Trusted Proxy

(a.k.a. Rights Amplifier, Limited View, Restricted Channel,
Integrity Preserving Function, Safe Protocol)

Abstract

A trusted proxy acts on behalf of the user to perform specific actions requiring more privileges than the user possesses. It provides a safe interface by constraining access to the protected resources, limiting the operations that can be performed, or limiting the user's view to a subset of the data.

Problem

It is often necessary to expose components that are not adequately protected to an audience of untrusted users. These components are identifiable by one or more of the following characteristics:

- The component offers protection mechanisms at too great a level of granularity. In order to accomplish necessary tasks, the user must be granted privileges that could be misused. For example, in UNIX, the right to append data to a file brings with it the right to overwrite that file.

- The component is designed for a "safe" environment, and does not offer protection mechanisms. Many products are designed for a LAN environment and cannot be safely exposed to the Internet.

- The component is very complex, and contains many features that could be misused, or many bugs that could be exploited. Many commercial general-purpose operating systems, utilities, and applications fall into this category.

- The component is subject to frequent change, and it is not possible to adequately assess the security of every change. Many major Web sites are continually revised. There simply isn't time to perform a full security assessment of every change.

Exposing these components to anonymous access in an untrusted environment can be dangerous. Many components can be misused to negatively effect accountability, availability, integrity and confidentiality. Worse, some vulnerabilities can result in total system compromise.

Solution

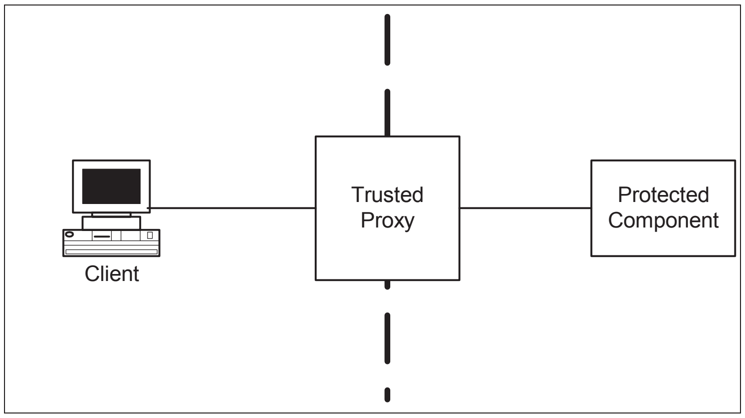
A trusted proxy acts as a buffer between inadequately protected components and an audience of untrusted users. It intercepts and filters all communication between the users and the component(s) in question. By preventing direct access, it can compensate for weaknesses in the protected component(s) and ensure that the appropriate policy is consistently enforced.

Trusted proxies allow custom, finely tuned security policies to be developed and enforced. The *Trusted Proxy* pattern can be used to add protection to components that do not offer any protection mechanisms. Or it can augment the protection mechanisms of existing components to enforce more restrictive policies.

Trusted proxies also allow integrity constraints to be enforced. For example, many systems require transactional models where multiple data files must be kept synchronized. For example, accounting systems require integrity constraints on the contents of multiple files. Inventory systems allow changes to the inventory, but record the identity of the individual making the change. And database systems require that any change to the database obey the integrity rules defined for that database.

When building a trusted proxy to protect an existing component, there are two basic approaches to filtering the data. You can either search the data for known bad characteristics, or you can create the new request to be delivered to the target from scratch and only copy certain parts from the original client-provided request. Scanning the original data is generally faster and allows new features to be added without breaking the proxy. But rebuilding the data is generally more likely to defuse new attacks. For example, when new vulnerabilities in the TCP/IP stack are found, proxies that simply send the packets provide little defense. Proxies that rebuild every packet usually prevent the malformed packets from being delivered to the target.

Proxies generally do not keep up with the latest and greatest features in the components that they are protecting. Whether the proxies are developed in house or commercially available, the developers of the proxy need additional time to update and perform quality assurance on the proxy. As a result, the proxy can be seen as an impediment to achieving other non-security goals. In an environment where such changes are likely to be required, a scanning approach to filtering is generally preferred.



Issues

Types of Proxies

Trusted proxies can be either transparent or opaque. A transparent proxy is one that requires no modification to the original system. It can be inserted between existing components without breaking compatibility. Often, the components in question will not even be aware of the existence of the proxy. Transparent proxies are useful for filtering out known attacks, or providing a limiting view of the protected resources. Many network-level filters act as transparent proxies, allowing benign traffic to flow through undisturbed, but intercepting traffic that is known to be dangerous.

An opaque proxy is one that offers a different interface from that offered by the protected resources. An opaque proxy requires that clients program directly to the proxy – if inserted between existing components, it will generally require modification to the client. Opaque proxies sometimes augment the existing interfaces with additional security features, such as authentication. Opaque proxies often define higher-level functions and then map these to the low-level functions offered by the protected resources.

For the sake of efficiency, it may be possible to have the target respond directly to the client instead of proxying the response as well as the request. When the proxy is used to enforce a conventional access control model, the security generally is applied only to the request and it is safe to have the response be unproxied.

If possible, proxies should be designed to be layered. A composite solution of simple, layered proxies is generally easier to maintain and secure than a single blob that enforces multiple policies and goals. Many proxies are developed for non-security purposes. In particular, load-balancing and fail-over mechanisms often require the ability to redirect requests from the original target to a stand-in. Make sure to test these facilities to understand how they interact with the proposed security proxies.

Defending the Proxy

Realize that the proxy itself will be the target of any attacks launched at the original. If the proxy is more vulnerable than the original, the result is a net minus. In particular, never use variable-sized buffers of unbounded string manipulation functions, as these are often compromised using buffer-overflow attacks. If possible, use an existing product or technology instead of developing one from scratch. Of course, any mechanism that defends an unprotected component is by definition stronger than the original.

Proxies should be developed with Implementation Hiding in mind. For example, when malformed or invalid data is encountered, the proxy should not provide a possible attacker with specific information that could aid in identifying or circumventing the proxy. The error should be logged, so that administrators can debug the problem, but the client should only be given a generic failure response.

A trusted proxy should filter all input from untrusted clients, as the components being protected may not perform such filtering. See the *Client Input Filters* pattern for specific strategies.

Designing Around the Proxy

The trusted proxy must be non-circumventable. Attackers generally prefer to attack the weakest part of a system. Often, that means looking for a way to circumvent the proxy rather than attacking the proxy itself. Don't fall into the trap of imagining that the attacker will focus all his/her efforts against the proxy. (See the *Red Team the Design* pattern.)

Do not give the proxy any privileges that it does not absolutely require. A vulnerable proxy running with administrative privileges may be a bigger security flaw than the components it was intended to defend. If at all possible, use the operating system or language access control mechanisms to restrict the privileges of the proxy to the minimum possible.

A proxy cannot fully eliminate the security implications on the systems that it defends. If an attacker figures out a way to misuse legitimate services, the proxy may simply forward the seemingly legitimate request through to the target. (See the *Share Responsibility for Security* pattern.)

Provide logging interfaces that allow invalid attempts to be monitored and stored. Many attackers will systematically probe the components in a system. A system administrator who is alerted to attacks on the proxy may be better able to distinguish attacks on other components. (See the *Log for Audit* and *Network Address Blacklist* patterns.)

Examples

At the code level, any object that contains private data is a form of trusted proxy. It ensures that the integrity constraints within the object are enforced by disallowing direct access to that data. It exports integrity preserving functions so that users who are not trusted to directly manipulate the data can be allowed to access the data via certain controlled interfaces. However, most programming languages (including C++), cannot guarantee that other components will not bypass these protections. Java is a notable exception: in most cases, objects will not be able to circumvent the language's protection mechanisms.

J2EE Enterprise JavaBeans (EJBs) can provide access to a database with only the specific limited functionality required by the application.

At the system level, the UNIX Mail delivery program has the privilege to write into any user's files. The mail delivery program is trusted to only append incoming mail to the appropriate mailbox file, and not touch any other files.

All database programs allow users to modify files in a constrained manner but generally disallow direct access to those files. Oracle, for example, does not give users permissions to open database files directly. They can invoke Oracle to open the file on their behalf, subject to Oracle's access control rules. Many databases also allow restricted views of tables to be established. A restricted view permits each user to see only that subset of a table that he or she has privilege to.

At the network level, a firewall is a classic trusted proxy. It provides a constrained interface to systems that are not trusted to be directly accessible from the Internet. Many firewalls also proxy outgoing connections, in order to prevent untrusted users from bringing dangerous materials into the protected network.

A Web/ftp server that accesses local files and serves them to remote users who have no privileges on the local system(s) in another example of a trusted proxy.

Many Web applications provide users with restricted access to a private database. Users are only allowed to see their own account data (and often only a subset of that). For example, Amazon.com will let you view your order history, but it will not show you the profile data that it has compiled about your activities at that site.

Trade-Offs

Accountability	Trusted proxies are often used to enhance accountability by enforcing authenticated access to protected resources and adding logging rules to those resources.
Availability	A trusted proxy adds another possible point of failure. If either the proxy or the proxy's target fail, the system will be unavailable. A proxy that is significantly less reliable than the protected resource, or vulnerable to overloading attacks, will drastically reduce the overall availability.
Confidentiality	This pattern can be used to protect important data or subsystems.
Integrity	Trusted proxies have a huge impact on integrity: they prevent misuse of important subsystems.
Manageability	Configuration of the trusted proxy can be extremely difficult, especially if the individual configuring the trusted proxy does not fully understand the types of data and the consequences of data flowing through the trusted proxy.
Usability	Trusted proxies generally do not affect usability, particularly if they are transparent to the user. However, firewalls are an example where a trusted proxy has been observed to reduce usability dramatically. Many users find that their on-line activities are thwarted by an overly restrictive firewall policy. One unfortunate result is that end users often attempt to circumvent firewalls when the firewall prevents them from conducting their desired activities.
Performance	A trusted proxy represents an additional level of indirection between the user and the protected resource and as such has a negative impact on performance. This impact can be quite severe and should be investigated using prototypes early in the development cycle.
Cost	A trusted proxy is a textbook example of information hiding, in the classic software engineering sense. A well-designed trusted proxy can greatly reduce inter-object coupling. However, if the object interfaces are not chosen well, the result can be a maintenance headache. The traditional object-oriented approach to maintenance (re-factoring of

	the object design) can run into problems if organizational security policies are defined in terms of those object interfaces.

Related Patterns

- *Red Team the Design* – a related pattern.
- *Share Responsibility for Security* – a related pattern.
- *Log for Audit* – a related pattern.
- *Network Address Blacklist* – a related pattern.

References

[1] Balestracci, S. "PC Week Hack of 1999". http://www.sans.org/infosecFAQ/threats/PC_week.htm, February 2001.

[2] Gamma, E., R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[3] Zwicky, E., S. Cooper, and D. Chapman. *Building Internet Firewalls 2nd Edition*. O'Reilly & Associates, 2000.