

Environment Poisoning and login –p

Source: <http://www.derkeiler.com/Mailing-Lists/FreeBSD-Security/2004-02/0135.html>

From: Tim Kientzle (*tim_at_kientzle.com*)

Date: 02/25/04

Date: Wed, 25 Feb 2004 10:54:31 -0800

To: freebsd-security@freebsd.org, das@freebsd.org

There's been an ongoing discussion (started by Colin Percival's recent work on nologin) about environment–poisoning attacks via "login –p".

I thought I saw a way to address this, but the more I learn, the uglier this looks. Maybe some of the good folks who read freebsd–security can puzzle this one out:

Problem: login –p can be used to propagate environment flags in order to trojan the user shell of the target account. There are several significant cases:

- * Dynamically–linked target shell.
- * Target account shell is a shell script.

Both of these are quite common in practice. (However, I will note that in –CURRENT, both "nologin" and /bin/sh are statically–linked and thus currently immune to this.)

The particular thread that started me looking at this concerned "nologin" scripts or programs that attempt to block access to certain accounts. David Schultz has demonstrated that login –p can be used to circumvent a dynamically–linked nologin program, for example. If your friend's account hasn't been blocked, you may be able use that to circumvent any blocks placed on your own account.

Possible fix: Ignore "–p" flag if target shell is not in /etc/shells. In this scenario, a nologin program would not be listed in /etc/shells, and thus such attacks would be blocked.

Problem: One common use of nologin scripts is to create ftp–only accounts. However, ftpd(8) limits ftp access to accounts with "standard" shells. As a result, ftp–only accounts must have their shell listed in /etc/shells.

Possible fix: Have login unconditionally discard LD_LIBRARY_PATH

FreeBSD-Security: Environment Poisoning and login -p

and LD_PRELOAD from the environment, even if "-p" is specified.

I'm unsure that this is sufficient. I'm also unsure whether this blocks legitimate use of "login -p." I am generally distrustful of blacklist security approaches; I'd prefer a whitelist approach that only passed through selected environment variables.

Possible fix: Eliminate the "-p" option to login.

This would certainly close the hole, but could introduce complications elsewhere. (I've looked at telnetd, which uses this flag. However, it appears to use it only to propagate the TERM flag, which our login program does even without -p.)

Part of the issue here is an underlying disagreement about the meaning of "standard shell." Skimming the -CURRENT source tree reveals several different definitions of "standard shell":

- * A group of "equivalent" shells from which users can choose. (from chpass(1))
- * An indicator that the user is allowed ftp access (from ftpd(8))
- * An indicator that su is permitted
- * An indicator that mail include files should be honored (in sendmail; based on a very quick skimming of the source)

These are not entirely consistent interpretations, resulting in security problems with "ftp-only" access, for example.

I've attached a patch to "login" that implements the two "Possible Fixes" above. I'm not entirely happy with it, though, for the reasons I've indicated.

Suggestions?

Tim Kientzle

Index: login.c

```
=====
RCS file: /home/ncvs/src/usr.bin/login/login.c,v
retrieving revision 1.98
diff -r1.98 login.c
86a87
> static int chshell(const char *);
468c469,472
```

FreeBSD-Security: Environment Poisoning and login -p

< * preservation – but preserve TERM in all cases

```
---
>     * preservation or the user has a non-standard shell.  In
>     * particular, this "non-standard shell" check blocks certain
>     * environment-poisoning exploits against nologin scripts.
>     * Preserve TERM in all cases.
471c475
<     if (!pflag)
---
>     if (!pflag || !chshell(shell))
476a481,491
>     * The chshell() check above isn't sufficient, though.
>     * For example, consider a custom nologin script used
>     * to limit accounts to ftp-only access.  ftpd(8) requires
>     * the user shell to be in /etc/shells, thus crippling
>     * the above check.  The following provides some modest
>     * additional security limits in such cases.
>     */
>     unsetenv("LD_LIBRARY_PATH");
>     unsetenv("LD_PRELOAD");
>
>     /*
935a951,968
> }
>
> /*
> * Return TRUE if the shell is a "standard" shell.
> * (That is, one listed in /etc/shells.)
> */
> static int
> chshell(const char *sh)
> {
>     int r;
>     const char *cp;
>
>     r = 0;
>     setusershell();
>     while ((cp = getusershell()) != NULL && !r)
>         r = (strcmp(cp, sh) == 0);
>     endusershell();
>     return (r);

```

freebsd-security@freebsd.org mailing list

<http://lists.freebsd.org/mailman/listinfo/freebsd-security>

To unsubscribe, send any mail to "freebsd-security-unsubscribe@freebsd.org"