dotNyet

by Mike Doyle, Cyndy Lilagan, Steve Landers, and Steve Huntley

Iomas Research LLC

In recent years, the notion of "cloud computing" has become all the rage. Despite its recent wave of popularity, distributed computing has been with us in various forms for more than 20 years.

Distributed computational systems are all about allowing one machine on a network to tap into the computational resources of one or more other systems connected to the same network. Currently popular frameworks for this range from commercial systems such as Microsoft's .NET to large open source projects such as the Globus Grid. One characteristic common to these systems is the relatively high level of complexity inherent in their architectures, deployment requirements, and development resources. The dotNyet system is a Tcl-based response to the complexity of these other systems. dotNyet exploits a handful of uniquely-Tcl-empowered technologies to provide a simple yet powerful platform for the easy development and safe deployment of secure platform-agnostic distributed Tcl applications over untrusted network environments.

Our overall objective in the dotNyet project has been to extend the advantages of web-based distributed application deployment to stand-alone applications, while adding more flexibility as well as fine-grained control over authentication, confidentiality and sandboxing of mobile executable code.

The architecture of the dotNyet system takes advantage of several powerful Tcl-based tools. These packages tap into capabilities unique to the Tcl platform to provide flexible means by which the system: 1) creates custom safe interpreters for mobile code execution, 2) provides cryptographic services for authentication and confidentiality, and 3) exploits data communication, synchronization and RPCs between and among clients and servers.

PoliTcl

A common requirement for distributed application frameworks is that of providing a safe way for systems which host remotely-provided mobile code to share that executable code without endangering the integrity of the hosting computer. PoliTcl is a system to provide such a capability to Tcl applications. The PoliTcl package allows pre-defined policies to be associated with particular modules of mobile code, so that those modules, once authenticated, can be provided with discrete custom-tuned execution spaces, or "sandboxes," which provide limited access to the resources of the host computer. These sandboxes can be very finely tuned, via the policy mechanism, so that the host computer can be protected against the dangers of malicious code, while allowing authenticated trusted code to have access to powerful capabilities. In the dotNyet system, PoliTcl provides capabilities for stand-alone Tcl programs similar to the policies package provided by the Tcl browser plugin.

Policies are associated both with particular modules of dotNyet mobile code and with specific ranges of data, so that the execution of the code can take place under safe and trustworthy conditions, and so that access to various parts of the data can be fine tuned for various users in a workgroup

PoliTcl was extracted and abstracted from the Tcl Plugin package. [1] A policy mechanism is highly desirable for a browser plugin as a basis for establishing trust that a skeptical user can safely run applications in the plugin in a standard browser. This use case can be generalized to encompass a skeptical service provider establishing policies to allow access to unknown visiting users, or in a cloud computing environment to establish a network of trust among nodes. PoliTcl allows existing paradigms such as client-server or peer-to-peer to be redefined as negotiated relationships of trust. When combined with cryptographic authentication, as described below, the PoliTcl system provides a safe way to move code around the network and to share distributed resources unobtrusively.

CryptKit

As distributed computing models begin to be more widely deployed over the commodity Internet, robust tools for securing those applications against eavesdropping and attack become increasingly crucial. In order to provide Tcl programmers with an extremely easy-to-use and powerful tool for this purpose, the Cryptkit package has been developed as a dynamically loadable Tcl extension that provides full access to all of the cryptographic functions of the well-known Cryptlib toolkit. [2] This package allows dotNyet to tap into best-of-breed facilities for file encryption, communication channel encryption, data integrity certification and user/code authentication.

From the Cryptlib web site - "The Cryptlib security toolkit is a powerful security toolkit that allows even inexperienced crypto programmers to easily add encryption and authentication services to their software. The high-level interface provides anyone with the ability to add strong security capabilities to an application in as little as half an hour, without needing to know any of the low-level details that make the encryption or authentication work. Because of this, cryptlib dramatically reduces the cost involved in adding security to new or existing applications." [3]

Cryptkit brings these benefits to Tcl. In the dotNyet world, Cryptkit provides support for cryptographic signatures to authenticate both users and code, allow encryption of code, and secure communication channels.

Tequila

The dotNyet messaging and control layer for distributed processing is provided by Tequila. [4]
The Tequila technology is a novel approach to the area of distributed computing. From the Tclers' Wiki: "Tequila is a little Tcl server JCW started in 1999, which implements persistent shared arrays. With Tequila, you need no longer think in terms of communication: a Tcl array gets "attached" to the server, and from then on all clients doing so can read/write/unset items in it. This approach works quite well in combination with traces (and it's also built on traces and file events). When properly set up, you can builds apps as monolithic ones and later split them up with minimal changes."

In addition to the shared array facility, Tequila provides a robust RPC mechanism that allows an RPC client to remotely execute Tcl code on a target RPC server. Essentially, Tequila allows applications to treat the network as the equivalent of a data bus. Multiple client applications can connect to a Tequila server to share data and computations transparently, by "attaching" local arrays to Tequila "pools." When any client changes the value in an attached array, that change is instantly and automatically propagated to all other client applications attached to the same Tequila pool. Client applications can also propagate remote procedure calls and program events through the pool mechanism. This provides the software developer with an extremely easy to use tool for building highly sophisticated distributed applications with minimal effort.

dotNyet

These pieces are woven together in the dotNyet system to allow a safe and secure environment for Tcl-based applications to span multiple locations, with components that can move freely among these locations and tap into selected resources across the network. When integrated with API-enabled productivity applications, dotNyet can provide the glue to create workspaces that can span across locations, platforms, and vendors. For example, the Scatclogic demonstration application described below creates a single collaborative workspace that integrates spreadsheet applications from both OpenOffice and Microsoft to create a hyper- (as in hypermedia) dimensional spreadsheet, one that can dynamically recalculate among multiple users across the network.

Scatclogic

To help illustrate the dotNyet framework, we used it to tackle a real-world problem area. How does one create high-level integrated systems using office productivity applications that famously don't like to talk together? The dotNyet technologies provided us with an intriguing way to approach this problem.
We've named the result of these efforts the Scatclogic approach.

The name loosely derives from "scattered Tcl logic," although the way we came up with the name was a bit more sophomoric. As one of the team summarized: "after throwing ideas against the wall to see what would stick, it's the one that floated to the top." (mixing our metaphors for a little scatological humor) In any case, the name stuck.

In a nutshell, the Scatclogic approach involves several layers of functionality:

- a code snippet is created within the native interface of your application of choice
- the snippet is then extracted via the API into the Tcl application layer
- it is signed by the appropriate cryptographic keys
- the snippet is then multicast out to all attached clients
- the snippet creator's public key is used to authenticate the author
- the appropriate policy is invoked - bound to the snippet creator's public key
    - one of the things that the policy mediates is who has permission to see, edit, or execute the code
- for all users who have permission to run the snippet, any tcl code in it is pre-processed, then the remaining (or output) snippet code is then inserted via the API into each target user's native application interface
- the code snippet is then executed by the application's native facilities

There are several implications to this methodology:

First of all, it allows a project leader to pull together teams of workers without being constrained by cross-vendor incompatibilities and tie-ins resulting from the use of a wide variety of office productivity applications across the team membership.

Secondly, Scatclogic allows the organization to leverage and build upon existing infrastructure investment, weaving together existing productivity applications into contiguous shared workspaces.

Finally, users can work with familiar tools, sharing the output of that work with others without being forced to conform to a single "corporate standard" word processor, spreadsheet, etc.

Scatclogic demo application: Tcl Between the Sheets

Probably the most common means by which laypeople in the workplace do their own programming is through the use of spreadsheet applications. Spreadsheets are programmed by editing formulas in cells which reference the values of other cells. It's an event-driven model, whereby the application detects that a change has been made to a cell, and then propagates the change through the linked series of cell references that cascade from the edited location.

In the late 1980s, researchers at Boeing Corp. pioneered the notion of linking across sheets with their BoeingCalc 3D spreadsheet package. Since then it has become commonplace to create cross-referential links between sheets running within the same container application. More recently, however, synchronization of sheets over networks has been demonstrated via specialized spreadsheet applications. Most users, though, don't use such specialized applications. Office workers tend to use the spreadsheet applications that have been installed as a part of one of the popular office suites. These applications are typically not designed to facilitate synchronization of sheets between distributed users of the same brand of office suite, much less communication between applications from competing brands.

To take the Scatclogic approach out for a spin, we decided that the spreadsheet cross-brand-app synchronization problem would be an ideal testing ground for the system. Keeping consistent with our demonstrated tongue-in-cheek naming tendencies, we decided to christen this new system "Tcl Between the Sheets," or TBS, for short. (note: the alternative name, "Secure Tcl Distribution," was also suggested, but ... well, let's just say that TBS was better)

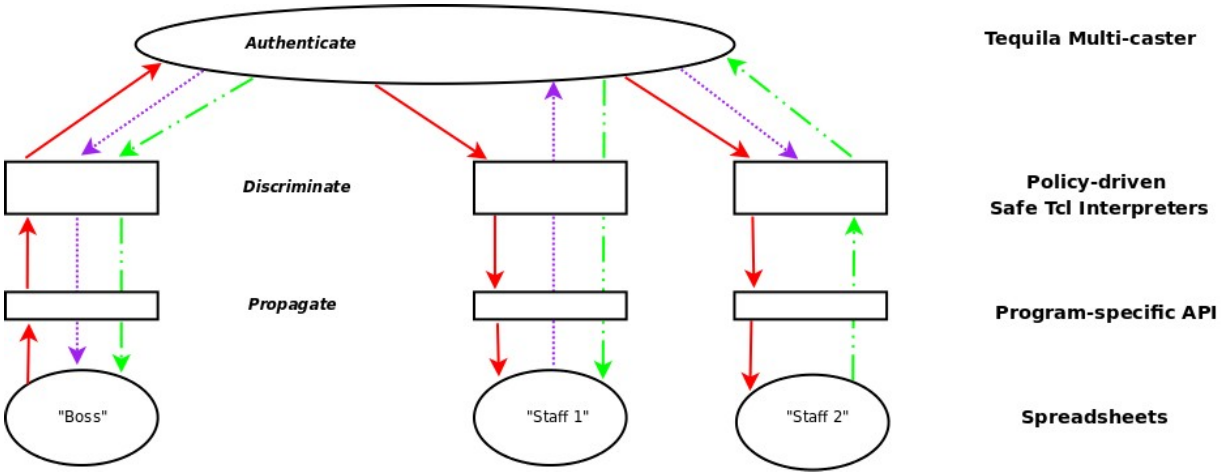

Figure 1: The "Tcl Between the Sheets" architecture

Much of the TBS functionality results as a consequence of the system's security model. This model includes the following:

Each TBS sheet-based application is a project.

A project can have one or more sheets. The author of a project is the owner of that project, and becomes the administrator of the various permissions relating to the project. Each project has an associated policy key pair, consisting of a private key and a group key. The group key is actually a public key, using PKI parlance, but it is kept as a shared secret among group members.

The project comprises a set of sheet ranges. This can be a disjoint set of cells spanning multiple sheets. Each range has a set of associated 5 public/private key pairs. This set includes keys that correspond to the owner (o), group (g), read permission (r), write/edit permission (w), and execute permission (x).

The author starts a project, and sets up a set of ranges within the project. To discuss this, we will consider, for example, 3 users, two using OpenOffice (users 1 and 2) and one using Excel (user 3).

The author sets permissions for each range, by selecting r,w,x permission levels per user.

The r permission defines who can see the cell, however it is possible that one might have execute permission without read permission, in certain special cases.

The w permission defines who can edit cells in the range, and the x permissions define where the cell executes. Judicial use of the x permission can lead to very powerful possibilities. For example, one could use this feature to create a parallel-processing spreadsheet, which would be just one specific instance of this general model.

As a result of the definition of the ranges and associated permissions, this data is uploaded to the server where a range policy is created. When a user subsequently logs into the server to access a project sheet, the server distributes public keys to the user for all ranges for which the user has access rights. The TBS system then administers the policies for each user accessing the project, according to the layered functionality scheme described above. In the example spreadsheet referred to above we will illustrate synchronous use of the spreadsheets by the three users simultaneously, although the TBS system follows an asynchronous model, just as a spreadsheet does, indicating error conditions in appropriate cells, until the sheet is developed enough to resolve all errors.

In order to use the example sheet, for example:

- User 1 would have already created the project and set permissions for the range's cells
- User 2 gets r,x permission and user 3 gets r,w,x permission
- These permission specifications determine the ranges
    - a policy definition would then be automatically generated (or modified) and stored on the server
- The various users open their spreadsheet applications, launch the TBS client applications, and log into the TBS server, then open the project.
- User 1 enters a formula into a cell.
- The formula is encrypted by user 1's private owner key, and the policy's group (public) key.
- The server then decrypts the formula code snippet, and authenticates it using the author's public key.
- The server then re-encrypts the snippet using the policy's private key and the group's "r" sub-key, then signs the snippet with the remaining group sub-keys (x,w). (note: If there is an "x but not r" permission associated with the range, the execution is treated specially. In such a case, the snippet must be pure Tcl code, which is executed on the server, rather than in a spreadsheet.)
- The code is then multicast to the project users via the Tequila shared array mechanism.
- Each user's client application then examines the relevant keys and signatures to determine what actions it should perform on the code, based on what permissions are enabled (determined by which keys are locally stored).
- The appropriate actions are performed on the code snippet
- Finally, the result of any Tcl pre-processing is then inserted into the appropriate cells in the native spreadsheet. If it is a valid spreadsheet formula, and if the correct permissions apply, the spreadsheet then executes the formula, and propagates any relevant recalculations to any linked cells in the rest of the spreadsheet.

The result of all this is that when user 1 enters a spreadsheet formula into a cell, it propagates to user 2's and user 3's spreadsheets. When user then 3 modifies the formula, all three users' sheets update. When user 2 tries to change the same cell, on the other hand, it reverts back to the original value.

So, now let's think about what just happened. A user in OpenOffice just entered a value in a spreadsheet on his desktop. Another user across the network, looking at the same spreadsheet file in Excel, sees the sheet automatically recalculate with the updated value. The Excel user changes the formula, and the OpenOffice sees the new calculation on his screen. All of this occurs with no changes being made to either OpenOffice or Excel. That is the power of Tcl.

Other Applications

The TBS example can be easily generalized to allow many other application types. One good example might be to use the approach in developing a "poor man's grid computer." By defining the x permission to distribute the execution locations for a large number of ranges to remote sites, and then linking together the ranges through cell interdependencies, creating a workable compute grid could be relatively straightforward.
In another application, combining TBS with a next-generation spreadsheet application such as Moodss could allow the creation of spreadsheet-based applications capable of monitoring and managing entire factories.

On the other hand, there is no reason that the office productivity application extended through the TBS approach need be a spreadsheet. It could even be a word processor or slide show application. For example, if it were the OpenOffice Write word processor application, then teams of collaborators could work on their own desktops to collaboratively edit a text document, with the “ranges' corresponding to pages which are shared among team members. This would essentially turn the native word processor into a wiki, but without the editing and formatting limitations so prevalent in the web-based wikis of today.

Returning to the grid computing example, it is intriguing to consider extending that model even further. If one were to, for example, combine TBS with Steve Landers' Klant system,[5] the users would be able to easily tap into remote X11-based applications to analyze and visualize the results of TBS calculations. Assuming adequate bandwidth between client and server, such a system would allow users with nothing more than a netbook to marshal vast computing and visualization resources in a rich collaborative network environment.

Shift Happens

There is often an overabundance of rhetoric in the technology literature about changing paradigms. Each minor technology improvement, it sometimes seems, is a portent of a change-the-world upheaval in the way people work. dotNyet started simply as an attempt to make it easier for people to be able to ship around executable code in a safe and effective way. From these humble beginnings we are beginning to glimpse bits of what might be a very promising future.

The thing about paradigms is... sometimes shift really does happen.

References

1. Tcl Plugin: http://www.tcl.tk/software/plugin/
2. Cryptkit: http://wiki.tcl.tk/13191
3. Cryptlib: http://www.cs.auckland.ac.nz/~pgut001/cryptlib/
4. Tequila: http://wiki.tcl.tk/1243
5. Klant: http://wiki.tcl.tk/12684