

BMC Software Inc.

Technical Disclosure Publication Document

Amol Vaikar, Jaswinder Bhamra, Neeran Karnik

X11 Tunnel Security using Process Relationships

Posted: February, 2012

Overview

This paper describes a novel technique for authorizing tunneled X11 connections without requiring credentials-based user authentication in the situation of a common datacenter in which a user of a management application such as BMC Bladelogic Server Automation (BSA) is *mapped* to a local operating system (OS) user on a server and needs to use tunneled X11 access to that server. It is necessary to ensure that only such user's X11 clients are allowed to use the X11 tunnel started on its behalf. This is achieved without requiring a user login, by validating that the connection requester is a *descendant* of the process that created the tunnel using the parent relationships of the process as well as the process group information, as described below.

Background

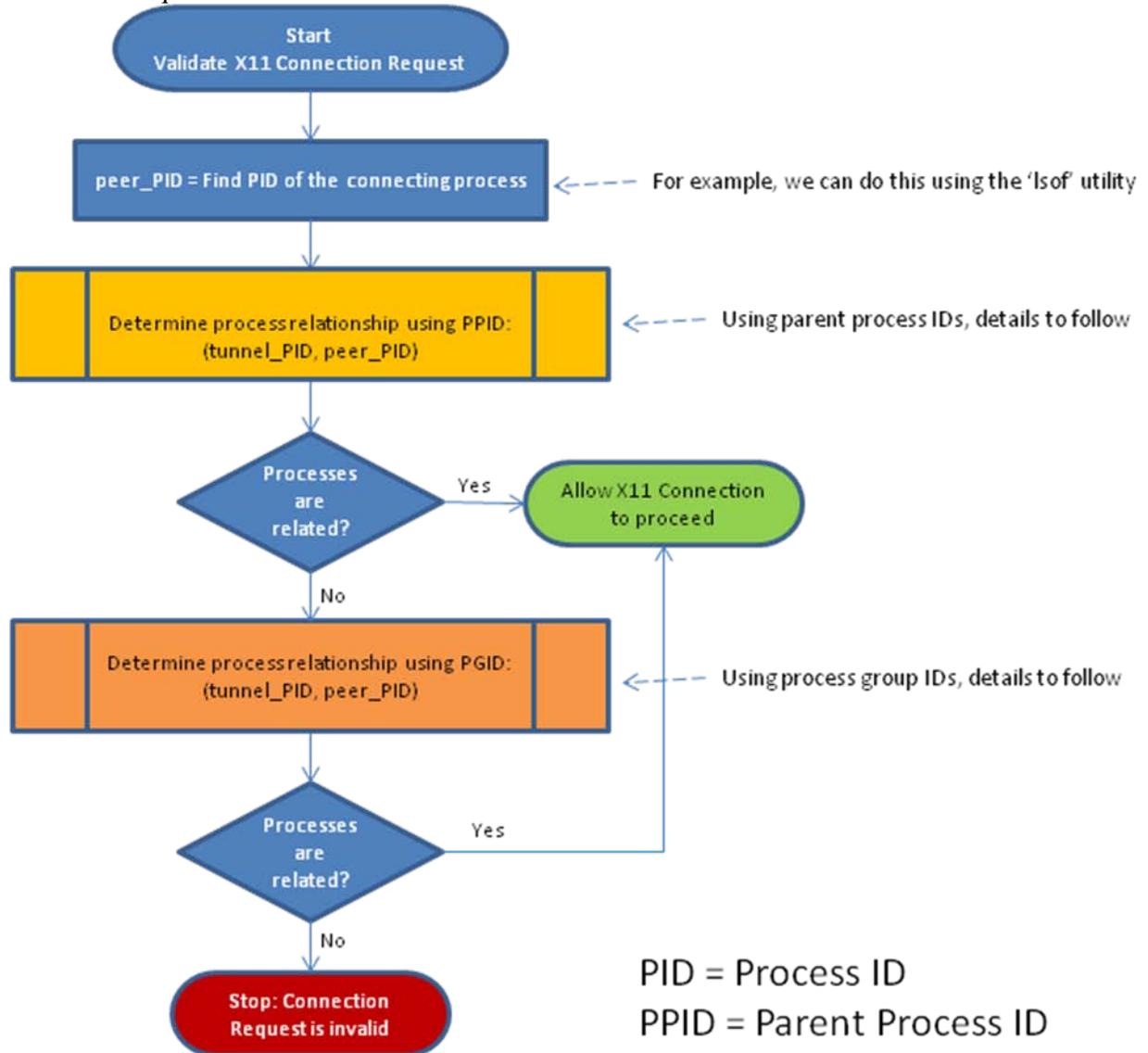
Most datacenter management applications install an agent on each server managed by such applications. A user remotely accesses the agent in the context of a local OS user identity on that server. This mapping between the application user and a local OS user is configured by a security administrator. Multiple application users often map to the same local user (e.g. 'guest', or even 'Administrator').

A feature of such an agent is the ability to set up an X11 tunnel to a client machine. This allows a remote user to run X Windows programs on the managed server, and view the X display on his or her client machine. The agent starts a tunnel process on the managed server, which relays X11 traffic to the X-server on the client machine. This process listens on a well known (or predictable) port number. If an attacker connects to an existing X11 tunnel process, the attacker might receive access to the original user's X Windows desktop. One way to prevent this would be to authenticate the user each time an X server connection is initiated, however, in this scenario with automated user mapping (without interactive authentication using passwords, for example), this is not feasible. In any case, the attacker could be legitimately mapped to the same local user, so the local user identifier (ID) is insufficient to protect the tunnel. What is needed is a mechanism that disallows such attacker's processes from piggybacking onto existing tunnels, while still allowing all legitimate processes (started by the tunnel initiator) to use the tunnel.

©Copyright 2012 BMC Software, Inc.

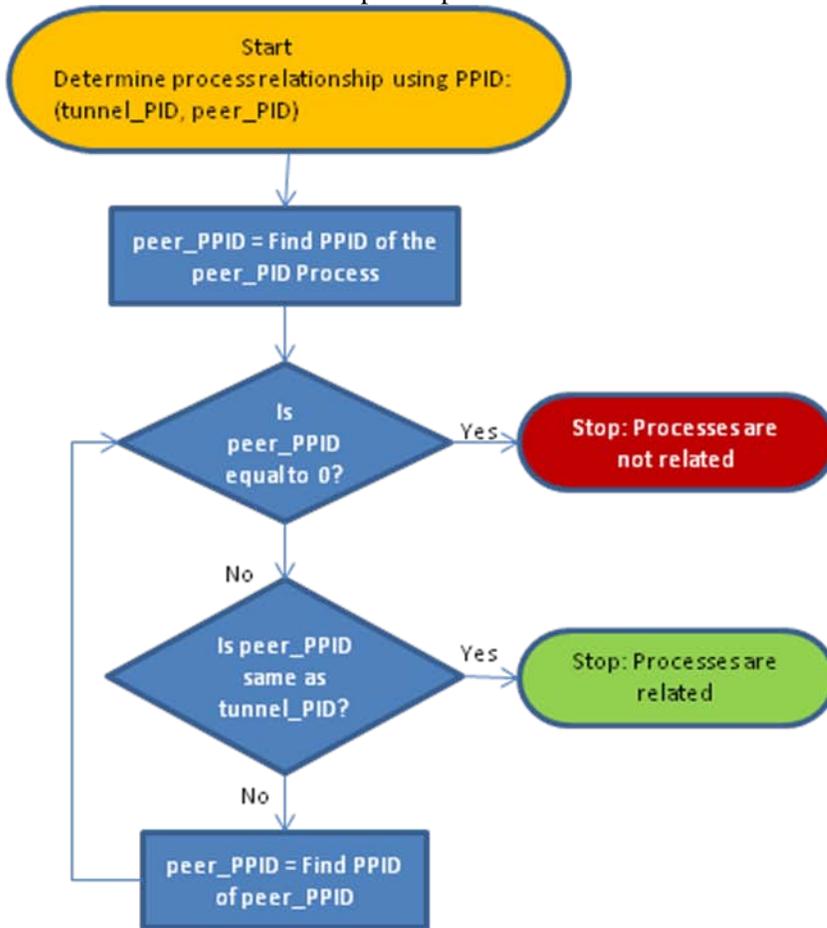
Solution

To prevent this attack, we do the following in the tunnel process that handles the incoming X11 connection request:



When a connection request is made, what must first be determined is the process ID of the peer process. This can be done, for example, using the 'lsdf' utility. The next task is to find whether the peer process is a descendant of the tunnel process, using the *parent process ID* mechanism. This is detailed below in a separate flowchart. If the processes are related, the X11 connection is allowed to proceed. If they are not related, a further test is done using the *process group ID* mechanism, also detailed separately below. Again, the connection is allowed only if the processes are related.

Below is the flowchart for the parent process ID mechanism:



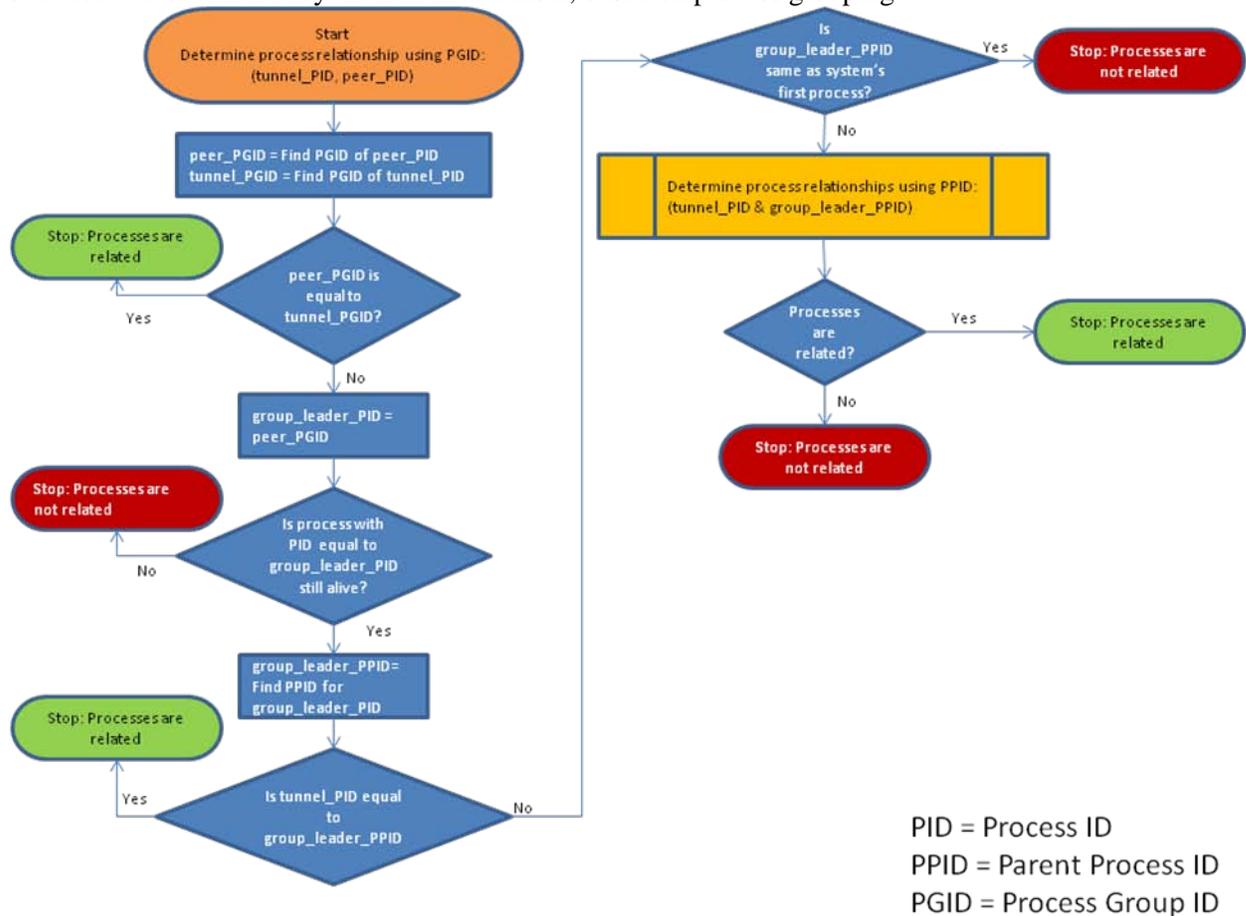
PID = Process ID

PPID = Parent Process ID

Each ancestor pid of the peer process is iteratively examined to check whether it matches that of the tunnel. If it matches, it can be concluded that the processes are related. If not, the process is repeated with the next-level ancestor, using the parent IDs iteratively. At some stage, if the parent ID is zero (the pid of the 'init' process in Unix flavors), this indicates that the system has traversed up the process hierarchy without encountering the tunnel's process ID. Thus, this routine can conclude that the processes are not related.

©Copyright 2012 BMC Software, Inc.

Of course this can also happen if an intermediate ancestor process has terminated. In such case, that process' children will have their ppid set to zero, thus breaking the chain. However, this situation can be handled by the next mechanism, based on process grouping.



In this routine, it is first checked whether the tunnel and the connecting peer both belong to the same process group. Because a process group ID is the same as the pid of the group leader (creator), this would imply that they have a common ancestor, and thus belong to the same OS user. The processes can thus be declared as 'related'.

If the peer process' group leader is alive, the search continues from its parent upwards. If this parent happens to be the tunnel process, it can be concluded that they are related. If it is the 'init' process, then they are not related. Otherwise, whether the tunnel process is an ancestor of the peer's group leader's parent is checked. This is done iteratively, reusing the earlier parent process ID mechanism.

In case the peer's process group leader is no longer alive, the result is a corner case – the chain is broken irretrievably, and it cannot be decisively concluded that the processes are related. Although this is a rare occurrence, it is important to declare that they are unrelated, to avoid any false positives.