The OpenSSH <=6.8 X11 SECURITY bug

There was a security problem (CVE-2015-5352) in OpenSSH <=6.8 that allowed malicious servers, if a client connected to them using `ssh -X`, to connect to the SSH client's X server without being subject to X11 SECURITY restrictions.

X authentication

After a client has connected to an X server, it needs to authenticate. This can be done by specifying explicit authentication information (in practice, that usually means using `MIT-MAGIC-COOKIE-1`, which requires the user to send an authentication "cookie" to the server), but can also be done implicitly - for example, for a local connection, the server might let the client in based on its UID.
Interestingly, the X server falls back to implicit authentication even if the client has explicitly specified invalid authentication data.

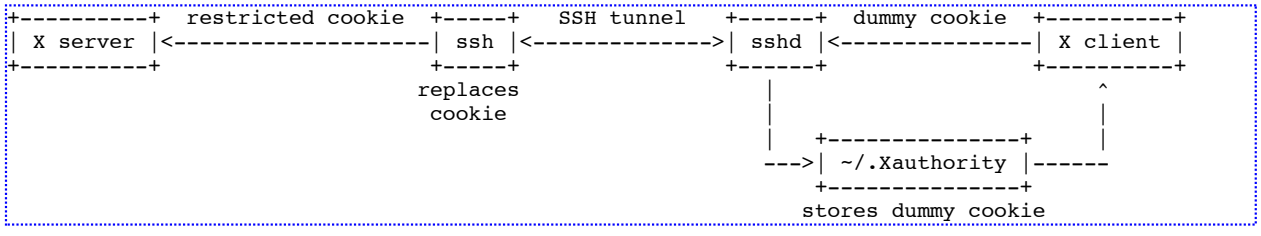
X11 SECURITY

The X11 SECURITY mechanism lets the user create magic cookies that, when used to authenticate to the X server, restrict what the client can do. (They prevent the client from using unsafe X extensions and prevent access to windows not subject to X11 SECURITY restrictions, but don't prevent access to the windows of another client that is subject to X11 SECURITY restrictions.)
As all magic cookies, magic cookies with X11 SECURITY restrictions can have a timeout associated with them, and after the cookie has been unused for the duration specified by the timeout, the cookie is deleted. **If a client that could successfully authenticate implicitly attempts to explicitly authenticate using an expired cookie with X11 SECURITY restrictions, the explicit authentication fails and the X server falls back to implicit authentication without X11 SECURITY!**

(non-trusted) X forwarding

When an SSH client connects to an SSH server with `ssh -X`, the SSH server can create channels through the existing SSH tunnel that the client forwards to the local X server. X authentication is handled as follows:

- When connecting to the SSH server, the SSH client registers a new MIT magic cookie with lifetime `ForwardX11Timeout` (default: 20 minutes) with the X server. This cookie is subject to X11 SECURITY restrictions. I'm going to call this the *restricted cookie* from now on.
- When connecting to the SSH server, the SSH client creates a *dummy cookie* that looks like an MIT magic cookie. It sends this string to the SSH server, and X clients on the SSH server have to send the dummy cookie when authenticating to the X server through SSH. The SSH client verifies the correctness of the dummy cookie, then replaces it with the restricted cookie before forwarding the authentication request to the X server. Here is some crappy ASCII art of the information flow:



The obvious problem with this approach is that, if no X clients connect to the X server through the SSH tunnel for the period specified by `ForwardX11Timeout`, the server will forget the cookie. If the SSH client allowed new connections to be created afterwards, the X server would not recognize the magic cookie and would fall back to implicit authentication using the UID that connected through the unix domain socket, giving the X client access without X SECURITY restrictions. Because of that, ssh rejects new X11 channel requests after the timeout has expired.

The problem

The problem is that, to be precise, ssh doesn't have to block new connections to the X server after the timeout has expired - instead, it has to block X11 authentication attempts. The cookie can still expire after a connection to the X server has been created, but before the X client has sent its authentication request. While connection creation is usually followed directly by authentication, a malicious attacker can delay it arbitrarily. This is the attack:

1. victim (SSH client) connects to attacker (SSH server) with `ssh -X`
2. attacker waits 19.5 minutes
3. attacker opens an X11 connection to the SSH server, the SSH server requests creation of a new X11 channel over the SSH connection, the SSH client connects to the X server
4. attacker waits 1 minute, timeout expires, X server forgets about the restricted cookie. SSH client allows no new X11 channels anymore
5. attacker sends authentication request with dummy cookie, SSH client sends authentication request with restricted cookie, X server doesn't recognize cookie and allows connection through implicit authentication
6. attacker interacts with X server without being subject to X SECURITY restrictions

In practice, you can create the one-minute delay by launching some X client under a debugger and breaking on `_xcb_get_auth_info`. After one minute has expired, let the program continue to run.

Impact

A program that is not subject to X11 restrictions can interact with all programs you have open, basically as if it was you. It can, for example, access all your open terminal windows and type arbitrary commands into them using the XTEST extension, which would probably allow the SSH server to completely compromise any client that connects with `ssh -X`. For example, an attacker can send arbitrary keystrokes to the active window like this:

```
$ gdb xdotool
[...]
(gdb) break _xcb_get_auth_info
Function "_xcb_get_auth_info" not defined.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 1 (_xcb_get_auth_info) pending.
[ wait here until the 19.5 minutes are over ]
(gdb) run type --delay 1000 ABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABC
[...]
Breakpoint 1, _xcb_get_auth_info (fd=3, info=0x7fffffffe380, display=10) at ../../src/xcb_auth.c:313
313     ../../src/xcb_auth.c: Datei oder Verzeichnis nicht gefunden.
[ wait here until the 20 minutes are over ]
(gdb) cont
Continuing.
```

The fix

OpenSSH 6.9 fixes this issue by checking for timeout expiry both when the SSH server requests a new X11 channel and when the SSH server sends X11 authentication data. Additionally, because something like an off-by-one or a timing skew would be critical here, the timeout for the MIT cookie is increased by one minute while the timeout after which ssh refuses X11 connections / authentication attempts stays the same .