MIME Extensions for Mail-Enabled Applications: application/Safe-Tcl and multipart/enabled-mail

Nathaniel Borenstein, *Bellcore*
Marshall Rose, *Dover Beach Consulting*

May, 1993

Status of this Memo

This document is a working draft. Do not cite, copy, or circulate.

Abstract

MIME [RFC-MIME] defines a format and general framework for the representation of a wide variety of data types in Internet mail. This document defines two new subtypes of MIME data, the application/Safe-Tcl and multipart/enabled-mail subtypes, for providing enabled mail [EM-MODEL] in the Internet community.

A table of contents appears at the end of this document.

1. Overview

Most electronic mail, even multimedia mail as standardized in MIME [RFC-MIME], is "passive" in the sense that the data are unidirectional. Textual, image, audio, or video data are simply displayed to the user, who reads, views, listens, or watches it and then must take specific action to initiate any response to the data, such as to reply to the originator, to replay the data, or to redistribute it to other recipients.

Less commonly used, but the subject of considerable research attention, has been "active" mail, in which the data delivered through the mail constitute a program in a well-specified language, allowing the program to be automatically evaluated on behalf of the recipient when the mail is "read." Researchers have demonstrated fascinating applications of this concept, and in recent years have shown that the critical problems of safety and portability can be solved in a straightforward manner [ATOMICMAIL].

This memo defines a standardized format for the interoperation of active mail in the context of MIME. It defines a new language, "Safe-Tcl", based on the "Tcl" language [TCL]. It also defines a new MIME content-type value, "application/Safe-Tcl", which may be used to tag a MIME entity (a mail body or body part) as being a program in the Safe-Tcl language. Additionally, this memo defines a new multipart subtype, "multipart/enabled-mail", for grouping together an interactive mail program and an arbitrary MIME entity to which it is related.

The reader should consult [EM-MODEL] for a description of the basic Enabled Mail model, which is not presented here. This memo also does not provide a tutorial in either the fundamental problems of safety and portability in active messaging, which the interested reader may find in [ATOMICMAIL]. Nor does this memo provide a tutorial in the Tcl language itself, for which the interested reader is referred to [TCL].

This memo assumes a basic familiarity with the syntax of Tcl, and defines Safe-Tcl in terms of its differences from standard Tcl. The resulting language is believed by the authors to be safe for email use, according to the reasoning outlined in [ATOMICMAIL]. In particular, it is the intent of the Safe-Tcl language design that it should be essentially harmless to evaluate a Safe-Tcl program that comes from an unknown or hostile sender.

2. The application/Safe-Tcl content-type

The language defined in this memo shall be labelled as a MIME body or body part by the use of the "application/Safe-Tcl" content-type. Two mandatory content-type parameters are defined for this content-type. The first parameter, "version", is a version number for the Safe-Tcl language itself. For the version of Safe-Tcl defined in this memo, the version value should be "6.8". The second parameter, "evaluation-time", is a string describing the intended time of evaluation for the Safe-Tcl program. Thus the content-type line might look something like this:

```
Content-type: application/Safe-Tcl; version="6.8";
        evaluation-time=activation
```

The choice of "6.8" is indicative of the fact that the Safe-Tcl language, as described here, is derived from Tcl version 6.8. However, this should NOT be taken to indicate that arbitrary other versions of Tcl may be used with a corresponding change to the version parameter. If a future version of Safe-Tcl is ever defined, it will be formally specified and published as part of the MIME process. It is explicitly NOT the case that arbitrary versions of Tcl may be used with a suitably modified version parameter.

The evaluation-time parameter may have one of two values, "delivery" or "activation", which corresponds to the delivery-time and activation-time phases defined in [EM-MODEL]. A value of "activation" means that the program is intended to be evaluated whenever the user views the message, and may need to interact with the user. A value of "delivery" means that the program is intended to be evaluated upon final delivery to the user's mailbox, and cannot interact directly with the user, though it may interact with user-supplied Safe-Tcl extensions.

Note that a MIME message that contains an application/safe-tcl entity with an evaluation-time of "delivery" is intended to be evaluated at delivery time. Such an entity will ONLY be evaluated, however, if it appears as either the top-level MIME content-type or a second-level type, directly inside a multipart/enabled-mail entity. A nested MIME application/safe-tcl entity with an evaluation-time of "delivery" should be ignored.

3. The multipart/enabled-mail content-type

This memo defines a new subtype of the MIME multipart content-type, "multipart/enabled-mail". A multipart/enabled-mail entity will have exactly two subparts, the first of which will be of arbitrary type, and the second of which will be of type application/Safe-Tcl (or some future language for enabled mail).

The intended semantics of multipart/enabled-mail, when viewed by a human reader, are as follows: If there is no application/Safe-Tcl interpreter available, or if the application/Safe-Tcl part has an evaluation-time of anything other than "activation", then the Safe-Tcl program should be skipped and the first part, an abitrary MIME entity, should be displayed normally. If, however, the Safe-Tcl program has an evaluation-time of activation, and a Safe-Tcl interpreter is available, then the Safe-Tcl program should be evaluated and the first part of the multipart/enabled-mail object should NOT be displayed. (Parts of it, however, may have been displayed under the control of the Safe-Tcl program.)

4. The Safe-Tcl Language

The syntax of Safe-Tcl is identical to the syntax of Tcl [TCL]. No syntactic constructs are changed. The only differences, therefore, between Tcl and Safe-Tcl is the set of available primitive functions and procedures. Safe-Tcl may be described as an "extended subset" of Tcl, in that the "dangerous" primitives in Tcl have been removed, while certain new primitives have been added.

It is assumed that the process evaluating a Safe-Tcl script will always have within it TWO interpreters, one for full Tcl and one for Safe-Tcl. A correct implementation will not let a program being evaluated by the Safe-Tcl interpreter have access to the full Tcl interpreter except via the mechanisms defined as part of the Safe-Tcl language.

4.1. The Core Safe-Tcl Language

Because Tcl is an evolving language, it is not considered sufficient to describe Safe-Tcl completely in terms of differences from this base, as this may prove dangerously confusing if some future version of the language includes a potentially dangerous primitive not mentioned in the list of differences. Therefore, this memo provides a complete list of all the Safe-Tcl primitives "inherited" from standard Tcl. No other primitives should be provided by a Safe-Tcl interpreter. As a convenience to the reader, this memo will also list the standard Tcl primitives that were consciously omitted from Safe-Tcl, but this list should not be considered exhaustive, in that any Tcl primitive that is not explicitly mentioned as being part of Safe-Tcl should be considered NOT to be part of Safe-Tcl.

In particular, the following standard Tcl commands are considered unsafe or inappropriate for email use, and are NOT to be interpreted by Safe-Tcl interpreters:

> auto_execok, auto_load, auto_mkindex, auto_reset, cd, close, eof, exec, file, flush, gets, glob, open, puts, pwd, read, seek, source, tell

The core set of standard Tcl commands which ARE a part of the Safe-Tcl language are:

> append, array, break, case, catch, concat, continue, error, eval, exit, expr, for, foreach, format, global, history, if, incr, info, join, lappend, lindex, linsert, list, llength, lrange, lreplace, lsearch, lsort, proc, regexp, regsub, rename, return, scan, set, split, string, time, trace, unknown, unset, uplevel, upvar, while

The core Safe-Tcl language also includes the following global variables from standard Tcl:

errorCode, errorInfo

Other global variables might be defined as needed by a Safe-Tcl interpreter, but their presence should not be relied on.

In addition, Safe-Tcl includes additional built-in procedures and variables that are NOT part of standard Tcl, defined in the sections that follow. Some of these are available to all Safe-Tcl programs, while others are available only with certain values of the evaluation-time parameter or in certain user interface environments.

4.2. Universal Safe-Tcl Functionality

The following primitves are always part of the Safe-Tcl language.

SafeTcl_getconfigdata -- `"SafeTcl_getconfigdata key ?default ?prompt?"`. To permit user customization of Safe-Tcl applications in the absence of any generalized file system access, Safe-Tcl includes a mechanism for associating a customization string with a key string. SafeTcl_getconfigdata should return the string value associated with the key, and should generate an error if this is not possible. If the user has not previously specified the customization value (which might be done by a mechanism that is specific to the particular Safe-Tcl interpreter), the Safe-Tcl interpreter should engage the user in a dialog to obtain the value, which should then be saved for future use. In this case, the optional second and third arguments provide a default value and a prompt to explain the nature of the data needed to the user. If the user HAS previously supplied a customization value to the user, then whether or not any user action is invoked by the SafeTcl_getconfigdata primitive is implementation-dependent, but a suggested action is to use the previously-supplied value as a default and to ask the user to confirm that this is still the correct value. Note that if the evaluation-time is "delivery", any user interaction is impossible. In this case, if user-supplied data is available it should be used, and otherwise an error should be generated.

SafeTcl_random-- `"SafeTcl_random min max"`. This primitive returns a pseudo-randomly generated integer greater than or equal to the integer min and less than or equal to the integer max. If min is equal to max, that value will always be returned. If min is greater than max, an error will be generated.

SafeTcl_encryptstring -- `"SafeTcl_encryptstring string algorithm key"`. This primitive encrypts the string supplied as the first argument using the algorithm specified by a string name in the second argument and the encryption key provided as the third argument. The encrypted value is returned. If encryption is not implemented or cannot be performed for any reason (such as international export controls), an error will be generated. The string names to be used for algorithms are registered with the IANA and documented elsewhere. An algorithm name starting with "x-" may be used by private agreement.

Additionaly, Safe-Tcl always defines a global variable that indicates the current evaluation-time context:

SafeTcl_evaluation_time -- A string that is set to either "delivery" or "activation" to indicate the current evaluation-time context.

4.3. Additional Messaging Functionality

The following primitives are always part of the Safe-Tcl language when used in the context of MIME, but may not be present if the language is adopted for non-email use.

Note that four of these procedures (SafeTcl_getheader, SafeTcl_getheaders, SafeTcl_getbodyprop, and SafeTcl_getparts) may make implicit reference to a MIME message. Each of them has an optional parameter, specified here as "?body?", which may contain a MIME entity. If this optional parameter is not supplied, then, in the case of evaluation-time "delivery", the ?body? is assumed to be the entire MIME message that has just been delivered. In the case of evaluation-time "activation", if the Safe-Tcl program is part of a multipart/enabled-mail MIME entity, then the ?body? is assumed to be the OTHER part of that multipart/enabled-mail entity. In the case of evaluation-time "activation" when the Safe-Tcl program is NOT part of a multipart/enabled-mail message, an error will be generated if an explicit body is not provided as a string argument.

SafeTcl_getaddrs -- `"SafeTcl_getaddrs string"`. This returns a list, each element of which is a string containing an electronic mail address found in the argument.

SafeTcl_getaddrprop -- `"SafeTcl_getaddrprop address property"`. This returns the specified property from the address string. Properties are:

```
Property   Returns Description
--------   ------- -----------
proper     string  official 822 rendering,
                   e.g. "phrase <local@domain>"
friendly   string  user-friendly rendering (see Appendix C)
address    string  local@domain rendering
phrase     string  the phrase part
local      string  the local part
mymbox     integer "1" if this is the recipient's address,
                   as determined by local configuration,
                   "0" otherwise.
domain     string  the domain part
```

If the string can not be parsed as a mail address, an error should be generated.

SafeTcl_getdateprop -- `"SafeTcl_getdateprop date property"`. This returns the specified property from the date string, which is a date specification in RFC 822/1123 format. Properties are:

```
Property   Returns Description
--------   ------- -----------
sec        integer seconds of the minute
min        integer minutes of the hour
hour       integer hours of the day (0-23)
wday       integer day of the week (Sun=0)
day        string  day of the week
                   (3 char abbreviation)
weekday    string  day of the week
sday       integer day of the week known?
                   (1=explicit 0=implicit, -1=unknown)
mday       integer day of the month
yday       integer day of the year
mon        integer month of the year
month      string  month of the year
                   (3 char abbreviation)
lmonth     string  month of the year
```

```
year       integer year (all digits, e.g. 1993)
zone       integer timezone in hours
tzone      string  timezone string
szone      integer timezone known?
                   (1=explicit, 0=implicit, -1=unknown)
date2local string  coerce date to local timezone
date2gmt   string  coerce date to GMT
dst        integer daylight savings in effect?
rclock     integer seconds prior to current time
proper     string  official 822 rendering
```

If the string can not be parsed as a date, an error should be generated. If the date specified is the empty string, the current date and time should be used.

SafeTcl_getheader -- `"SafeTcl_getheader field ?body?"`. This returns the value of a field in the message's (or MIME entity's) headers. If the field is not present in the headers, the null string is returned. If the header field occurs more than once in the headers, the associated values are concatenated together according to the rules of RFC 822 (e.g., the values associated with multiple occurrences of a header field which contains addresses are concatenated with a comma).

SafeTcl_getheaders -- `"SafeTcl_getheaders ?body?"`. This returns an array of lists, each of which identifies a header field contained in the message (or MIME entity). (The SafeTcl_getheader routine can be used to extract the value associated with each header field.) Each of these lists has two string elements, the first of which is a header field name, and the second of which is a header field body. If the MIME entitiy contains multiple header fields with the same name, each will appear as a different list.

SafeTcl_makebody -- `"SafeTcl_makebody content-type [-parameter string] [-description string] [value encoding] ..."`. This creates a MIME entity of the specified content-type. (If the empty string is given as the content-type, "text/plain" is assumed.) The "-parameter string" sequence may occur zero or more times to indicate whatever parameters are associated with the content type. The "-description string" sequence may occur at most once to specify the content-description field. For a multipart content, each remaining parameter describes a subordinate body-part. Otherwise, only one remaining parameter is present, which specifes the data associated with the body. Each body is described as a list, the first element of which is the data value, and the second element of which is a string describing any content-transfer-encoding algorithm that has been applied (i.e. "base64" or "quoted-printable" or "" for no encoding). If no encoding has been applied, the second element may be omitted from the list describing the body-part.

SafeTcl_getparts -- `"SafeTcl_getparts ?body?"`. This returns a list, each element being a list that identifies a MIME entity contained within the body parameter. Each of these lists consists of a numeric identifier, a content-type, and a content-description string. The return value is constructed by a pre-order traversal of the body, with the prefix of a subordinate entity being copied from its parent. For example, if the structure of a message were:

```
multipart/mixed
  text/plain
  multipart/digest
    message/rfc822
  audio/basic
```

then the list returned would have this structure:

```
[["1" "multipart/mixed" "A bunch of stuff"]
   ["1.1" "text/plain" "Introduction"]
   ["1.2" "multipart/digest" "Today’s news"]
     ["1.2.1" "message/rfc822" "A word from Bill"]
   ["1.3" "audio/basic" "Many words from Bill"]]
```

SafeTcl_getbodyprop -- "`SafeTcl_getbodyprop property part ?body?`". Returns a string containing the value of the specified property for body part specified by the second and third parameters. Properties are:

```
Property   Returns Description
--------   ------- -----------
type       string  value of Content-Type field
                   (without parameters)
parms      list    each element a parameter from
                   the Content-Type field, given
                   as a list of two items
```

```
                      [paramname paramvalue]
id          string    value of Content-ID field
descr       string    value of Content-Description
                      field
value       string    data value, possibly encoded
encoding    string    "base64", "quoted-printable",
                      or "" (where "7bit", "8bit",
                      or "binary" was used)
```

SafeTcl_encode -- "`SafeTcl_encode encoding data`". This takes the specified data and encodes it according to the MIME encoding specified by the encode argument, which must be either "quoted-printable" or "base64". It returns a string that is the encoded data.

SafeTcl_decode -- "`SafeTcl_decode encoding data`". This takes the specified encoded data and decodes it according to the MIME encoding specified by the encode argument, which must be either "quoted-printable" or "base64". It returns a string that is the decoded data.

4.4. Additional Delivery-Time Functionality

When a Safe-Tcl program is delivered in a mail message with the evaluation-time parameter given as "delivery", then no interaction with a user is possible. In this context, two additional global variables are available:

SafeTcl_originator -- A string containing the originator of the message, as indicated by the envelope.

SafeTcl_recipient -- A string containing the recipient of the message, as indicated by the envelope.

Also, the following additional procedures are defined:

SafeTcl_getmessagelength -- "`SafeTcl_getmessagelength`". This function, which takes no arguments returns a string giving the length of the message in octets, including the message headers.

The way that CRLF pairs are counted, for this purpose, is implementation-dependent, but should be consistent with the way SafeTcl_getmessage functions.

SafeTcl_getmessage -- "`SafeTcl_getmessage start len`". This function returns a string containing all or part of the entire message, including the message headers. The start parameter tells where in the message to the string should start, where 0 is the first octet. The len parameter tells how many octets of the message to return, where -1 means the whole message. Thus "SafeTcl_getmessage 0 -1" will return the entire message, but more sophisticated approaches can be used to avoid allocated a Tcl string for, say, a 2 gigabyte video clip.

4.5. Additional Activation-Time Functionality

When a Safe-Tcl program is received in a mail message with the evaluation-time parameter given as "activation", the mail message is intended to be run in an interactive setting, with the ability to interact with the user. Several additional Safe-Tcl procedures may become available in this context, some of which are only available in certain user interface contexts.

In a non-mail Safe-Tcl application, these primitives may also be present if and only if there is a user with whom the program can interact.

4.5.1. User Interaction Models: SafeTcl_InterfaceStyle

Second only to safety as a critical issue for an active messaging language is the question of user interface capabilities. Since the language needs to be able to work in a wide variety of hardware and software environments, it is difficult to avoid a "lowest common denominator" user interface. Safe-Tcl addresses this problem by providing a few lowest common denominator primitives, and then by providing a mechanism by which the availability of well-defined packages of more advanced user interface mechanisms can be made known to a Safe-Tcl program at runtime.

In particular, it is specified that each Safe-Tcl interpreter must provide a global variable, SafeTcl_InterfaceStyle, which indicates any user interface extensions that are available. If no user interface extensions are available, the variable should be set to the one-element list "{generic}" to indicate that only the generic "lowest common denominator" functions are available.

In order to permit future Safe-Tcl interpreters to implement multiple user interface extensions, the value of SafeTcl_InterfaceStyle may be a Tcl list of supported user interface extensions. Thus if a Safe-Tcl interpreter supported both the "foo" and "bar" user interface extensions, it would set SafeTcl_InterfaceStyle to {foo bar} or {bar foo}.

(The support for the generic functions is always required, and therefore need not be declared in SafeTcl_InterfaceStyle. However, it would also be legitimate to set this value to {foo bar generic}, for example.)

The string or strings in the SafeTcl_InterfaceStyle should all be interpreted in a case-insensitive manner.

It is expected that a common approach to writing Safe-Tcl programs will be to include multiple versions of user interface functions, as in the following pseudo-Safe-Tcl-code

```
if {[lsearch $SafeTcl_InterfaceStyle "Tk3.2"]} {
    do_Tk_style_interaction
} else {
    do_generic_style_interaction
}
```

It is expected that gradually many such constructs will be abstracted and hidden behind reusable "portable" procedure definitions (e.g. an "AskMultipleChoice" procedure), but the SafeTcl_InterfaceStyle mechanism will permit Safe-Tcl program authors access to more general user interface mechanisms than if they were limited to such portable procedures, as in [ATOMICMAIL].

4.5.2. Generic User Interaction

The following user interaction primitives are available in the "generic" interface style, and hence are available to all interactive Safe-Tcl programs.

SafeTcl_displaytext -- `"SafeTcl_displaytext txt"`. This primitive simply shows the given text to the user. The string specified may be of arbitrary length, so consideration must be given to scrolling or pagination as necessary. Zero is always returned; an error is generated if the string could not be displayed.

SafeTcl_displayline -- `"SafeTcl_displayline txt"`. This primitive simply shows the given text to the user. The string specified is a single line of text. Zero is always returned; an error is generated if the string could not be displayed.

SafeTcl_gettext -- `"SafeTcl_gettext prompt ?default?"`. This primitive obtains an arbitrary body of text from the user, which is returned as a Tcl string. The first argument is used as a prompting string to solicit the text from the user, while the optional second argument is the default to be offered. An error is generated if the string cannot be obtained.

SafeTcl_getline -- `"SafeTcl_getline prompt ?default?"`. This primitive obtains a single line of text from the user, which is returned as a Tcl string. The first argument is used as a prompting string to solicit the text from the user, while the optional second argument is the default to be offered. An error is generated if the string cannot be obtained.

SafeTcl_displayentity -- `"SafeTcl_displayentity entity ?body?"`. This causes a MIME entity to be displayed to the user. The "entity" string may be one of the numeric values returned by SafeTcl_getparts, or (if it begins with "<" and ends with ">") it may be a Content-ID value identifying the MIME entity. The manner in which the entity is displayed, and the set of MIME types that are supported, is implementation-dependent. SafeTcl_displayentity returns 0 if the entity was displayed to the user, and generates an error if for any reason it was not.

Note that the five primitives SafeTcl_displaytext, SafeTcl_displayline, SafeTcl_gettext, SafeTcl_getline, and SafeTcl_displayentity, constitute the entire "generic" user interface of the core Safe-Tcl language. Additional user interface capabilities may be indicated using the always-present global variable SafeTcl_InterfaceStyle, as described above. However, these five primitives are guaranteed to be available for every Safe-Tcl implementation, and can be used either to write "lowest common denominator" user interfaces or to provide a backup user interface when the SafeTcl_InterfaceStyle variable indicates that no recognized user interface extensions are available.

4.5.3. X11 Interaction: Interface Style Tk3.2

This document defines the use of a single user interface extension set, corresponding to a large subset of Tk, the X Window System extensions for Tcl. The availability of this user interface capability is declared by the inclusion of the string "Tk3.2" in the SafeTcl_InterfaceStyle variable. The choice of "3.2" is indicative of the fact that the Tk primitives described here are derived from Tk version 3.2. However, this should NOT be taken to indicate that arbitrary other versions of Tk may be used with a corresponding change to the SafeTcl_InterfaceStyle string. If a future version of the Tk interface style for Safe-Tcl is ever defined, it will be formally specified and published as part of the

MIME process. It is explicitly NOT the case that arbitrary versions of Tk may be used with a suitably modified InterfaceStyle value.

As with the core Safe-Tcl language, the Tk extensions will be described in terms of differences from the standard Tk3.2 language. Since Tk does not extend the basic syntax of the language, all that neds to be specified is the set of available primitives.

Only one Tk primitive is omitted from Safe-Tcl, namely the "send" primitive.

All of the other core Tk procedures and functions are retained. In particular, the COMPLETE set of functions to be define by the "Tk3.2" interface style for Safe-Tcl is as follows:

> after, bind, button, canvas, checkbutton, destroy, entry, focus, frame, grab, label, lineto, listbox, menu, menubutton, message, moveto, option, pack, place, radiobutton, scale, scrollbar, selection, text, tk, tk_bindForTraversal, tk_firstMenu, tk_getMenuButtons, tk_invokeMenu, tk_mbButtonDown, tk_mbPost, tk_mbUnpost, tk_menuBar, tk_menus, tk_nextMenu, tk_nextMenuEntry, tk_traverseToMenu, tk_traverseWithinMenu, tkwait, toplevel, update, winfo, wm

5. Extensions to the Safe-Tcl Environment

All Safe-Tcl extensions are handled by a single Safe-Tcl procedure, SafeTcl_untrusted_eval:

SafeTcl_untrusted_eval -- "`SafeTcl_untrusted_eval command arguments...`". This command may be used by a Safe-Tcl program to communicate with the TRUSTED Tcl interpreter in the same process. SafeTcl_untrusted_eval makes all substitutions and evaluates all of the arguments in the untrusted environment. It then passes its arguments on to the procedure "untrusted_eval" in the trusted interpreter. The trusted interpreter’s untrusted_eval procedure will decide whether not the given command and arguments are safe to evaluate, and, if they ar deemed safe, will return the result of the evaluation, and will otherwise generate an error. The untrusted_eval procedure will always set the global variable SafeTcl_downgraded_cmd in the untrusted interpeter. Normally, it will set it to the empty string, but in the event that an error was generated because the command was deemed unsafe, but there is a "downgraded" version of the same command that would be considered safe, this command will be placed in the SafeTcl_downgraded_cmd variable. A safe-tcl program could thus use the Tcl "catch" facility to catch errors and execute the downgraded command if desired.

The actual decision about what can and cannot be evaluated by SafeTcl_untrusted_eval is made by the untrusted_eval procedure in the trusted interpreter, which is discussed in the following section.

This mehanism makes possible a wide variety of extensions to SafeTcl. For example, the core Safe-Tcl language does not include a library mechanism, but a library mechanism could be implemented with the careful extension of the untrusted_eval mechanism.

6. Recommended Extensions to the *Trusted* Tcl Interpeter

In order to permit the core Safe-Tcl language to have extensibility and minimal support to sending message and generating paper output, it is necessary that certain procedures be added to the trusted interpreter. Note that these commands should NOT be added to the Safe-Tcl interpreter, as this would not be considered safe. (One could imagine a mail message that automatically generates hate mail in the recipient’s name, or a message that maliciously wastes printer resources or sabotages a programmable printer.)

The following procedures should be included in the TRUSTED Tcl interpreter that is accessible to Safe-Tcl programs via the extension mechanisms previously described:

MIME_sendmessage -- `"MIME_sendmessage -to <addrlist> -subject <string> -body <body> -cc <addrlist> -auxheader <name> <value>"`. This command may be used to send a message. It takes a variable number of key/value arguments, three of which are required. The required "-to <addrlist>" argument specifies a string containing one or more electronic mail addresses in Internet-standard (RFC 822) format (e.g. with commas separating multiple addresses). The required "-subject <string>" argument specifies the subject of the mail being sent. The required "-body <body>" argument specifies the mail body, which is the value returned by a SafeTcl_makebody call. The optional "-cc <addrlist>" argument specifies a CC address list, in the same format as the -to addrlist. Finally, the optional "-auxheader <name> <value> argument, which may appear multiple times in a single call to MIME_sendmessage, specifies auxilliary headers which may be added to the mail message. For example, "-auxheader Reply-to: bgates@microsoft.com" will cause a header line

"Reply-to: bgates@microsoft.com" to be added to the message. MIME_sendmessage should return 0 on successful mail delivery, and generate an error if the mail cannot be sent. It is acceptable, but not m andatory, in the "activation" evaluation-time or other interactive contexts, to offer the user the opportunity to edit such mail before it is delivered.

MIME_printtext -- "`MIME_printtext txt`". This command may be used to send plain textual data to a locally-available printer. It takes one argument, the text to be printed. MIME_printtext should return 0 on successful submission of the print job, and generate an error if the material cannot be printed.

untrusted_eval -- "`untrusted_eval evaluation-time command arguments...`". This procedure is invoked in the TRUSTED Tcl interpreter whenever the SafeTcl interpreter in the same process evaluates the SafeTcl_untrusted_eval command. Before this invocation, SafeTcl_untrusted_eval makes all substitutions and evaluates all of the arguments in the untrusted environment, and adds in the evaluation-time parameter based on the context in which the Safe-Tcl program is being evaluated (either "activation" or "delivery"). Other than evaluation-time, the parameters for untrusted_eval are the same as for SafeTcl_untrusted_eval except that all substitutions and evaluations have already been performed. The implementation of untrusted_eval is the key both to all extensions of Safe-Tcl and to the integrity of the safety mechanisms designed into Safe-Tcl, and is discussed in detail in Appendix D.

eval_in_safetcl -- "`eval_in_safetcl program`". This command evaluates the given Tcl program in the untrusted environment. This gives code written in the trusted environment -- and particularly implementations and extensions to untrusted_eval -- access to the current state of the untrusted interpreter.

MIME_savemsg -- "`savemsg type ?destination?`". This command appends the message to either a mailbox or a folder, as specified by the first parameter, which should be either "mailbox" or "folder". If the second parameter is the empty string, the recipient's default mailbox or folder is used. Note that the behavior of this command will be highly implementation-dependent, and may also be user-customizable to deal with different mailbox and folder formats.

7. Notes To Implementors

Implementation details are usually considered beyond the scope of a specification such as this one. However, the extremely sensitive nature of a Safe-Tcl interpreter, and the safety issues entailed in the implementation of such an interpreter, suggests that some advice to implementors might be useful here.

1. Be careful how you implement any user interface code that asks for confirmation of potentially dangerous actions, e.g. in MIME_sendmessage or MIME_printtext. In particular, such code should always be written in the trusted interpreter, to prevent hostile programs from "reverse engineering" your implementation and short-circuiting the confirmation code. (A more radical alternative is to prohibit the use of the Tcl primitives proc or rename to redefine any built-in Safe-Tcl primtives or any Safe-Tcl procedures defiined for the confirmation process itself.)

2. A Safe-Tcl interpreter should ensure that there is ALWAYS some indication on the screen that an untrusted program is being run. A suggested mechanism is to reserve, as a "status line" the top or bottom line of each terminal or window in which Safe-Tcl is running. That line should indicate that the program is not to be trusted with sensitive information. This will help to prevent a clever Safe-Tcl program from fooling the user into supplying a password, e.g. by spoofing a login program.

3. A Safe-Tcl interpreter that runs on a video display terminal or terminal emulator should beware of permitting a Safe-Tcl program to send escape codes to the terminal. Some terminals can be programmed, using binary escape codes, to send data to the terminal which is then sent back to the host computer, and might be used to "break out" of the Safe-Tcl environment. Safe-Tcl interpreters that run on such terminals might wish to render into printable characters all lines that are displayed with such primitives as SafeTcl_displaytext and SafeTcl_displayline, thus inhibiting the transmission of raw escape codes to the terminal.

4. Special care must be paid to all aspects of untrusted_eval, as discussed in appendix D.

Security Considerations

Active messaging, in which programs are sent through the mail to be evaluated automatically or semi-automatically on behalf of the recipient, is an area fraught with potential security problems. Accordingly, the most important aspect of the design of the application/Safe-Tcl MIME type was the care that was paid to defining an active messaging language that was capable of safe implementation.

Despite this care, there remain two potential pitfalls that could cause the application/Safe-Tcl type to become a vehicle for security problems, and all implementors and administrators should be aware of these problems:

1. Implementation bugs. No matter how much care is paid to the design of a language for active messaging, interpreters of such a language will remain as vulnerable to security-compromising bugs as any other network services. Just as the Internet worm was able to exploit bugs in the finger and sendmail programs, so too bugs in a Safe-Tcl interpreter might be exploited by future sociopaths. This does not argue against the concept of Safe-Tcl, but suggests that system administrators must be clear about the difference between a safe language and a correct implementation of a safe language, and should INSTALL ONLY THOSE SAFE-TCL INTERPRETERS THAT COME FROM EXTREMELY TRUSTED SOURCES.

2. Poorly-conceived extensions or supersets. Safe-Tcl is, by design, a highly restricted language. It is very easy to add extensions that will make it more powerful, but such extensions can easily have the effect of undoing all the security-consciousness that went into the original design of the language. (Such extensions won't exactly promote interoperability, either, another good reason to avoid them.) Administrators should beware of installing software that claims to implement a superset of Safe-Tcl, as the basic Tcl language is not itself safe for sending through the mail. Users and administrators alike must exercise care in the use of the Safe-Tcl extension mechanisms. When in doubt, simply don't install an extension to Safe-Tcl.

Authors' Addresses

For more information, the author of this document may be contacted via Internet mail:

Nathaniel S. Borenstein
MRE 2D-296, Bellcore
445 South St.
Morristown, NJ 07962-1910
US

Phone: +1 201 829 4270
Fax: +1 201 829 5963
Email: nsb@bellcore.com

Marshall T. Rose
Dover Beach Consulting, Inc.
420 Whisman Court
Mountain View, CA 94043-2186
US

Phone: +1 415 968 1052
Fax: +1 415 968 2510
Email: mrose@dbc.mtview.ca.us

Acknowledgements

This document reflects the input and ideas of many researchers and developers who have worked in the field of active messaging over the last two decades. Particularly helpful in the drafting of this document have been Dave Crocker, Karl Lehenbauer, John Ousterhout, Rich Salz, and Allan Shepherd.

Special thanks are due to John Ousterhout, for the design and implementation of Tcl and Tk.

References

[RFC-MIME] Borenstein, N., and N. Freed, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1341, June, 1992.

[ATOMICMAIL] Borenstein, Nathaniel S., "Computational Mail as Network Infrastructure for Computer-Supported Cooperative Work", in Proceedings of CSCW ’92 Conference, Toronto, Ontario, November, 1992.

[TCL] Ousterhout, John, An Introduction to Tcl and Tk. Addison-Wesley, 1993 (to appear).

[EM-MODEL] Rose, M., and N. Borenstein, "A Model for Enabled Mail (EM)", draft in preparation, May, 1993.

Appendix A: Examples

A. 1. Usage Example: Delivery-Time Enabled Mail

Here is a brief example of a program that might be evaluated during the delivery evaluation-time.

```
Content-Type: application/safe-tcl;
      evaluation-time=delivery

SafeTcl_untrusted_eval \
MIME_sendmessage \
      -to $SafeTcl_originator \
      -subject "Delivery Notification for $SafeTcl_recipient" \
      -body [SafeTcl_makebody "text/plain" \
            [SafeTcl_getheader "Message-ID"]]
```

This simply sends a message to the originator indicating that the incoming message crossed the delivery slot for the recipient.

A. 2. Usage Example: Activation-Time Enabled Mail

Here is a brief example of a program that might be evaluated during the activation evalution-time.

```
Content-Type: application/safe-tcl;
      evaluation-time=activation

proc ordershirt {} {
    SafeTcl_untrusted_eval \
        MIME_sendmessage  -to tshirts@whitehouse.gov \
            -subject "Shirt request" \
            -body [SafeTcl_makebody "text/plain" \
                         [SafeTcl_getline \
                              "What size t-shirt do you wear?" \
                              "medium"]]
    exit
}

global SafeTcl_InterfaceStyle
if {[lsearch $SafeTcl_InterfaceStyle "Tk3.2"] >= 0} {
    message .m -aspect 1000 \
      -text "Click below if you want a free Bill Clinton t-shirt!"
    button .b -text "Click here for free shirt!" \
        -command ordershirt
    button .b2 -text "Click here to exit without ordering" \
         -command exit
    pack append . .m {pady 20} .b {pady 20} .b2 {pady 20}
} else {
    set ans [string index \
                     [SafeTcl_getline  \
                     "Do you want a free Clinton t-shirt? "  \
                     "No"] \
                   0]
    if {$ans == "y" || $ans == "Y"} {
        ordershirt
    }
    exit
}
```

The above program (which will use the Tk3.2 interface style if it is available, and will otherwise use the default style) will offer the user the opportunity to order a free t-shirt.

Appendix B: Summary of Safe-Tcl Primitives

**** Need to make a table, with domains of applicability for each.

Appendix C: User-Friendly Renderings

When displaying an RFC 822 address, a user-friendly rendering may be preferred. In practice, an RFC 822 address usually appears in one of these two forms:

phrase (comment) <local@domain>
local@domain (comment)

Although the algorithm for generating such a rendering is implementation specific, the following is recommended.

1. if a phrase is present, return that as the user-friendly rendering; otherwise,

2. if at least one comment is present, take the first one, remove the parenthesis, and return that as the user-friendly rendering; otherwise,

3. if the local-part does not appears to be in the syntax defined by RFC 1327 (e.g., a collection of /key=value/ strings), then return the local-part as the user-friendly rendering; otherwise,

4. if a string of the form

   /PN=value/

   is present in the local-part, then replace any dots in "value" with spaces and return that as the user-friendly rendering; otherwise,

5. if a string of the form

   /S=value/

   is not present, then return the local-part as the user-friendly rendering; otherwise,

6. if a string of the form

   /G=value/

   is present , then return "G-value S-value" as the user-friendly rendering; otherwise,

7. return "S-value" as the user-friendly rendering.

Appendix D: Recommendations Regarding untrusted_eval

The simplest possible implementation of untrusted_eval would be to simply evaluate the given command with the given arguments. This would be simple, but would also open up an enormous security hole, effetively granting each mail reader complete computational access to the environment of the recipient. This is not considered acceptable under any circumstances.

The safest possible implementation of untrusted_eval is to always return an error. Unfortunately, this denies access to some functionality that is expected to be vital to Safe-Tcl programs.

Therefore this appendix specifies, in algorithmic form, a recommended minimal implementation of untrusted_eval. Implementors, administrators, and users alike are all cautioned to think long and hard about any further extensions they make to untrusted_eval. It should always be borne in mind that this code might be evaluated for mail sent by hostile senders, running with the identity and privileges of unwary recipients. In particular, NETHER untrusted_eval NOR ANY PROCEDURE CALLED BY IT SHOULD EVER EVAUATE ITS ARGUMENTS AS TCL EXPRESSIONS OR PROGRAMS.

What follows is a recommended default algorithm for untrusted_eval.

```
SET SafeTcl_downgraded_cmd to ""
IF (command == "MIME_sendmessage") then
   IF (evaluation-time == "delivery") then
      IF (to-and-cc-names == Tcl_Originator) then
```

```
         EXECUTE-COMMAND
      else
         SET SafeTcl_downgraded_cmd to be the
            original command with the
           -to argument set to Tcl_Originator
           and -cc argument set to ""
         GENERATE-ERROR
      endif
   else
      if (User inspects mail and OK’s it) then
         EXECUTE-COMMAND
      else
         GENERATE-ERROR
      endif
   endif
else IF (command == "MIME_printtext") then
   IF (evaluation-time == "delivery") then
      GENERATE-ERROR
   else
      if (User inspects text and OK’s it) then
         EXECUTE-COMMAND
      else
         GENERATE-ERROR
      endif
   endif
else
   GENERATE-ERROR
endif
```

In other words, at activation time, the user is given a chance to OK sending mail or printing information, but all other actions are rejected. At delivery time the only action accepted by untrusted_eval is sending mail to Tcl_Originator only.

Because the envelope information on which Tcl_Originator is based may be forged, it is further recommended that, at delivery time, any mail which is sent by this mechanism to Tcl_Originator should be sent with a "From:" address that clearly indicates that the reply was sent by an automatic agent, rather than a human being. Thus, instead of generating mail with

```
From: Nathaniel Borenstein <nsb@bellcore.com>
```

the mail should be generated with a From field such as

```
From: Mail Delivery Agent for Nathaniel Borenstein
      <nsb@bellcore.com>
```

Note also that it is better to put this information in the RFC 822 phrase than in a comment, since comments in mail addresses are sometimes discarded.

It is also prudent for implementations to save the delivery-time Safe-Tcl messages that generate such mail for a few days, to be used for investigations into abuses of the facility.

Things To Do Before Publishing

-- Specify at least one encryption algorithm to use (enlist Peter Winkler?)

-- Flesh out appendix B

Table of Contents

1. Overview 1
2. The application/Safe-Tcl content-type 2
3. The multipart/enabled-mail content-type 3
4. The Safe-Tcl Language 4
4.1. The Core Safe-Tcl Language 4
4.2. Universal Safe-Tcl Functionality 5
4.3. Additional Messaging Functionality 6
4.4. Additional Delivery-Time Functionality 10
4.5. Additional Activation-Time Functionality 11
4.5.1. User Interaction Models: SafeTcl_InterfaceStyle 11
4.5.2. Generic User Interaction 12
4.5.3. X11 Interaction: Interface Style Tk3.2 13
5. Extensions to the Safe-Tcl Environment 14
6. Recommended Extensions to the Trusted Tcl Interpeter 15
7. Notes To Implementors 17
Security Considerations 18
Authors' Addresses 19
Acknowledgements 19
References 20
Appendix A: Examples 20
A. 1. Usage Example: Delivery-Time Enabled Mail 20
A. 2. Usage Example: Activation-Time Enabled Mail 20
Appendix B: Summary of Safe-Tcl Primitives 22
Appendix C: User-Friendly Renderings 22
Appendix D: Recommendations Regarding untrusted_eval 23