Mobile Agent Security and Telescript

Joseph Tardo and Luis Valente

General Magic, Inc.
420 North Mary Avenue
Sunnyvale, CA 94086

Abstract

Telescript is a software technology for building distributed applications using the mobile agent paradigm. Telescript mobile agents are migrating processes capable of being executed on any Telescript service host. Currently, Telescript incorporates safety and security features sufficient for deploying the first generation of mobile agent applications, subject to certain operating restrictions. The ultimate goal is the safe hosting of arbitrary, previously unseen programs, or "programming the network."

1.0 Introduction

Telescript is an interpreted, object-oriented language for writing mobile agents [1, 2, 3]. In common with other object-oriented languages, a Telescript program is basically a collection of hierarchically organized classes, with subclassing and multiple inheritance. However, Telescript also includes built-in classes that support pre-emptively multitasked, priority scheduled process objects. A single Telescript engine can simultaneously host processes on behalf of multiple users, and thus exhibits many of the security issues related to information sharing that arise in multi-user operating systems.

One interesting aspect of Telescript's process model is its built-in support for autonomous process migration. Indeed, in the first generation technology, most communication occurs in this way. A process of the Agent class, which is a sub-class of the Process class, has a go method. Upon the agent's successful invocation of go on itself, the following instruction is executed at a different location, possibly on a different computer.

Not all processes move around. In particular, Places are stationary processes that provide the destinations (i.e., server objects) for agents to visit. When an agent occupies a place, it is given a reference to that place so that it can avail itself of the service interfaces (i.e., APIs) offered there. By means of agents, places, and go, Telescript enables more than just remote programming, it provides a networking platform capable of hosting "intelligent" distributed applications.

2.0 Security Issues

Obviously, security is a major concern in such an environment. In an age of colorful, network-based threats such as viruses, worms, freeloaders, and zombies [4], there is amply justified reluctance to blindly execute unknown code. In order for Telescript to be viable, then, it must incorporate sufficient security and safety features to allay such concerns.

To be more precise, the security concerns revolve around:

- Authentication of the user. Who is responsible for this script? Who is accountable for any charges it runs up?
- Authorization of the user. Can this user execute this script? Which operations on which objects is this user allowed to invoke? What resources can this script consume?
- Virus propagation. Can agents corrupt the underlying system, or processes that they come in contact with?
- Trojan horses. Since Telescript does inter-process communication by passing objects around, and the only way to "use" an object is to invoke an operation on it, how does one know whether or not an arbitrary class contains malicious code?

The Trojan horse threat is perhaps the most challenging to deal with. For example, malicious code might try to:

- communicate private information (e.g., a credit card number, password, encryption key, or e-cash token).

- modify data (e.g., change an encryption key to a known value).
- impersonate another user (e.g., take advantage of running under its identity to defeat access controls).
- mount denial of service attacks (e.g., bring down the system, or consume excessive resources).

These concerns are amplified when one of Telescript's primary intended application areas, electronic commerce, is taken into account. It is important for the network and its users to understand and manage the trust each is placing in the other.

3.0 Safety and Security

There has been relatively little precedent for solving security problems such as these. In general, the approach has been to simply not admit unknown code. However, there has been some relevant work, mostly based on encapsulating untrusted code inside of "safe," restricted environments.

One example is the use of the chroot operation in variants of the UNIX™ operating system. Chroot changes the process's view of the operating system root directory. This is often used with services such as anonymous ftp to confine the directories to which an unauthenticated user has access. Even so, it is considered unwise to allow anonymous users to upload and run arbitrary executables.

Another may be found in the Safe-Tcl scripting language [5]. Tcl is a powerful, general purpose, interpreted scripting language that is often used to facilitate rapid development of portable, graphical user interface driven applications. Tcl applications have access to all system resources that the account they are running under does, just as any other application. However, the Safe-Tcl variant uses a modified interpreter from which "dangerous" operations have either been removed, or replaced with harmless ones. Since Safe-Tcl scripts do not make calls directly to the underlying system but, instead, access everything through the interpreter, the interpreter has the opportunity to act like a reference monitor and enforce access policies.

An analogous mechanism is used in the Java™ interpreter [6]. In common with Telescript, Java is object-oriented and supports multiple threads of control using built-in thread objects. HotJava™ [7], a Java-based browser for the World-Wide Web, supports downloading and executing Java applets on client computers. Namespace partitioning is used to isolate locally installed (and hence implicitly trusted) classes from downloaded (and hence, suspect) ones. Namespaces are controlled using class loaders, objects that both maintain the classes they install and resolve run-time class references for them. For example, the network class loader can enforce a policy of not mixing class definitions from different network hosts. A class loader also has an associated security policy manager object used to decide access to system resources for classes that it has loaded.

4.0 Telescript Security Model Overview

Telescript's approach to security is a bit more complex than the above, and includes a number of different mechanisms that can be applied at various degrees of granularity. The Telescript security model can be considered at four levels:

- *Object Runtime Safety*, which supplies the basic underpinnings upon which the other mechanisms rely.
- *Process Safety and Security*, which allows relatively safe interaction among agents and between agents and places.
- *System Safety*, which protects access to resources in the underlying operating system.
- *Network Security*, which provides authentication, communications privacy, and system level authorization.

Most of Telescript's security features are discretionary. That is, scripts need not use them if they don't need them. However, it may be risky not to do so.

5.0 Object Runtime Safety

Telescript is intrinsically "safe" from a programming perspective. Scripts either do what they are supposed to do or fail gracefully. Telescript does not have pointers or pointer arithmetic. Instead, objects are accessed only via their interfaces and only using valid references. Telescript engines provide run time type checking, automatic memory management with garbage collection, and exception processing. In addition, Telescript provides a number of built in classes for safe and efficient data structuring (e.g., List), with exceptions raised in the case of inappropriate indexing arguments.

This does not mean that Telescript programmers can be careless and are automatically freed from worrying about programming hazards. For example, Telescript provides synchronization and exception handling primitives which must be properly used.

From a security perspective, the most important thing is that the interpreter functions as though a reference monitor to mediate access to objects. The interpreter strictly enforces object encapsulation with public and private access modifiers for features (operations and attributes). Private features are only available internally within the class and its sub-classes.

With respect to viruses in Telescript itself, classes can only be installed once in the engine, and are immutable thereafter. An agent cannot patch or re-install a class that is already in the engine, or install a class of the same name in an attempt to subvert the class binding mechanism.

6.0 Process Safety and Security

Telescript builds upon object encapsulation to provide inter-process protection as well. In addition, Telescript provides a number of process security mechanisms, including:

- authenticated, unforgeable identities for each process.
- protected (read only) object references.
- quotas and privilege restrictions using permits.
- engine-mediated protocols (in the form of specific operations) for process rendezvous (agent meeting and place entry).
- security "mix-in" classes which permit objects to inherit specific security functionality, such as access controls.

6.1 Authenticated Authorities

An authority is a unique identifier that denotes a particular individual or organization. Every process has an associated authority. Telescript accommodates different kinds of authorities, including some that are registered with a naming authority.

The authorities used with portable, hand-held Telescript devices are generated cryptographically. During device personalization, users generate a randomly seeded authentication RSA key pair. The authority is derived from a digitally signed buffer using a cryptographic one-way hash function. Thus, devices can register their authority with a Telescript service and subsequently authenticate by proving they know the private key used to generate the authority.

Every Telescript engine runs under its region authority, a privileged authority analogous to the root account in UNIX. Within a region there can be places with different user authorities, occupied by agents with different user authorities as well. Each authority has an associated authentication level, corresponding to the degree of trust the region places in it. The authentication level may be taken into account when making identity-based authorization decisions.

6.2 Protected References

All objects must have an owner process to avoid being garbage collected. In the case of an agent, ownership conveys the right to take the object instance along should the agent move. Processes may obtain references to unowned objects, but such references could go void as a result of an agent doing a `go`.

Generally, a reference to an object enables invoking public features of that object. However, a protected reference to an object cannot be used to modify that object. A protected reference can be created from another reference using the protect operation. `Protect` is a public feature of the class Object and is inherited by all object instances.

6.3 Permits

Permits provide a mechanism for limiting resource consumption and restricting the capabilities of executing code. A permit is an object that can specify a maximum age, size, or process priority, or that indicates whether new processes can be created or whether an agent can go. Telescript engines enforce four kinds of permits:

- *native*, assigned by a process' creator.
- *regional*, assigned by an engine when a process enters or is created.
- *local*, assigned by a place when an agent enters that place or a process is created in that place.
- *temporary*, which can be imposed over arbitrary blocks of Telescript code.

The effective permit in force in any situation is computed by the engine by logically intersecting all of the permits that apply. If a process or fragment of code violates its permit, the engine throws an exception which usually (but not always) results in the process terminating.

6.4 Entering and Meeting

In processing an agent's `go`, Telescript allows the destination place to decide whether or not it will let the agent in. The engine calls the place's `entering` operation and supplies the agent's authority, class and permit. The place can decide to admit the agent, perhaps imposing a local permit, or it can deny entry by raising an `OccupancyDenied` or `DestinationUnknown` exception. If the place admits the agent, the engine supplies it an unprotected reference to the agent, and the agent resumes execution in the new place. The agent can also obtain an unprotected reference to the place, permitting either to initiate interaction with the other by invoking the other's features. In a similar fashion, the engine mediates a meeting protocol between agents.

To some extent, places can be used to provide restricted environments for safely hosting unfamiliar agents. An example of a restricted place is "purgatory," a place found in production Telescript platforms and used as a default destination for agents that can't be delivered to their desired destination for some reason. An agent might return from a `go` in purgatory with a trip exception and a permit

giving it a very short time to live. The intention is to provide a harmless place for the agent to catch the exception and decide to go someplace else. Purgatory has no operations for the agent to call, nor does it care what interfaces the agent supports.

6.5 Security Mix-ins

A mix-in is a class that can be inherited by another class, but which can not have its own object instances. Mix-ins provide a limited form of multiple inheritance. Telescript provides security mix-in classes that result in special treatment by the engine.

- *Unmoved*. An exception is thrown if an agent attempts to move the object.
- *Uncopied*. An attempt to copy the object returns a reference to the original object.
- *Copyrighted*. The engine applies a configurable authorization policy to control instantiation of objects.
- *Protected*. The object cannot be modified once created, with the possible exception of ownership transfer.

In addition, application frameworks, such as the Telescript Application Library, can provide mix-in classes for including identity-based access control list functionality in class definitions. These classes are not, however, treated in any special way by the engine.

7.0 System Safety

Telescript provides visibility to underlying system resources, such as files or network access, through system objects built into the engine. To access a file, for example, a script would instantiate a External File object, supplying the file name and related information.

System objects of this type are also Copyrighted, Unmoved, and Uncopied classes. The access policy is embodied in a "copyright enforcer" object maintained by the region authority. In first generation Telescript engines, the following system classes are protected in this way:

- External File (used to access files).
- Control Manager (used to perform privileged management functions, like causing the engine to exit).
- External Handle (used to access network ports).

Under normal circumstances, only system management functions access the local file system, usually for purposes like configuration, auditing and logging. Agents deprived of file system assess do not have an opportunity to install a virus there.

Telescript does not provide a way for scripts to execute arbitrary commands on the underlying system.

8.0 Network Security

A Telescript network is a set of interconnected engines involving possibly many different regions. Each region can set its own security policy. Furthermore, agents travel between regions over secure channels.

8.1 Region Policies

The outermost place in any engine is its engine place. An engine place is a privileged process running under the region authority. The Engine Place class has operations for deciding whether to admit an agent to the region at all, and for imposing a regional permit on that agent. It can also control what places the agent can visit.

Particular Telescript engines will customize their region policies according to their own security requirements.

8.2 Secure Channels

A secure channel is an authenticated, private network connection between regions. In first generation Telescript platforms, agents are not individually authenticated. Instead, authentication is region to region. When an agent arrives, the incoming engine place is given the agent's authority, the remote region's authority, and the security regime that was used for the connection. Regions need to decide whether to believe a remote region's identification for transferred agents. For example, since single-user devices are considered a "region," a typical policy when communicating with such a device is to only accept agents whose authority was the same as the remote (authenticated) region.

A security regime is a pre-determined set of security services that must be negotiated together when setting up a secure channel. Currently, Telescript supports the following security regimes:

- *identification*, where only authority information is exchanged, with no encryption performed. This is usually used for public access, for example, to obtain a sign-up form.
- *registration* uses RSA public key authentication, Diffie-Hellman key exchange, and RC4 encryption to create a secure channel. A device uses the registration regime the first time it contacts a Telescript service. When this regime is used, the region policy only permits certain classes of registration agents to enter, and limits the places that these agents can visit. After a successful registration, the engine retains authentication state information about that remote device.
- *fast authentication* uses the DES encryption algorithm to both authenticate the remote region and to securely transfer shared secrets from which an RC4 traffic key is

computed. This is considerably faster than doing public key operations each time a device contacts the service.

- *re-key* is essentially the same as registration, involving public key authentication, but only used when a DES key expires or under circumstances where the device or service lose the shared key.
- *authentication* is similar to fast authentication but uses the Diffie-Hellman algorithm to negotiate a shared RC4 traffic key. This provides perfect forward secrecy [8].

For the registration, re-key, and authentication regimes, an asymmetric authentication exchange is used with devices authenticating to Telescript services. As previously mentioned, devices authenticate by proving ownership of their authorities. The Telescript service authenticates to the device using a public key certificate.

8.3 Export Restrictions

Telescript is intended to be used world-wide. As a consequence, only "export strength" encryption was used in first generation Telescript products. In other words, the 40-bit RC4 keys needed to qualify for relatively unrestricted US government export approval have subsequently been demonstrated to be breakable. In addition, relatively short 512-bit key lengths are used for RSA and Diffie-Hellman operations.

9.0 Limitations of the Current Technology

Although Telescript has a large number of security mechanisms, the current design has some limitations. Chief among these are:

- safety is achieved by careful defensive coding practices, not "by default."
- safe object sharing can only be done using known, trusted operations.
- access control policy is only partly engine-enforced.
- export crypto is too weak.

The first two of the above are perhaps the most serious from a language design viewpoint. The others can and will be addressed in future product releases using more or less available technology, such as Internet encryption protocols as they become standardized.

Telescript provides a large number of security mechanisms that must be used defensively. Deciding which to use and in which situations can be daunting to the novice programmer. In addition, there are important scenarios that the mechanisms do not completely cover. In these cases, the only recourse is to only execute known, trusted operations.

For example, a place can safely admit agents of previously unseen classes by imposing a suitably restricted permit at entering, performing access control checks on all public features, and only exchanging objects of known classes (not sub-classed with specialized operations that might contain Trojan horses). It should also avoid calling operations on the agent directly unless it imposes appropriate temporary permits. Even so restricted, the agent might use the opportunity to masquerade as the place. If other agents occupying the place come in contact with that agent, they must observe similar precautions.

One common practice is to use sealed features or entire interfaces. The concept of sealed is similar to "final" in other object oriented languages. The implementation of a sealed operation cannot be overridden in a sub-class, hence the behavior of that operation is known apriori.

While the current mechanisms are sufficient for deploying first generation Telescript networks, relying on "paranoia programming" is contrary to the goal of future, open Telescript networks.

10.0 Directions for Future Work

The next generation Telescript networks can be expected to have additional security features. At this writing, the feature set has not been finalized. However, the following mechanisms are representative of those under active consideration:

- *certified classes* that have associated with them one or more digital signatures. For example, a class can be signed by its author or by some certifying agency.
- *access controls* that are easier for programmers to specify and use and that are enforced by the engine.
- *permit extensions* for specifying additional policy dimensions and restrictions.

These mechanisms are inter-agent and agent-to-place. In addition, it would be nice to reduce the amount of trust agents need to have in the places and regions that they visit.

11.0 Acknowledgments

Telescript is the result of the design contributions of many people, but is primarily due to the inspiration of Jim White, Rich Miller, Doug Steedman, and Chris Helgeson.

12.0 References

[1] White, J.E., Telescript Technology: An Introduction to the Language, General Magic, Incorporated, Sunnyvale, CA., October, 1995.

[2] White, J.E., Mobile Agents, in Software Agents, Bradshaw, J. (ed.), AAAI Press and MIT Press, Menlo Park, CA., January, 1996.

[3] Chess, D., Grosof, B., Harrison, C., Levine, D., Parris, C., and Tsudik, G., Itinerant Agents for Mobile Computing, IEEE Personal Communications, October, 1995, pp. 34-49.

[4] IBM Corporation, Things that go bump in the Net, http://www.research.ibm.com/xw-d953-bump.

[5] Osterhout, J.K., Scripts and Agents, the New Software High Ground, Invited Talk, Winter, 1995 USENIX Conference, New Orleans, LA.

[6] Gosling, J., The Java Language Specification, Sun Microsystems, Mountain View, CA., May, 1995.

[7] Sun Micrososystems, HotJava, the Security Story, http://www.javasoft.com/1.0alpha3/security/security.html.

[8] Diffie, W., Van Oorschot, P., and Wiener, M., Authentication and Authenticated Key Exchanges, in Designs, Codes and Cryptography, volume 2, Kluwer Academic Publishers, 1992, pp. 107-125.