content that is to be provided, if any, where
   such instructions are stored in a function defined as a single named
   variable. Such a named function SHOULD be conservative in its
   instructions and liberal in its static content. The specified
   instructions MUST be expressed as DOM methods, XPath expressions,
   and/or the innerHTML method with anticipation that such instructions
   will be processed by the client without decisions or formatting onto
   the instructions, its content, or its perceived impact on the
   document. DOM instructions MUST be specific to the document received
   by the agent of execution, or they SHOULD expect to fail to execute
   at the provoking entity.
Example of XMLSmtpPush object in JavaScript language assuming MML header

```javascript
var checkoutPage1 = function () {
    var safe, insta, instb,
        div = document.getElementById('content'),
        select = document.getElementById('selectlist').value,
        checkbox = document.getElementById('checkbox').checked,
        chkresponse = document.createElement('p'),
        slctresponse = document.createElement('p'),
        from = document.getElementById('from'),
        replyto = from = document.getElementById('reply-to')[0],
        chkresponse.setAttribute('id', 'checkresponse'),
        slctresponse.setAttribute('id', 'selectresponse'),
        insta = function () {
            div.appendChild('chkresponse');
        },
        instb = function () {
            div.appendChild('slctresponse');
        },
        if (XMLSmtpPush) {
            safe = new XMLSmtpPush();
            if (safe.getHeader('from').value !== undefined) {
                safe.open(getHeader(from).value);
            } else if (safe.getHeader('reply-to').value !== undefined) {
                safe.open(getHeader(replyto).value);
            }
            if (checkbox === true && select === 'default') {
                safe.send(insta);
            } else if (checkbox === true && select === 'first') {
                slctresponse.textContent = 'You selected the first option.';
                safe.send(insta, instb);
            } else if (checkbox === true && select === 'second') {
                slctresponse.textContent = 'You selected the second option.';
                safe.send(insta, instb);
            } else if (checkbox === false && select === 'first') {
                slctresponse.textContent = 'You selected the first option.';
                safe.send(insta);
            } else if (checkbox === false && select === 'second') {
                slctresponse.textContent = 'You selected the second option.';
                safe.send(insta);
            }
            safe.abort();
            return true;
        } else {
            return false;
        }
};
```
Processing role of the client

The client MUST be prepared to execute DOM instructions, XPath expressions, or the innerHTML method. Any other client-side script execution MUST NOT occur aside from meta-language parsing of a document only with regard to the appropriate meta language the markup language conforms to. The user MUST NOT be allowed to interfere with the processing of such instructions. If the user did not wish for such instructions to execute the user would not have voluntarily allowed execution of the SAFE Model.

The intent is to remove security vulnerabilities associated with client-side code execution requests from a message by pushing those vulnerabilities onto a server not directly associated with the user. The user SHOULD be fully aware that data MAY be coming in as asynchronous updates and not new documents by convention of the SAFE Model.

Security

Client-side script execution is associated with the highest quantity of security vulnerabilities in computing. Eliminating client-side scripting on the web and instead using SAFE can nullify most of these problems making the internet a safer place.

IANA Considerations

This document contains no IANA considerations.

References

[ARCHITECTURE]

[DOM]

[FORMAT]

[IANA]


<http://www.w3.org/TR/xmlschema-0/>


Boyer J. "XForms 1.0 (Third Edition)", W3C Recommendation, IBM, October 2007.  
<http://www.w3.org/TR/2007/REC-xforms-20071029/>

Bray T, Paoli J, Sperberg-McQueen C, Maler E, Yergeau, F. "Extensible Markup Language (XML) 1.0 (Fourth Edition)", W3C Recommendation, Textuality, Microsoft, W3C, Sun, August 2006.  
<http://www.w3.org/TR/REC-xml/REC-xml-20060816.xml>

<http://www.w3.org/TR/xhtml1/>

<http://www.w3.org/TR/xhtml2/>
Acknowledgements

This work was inspired by the AJAX concept, the security vulnerabilities of client side scripting on the WWW, and the potential implications of the Mail Markup Language (MML).

Author’s Address

Austin Cheney
User Interface Technologist, Travelocity
3150 Sabre Drive
Southlake, TX 76092

PHONE: (682) 605-1000
EMAIL: austin.cheney@us.army.mil

Copyright (c) 2009 IETF Trust and the persons identified as the document authors. All rights reserved.

Expires February 1, 2010

Cheney Standards Track [Page 14]