The following paper was originally published in the
Proceedings of the First USENIX Workshop on Electronic Commerce
New York, New York, July 1995.

Safe Tcl: A Toolbox for Constructing Electronic Meeting Places

Jacob Levy and John Ousterhout
Sun Microsystems Laboratories

A Safe Tcl Toolkit For Electronic Meeting Places

(Extended Abstract)

Jacob Y. Levy and John K. Ousterhout

jyl@eng.sun.com, ouster@eng.sun.com

Sun Microsystems Laboratories

2550 Garcia Ave. M/S UMTV29-232

Mountain View, CA 94043

Abstract

Electronic commerce needs electronic meeting places to conduct business. To be useful, such meeting places must be safe for all participants and for hosts (owners of places). In this paper we discuss safety issues for participants and hosts. We then describe a system we are building, Safe Tcl, that will allow the construction of electronic meeting places with a range of safety properties. Safe Tcl has two attractive properties. First, it uses a simple security model based on ``padded cells'' that allows participants to coexist and interact safely. Second, Tcl makes it easy to integrate the numerous facilities required in an electronic meeting place such as integrity verification and authentication.

1 The Problem Of Safety

Electronic commerce, like human commerce, needs “safe places” where participants can meet to conduct business. The safety of a place can be measured by e.g.:

- Whether the host is protected against malicious or erroneous actions of individual participants.

- Whether participants are protected from each others’ malicious or erroneous actions, and whether participants can be coerced by other participants to release, against their free will or without their knowledge and agreement, valued resources they carry with them (including information).

- Whether participants are protected from the actions of the host, both malicious and erroneous.

Tools for constructing safe meeting places for electronic commerce will become increasingly important as electronic commerce becomes more widely used. We believe that the basic security mechanisms for privacy, authentication, integrity checking and non-repudiation are relatively well understood. However, how to combine these mechanisms into higher level policies is less clear. Therefore, at this stage it is useful to create tools that allow experimentation and rapid prototyping as well as the construction and deployment of completed electronic commerce systems. Experience from human based commerce systems may be a useful guide in constructing electronic meeting and in choosing which tools to provide. We show how each safety problem identified above can be addressed in a computational context by drawing parallels from current common practice.

Currently, the human host and participants are protected from malicious intent of a participant by ensuring that no coercion tools (weapons etc.) are brought into the meeting place. Without a means for coercion there is no way for one participant to force another participant to release valued resources (such as the $1 million they are carrying in a briefcase) or information they own. Also, without means for coercion, there is no way for one participant to coerce the host to deny service or subvert its service to another participant. The equivalent in computational systems is to place each participant (or group of mutually trusting participants) in a separate environment (``padded cell''), thus restricting their ability to manipulate the state of other participants or the host. Functionality in an environment is restricted to remove any method for a participant inside the environment to harm another participant outside the environment. To enable communication between participants, environments are extended with controlled communication channels that only allow legitimate communication.

Protecting a human participant from the host is currently achieved through insurance and liability based mechanisms. Upon entry into a meeting place the participant is at risk of being coerced by the owner of the place to divulge information or to part with valued resources. These risks can be ameliorated by insurance or liability shifting arrangements, or by bonding. Similar mechanisms can be implemented in an electronic commerce system: a third party can offer insurance covering aspects of electronic business such as compromise of a transaction or participant owned resources by a host. Since these mechanisms are based on authentication, integrity checking and privacy, a system that provides access to these building blocks suffices.

2 Tcl And Tk

Tcl [1] is an interpreted scripting language originally developed by Ousterhout at UC Berkeley, widely used for rapid application development. Tcl is easy to extend and one of its most popular extensions, Tk, is very popular for rapidly constructing highly interactive GUI components for programs.

Tcl programs are portable between systems on which a Tcl interpreter is implemented. Tcl programs are strings, as is all data manipulated by a Tcl program. Tcl programs can thus be input data for other Tcl programs, and Tcl programs can be generated by Tcl programs and executed on the fly or transported in textual form. The global data state of a Tcl execution can be obtained and saved as a Tcl script, which when executed, restores the state to what it was when the script was created.

3 Borenstein's And Rose's Safe Tcl

Safe Tcl is an extension of Tcl created by Borenstein and Rose [2] to allow safe execution of programs transported by email. It takes advantage of the ability of Tcl to support multiple totally independent Tcl interpreters within a single application process. Each Tcl interpreter provides a separate name space for variables and procedure definitions and a separate call stack for procedures. The command set of each interpreter can be tuned to include exactly the commands desired.

Borenstein and Rose's Safe Tcl provides two interpreters: one is unrestricted, and the other's command set has been restricted to exclude unsafe facilities such as access to files and creating subprocesses. Incoming scripts can be executed in the restricted interpreter without fear of damage to the environment. The application's unrestricted intepreter can implement safe methods for access to unsafe facilities and export these to the restricted interpreter by ``declaring them harmless''. The overall effect is much like the common separation in operating systems between kernel space and user space: the unrestricted interpreter corresponds to kernel space, the restricted interpreter corresponds to user space, and the features that are declared harmless parallel system calls. Safe Tcl is more flexible than the traditional kernel-user space mechanism because features exported to the restricted interpreter can be modified at run-time.

4 The New Safe Tcl

The Safe Tcl extension created by Borenstein and Rose is useful for executing simple untrusted scripts. However, because it allows only one untrusted interpreter to exist at a time it is not powerful enough to host mutually suspicious scripts and it does not provide communication mechanisms between such scripts. Also, it is intimately tied to email as a transport mechanism, and it does not provide security features such as authentication or integrity checks.

At SunLabs we have generalized Borenstein and Rose's ideas to allow for multiple restricted interpreters, called ``slaves'', under the control of a ``master'' interpreter. A slave can be master for other slaves, forming a hierarchy of interpreters. A master interpreter can configure the command set of its slaves to be a proper subset of the features that it itself has access to. A master can execute arbitrary scripts in the slaves. A slave has no access to its master except as described below.

We have also generalized the ``declare harmless'' mechanism of Borenstein and Rose's Safe Tcl into a more general ``alias'' mechanism for communicating between interpreters. An alias creates a link between two interpreters such that when a particular command is invoked in one, the source interpreter, another command is executed in the other, the target interpreter. The target command receives the arguments of the source command, and the result of the target is returned as the result of the source command. A master interpreter can create aliases between its slaves and itself, which are analogous to the ``declare harmless'' mechanism of Borenstein and Rose. A master can also create aliases between any of its slaves in order to allow them to communicate directly.

Our third generalization will be to incorporate authentication techniques [3, 4] into Safe Tcl. A master interpreter will be able to authenticate scripts before executing them in slave interpreters. Based on the trust assigned to the script's author, the master can customize the facilities available to the slave. A master intepreter will also be able to authenticate aliases individually, so that a script can include both untrusted parts that must execute in a slave interpreter and trusted parts that form aliases to be executed in the master.

Our final generalization is to separate how to execute untrusted scripts from how they are transported. We feel that transport issues are not a part of the problem to be solved by Safe Tcl and that we should not innovate here. We will connect Safe Tcl to email, HTTP and other mechanisms to allow participants to enter and leave places through a variety of mechanisms. Also, how participants find and name each other is not a problem solved by Safe Tcl. Safe Tcl will be able to use a wide choice of external naming schemes. Of course, for Safe Tcl to be widely useful, eventually a small set of core transports and naming schemes must be supported. However, at this time we are building for experimentation and thus we will provide no built in naming or communication schemes.

Safe Tcl offers two advantages in setting up electronic meeting places. First, its security model is very simple and makes it easy to isolate mutually suspicious participants. Safe Tcl is based on the kernel-user model that has been used successfully for thirty years in timesharing systems, and it provides considerably more flexibility. Safe Tcl is implemented as part of the Tcl interpreter, so it offloads many security concerns from the operating system kernel to the Tcl interpreter. For example, Safe Tcl ensures that Tcl programs can not modify other programs' address spaces even on insecure systems such as some personal computers' operating systems which do not provide this assurance.

The second advantage of Safe Tcl stems from the Tcl scripting language, which makes it easy to integrate a variety of components. Electronic meeting places for commerce must include a large number of mechanisms from electronic cash to encryption to databases. With Tcl it is possible to extend the system facilities with code written in C or to write Tcl scripts to glue existing components together. Tk allows electronic participants to interact with humans as well as with other participants. Since Tk interfaces are Tcl scripts, they can be part of a participant's script.

5 Electronic Meeting Places in Safe Tcl

Separate interpreters allow mutually suspicious and antagonistic participants to be in one meeting place while preventing them from harming each other or obtaining access to each other's private data. Because they are executing in separate interpreters from that of the host, participants can not harm the host. The host controls the creation of communication channels that allow participants to communicate with each other and with the host. These channels can be audited to provide accountability and non-repudiation. The host can also grant controlled access to a participant to specific safe uses of unsafe functionality on an as-needed basis. Using resource allocation policies implemented with tools provided by Safe Tcl, a host will be able to limit resource usage by scripts executing in its slave interpreters to prevent denial of service attacks or resource hogging.

The roles of host and participant are fluid. Using the tools provided by Safe Tcl a participant can construct a temporary meeting place for its own use while it is visiting a place that allows this. It can control which participants are admitted to its place and all aspects of their computation while the participants are within its own place.

Protecting participants from a host is a difficult problem. It is fundamentally impossible to protect a program from malicious behavior on the part of the engine that executes it. The host controls the engine (the Tcl interpreter) that executes the particpants, so there is no way to protect participants from a malicious host. A viable approach is for participants to ``avoid bad neighborhoods'' by only entering meeting places that are known to be trustworthy. For example, trustworthy meeting places could be independently certified and identify themselves with well-known public keys. A would-be participant can issue a challenge to the host with an advertized public key and only enter the place when the challenge is satisfied. Also, participants can insure themselves or require hosts to be bonded. Safe Tcl can be integrated with the needed mechanisms to enable this scenario.

6 Remaining Open Problems

Here are some remaining open problems that Safe Tcl currently does not address:

How to implement electronic currencies. There are several well known approaches to solving this problem [5,6]; we believe that interfacing Safe Tcl to one or more of these is simple, and we will do so in a future version of Safe Tcl.

Persistence and recoverability of participants. Safe Tcl does not afford participants protection against crashes of a host; all state and information carried by a participant will be lost if a host crashes or disallows the computation of a participant to complete. This is a hard problem, a sub-problem of protecting a participant from a place's host, and will be solved in that context.

Resource Controls and Denial of Service Attacks. Safe Tcl itself does not provide any mechanism for accounting for resources used by an application, and for preventing an application from mounting a denial of service attack through excessive resource consumption. However, the clean separation between applications through the use of separate interpreters affords a clean starting point for building a resource accounting system. A future version of Safe Tcl will explore these issues.

7 Availability

Safe Tcl is available by anonymous FTP. Retrieve the file ftp.smli.com:/pub/tcl/stcl0.2.tar.gz. The software is also available at the Tcl archive managed by Alcatel Inc., at ftp.aud.alcatel.com:/tcl/extensions/stcl0.2.tar.gz.

8 References

[1] J. K. Ousterhout, "Tcl And The Tk Toolkit", Addison-Wesley, (1994).

[2] N.S. Borenstein and M. Rose, "Safe Tcl", available at ftp://ftp.fv.com/pub/code/other/safe-tcl.tar.Z.

[3] S.Garfinkel, "Pretty Good Privacy", O'Reilly & Associates, Inc., (1995).

[4] W. Diffie and M. Hellman, "Privacy and Authentication", IEEE Proceedings, pp. 397-427, (1979).

[5] D.Tygar, "NetBill: An Internet Commerce System Optimized for Network Delivered Services", IEEE CompCon '95, (1995).

[6] D.Chaum, "Online Cash Checks", Advances in Cryptology EUROCRYPT '89, pp. 288-293 (1989).