The Ultimate Reuse Nightmare: Honey, I got the wrong DLL.

ACM SSR '99 Panel Statement[*]

P. Devanbu
Dept. of Computer Science,
devanbu@cs.ucdavis.edu
http://seclab.cs.ucdavis.edu/~devanbu
University of California, Davis

November 10, 1999

Software reuse is now a reality. If you think I'm kidding, take a look at the installation of any major desktop software product, and ask yourself this question: Which files belong *only* to this application, and are *not* reused? OK, quick, can you tell? Is this really a word processing application, or is it a COM component that is also used by a spread sheet application and my presentation application? Does this Java class belong to the internet browser, or my accounting package? Who knows? More importantly, who cares?

But that's exactly the point. Reuse should mean never having to say you care! Component builder Ahmed should be able to write a text formatting component $\mathcal{TP}$, version 1.0, packages it up as a shareable library (DLL, or Java `.class` file) and ship it off to application developers Fuyuko and Estéfan. Fuyuko may use it to build an internet browser NITSCOPE, and Estéfan may use it in a game, DEADMEAT. So, now, what do I mean about not caring? I mean this: Ahmed shouldn't care who is using $\mathcal{TP}$; Fuyuko shouldn't care who *else* is using $\mathcal{TP}$, and likewise Estéfan. *Most importantly*, the users of NITSCOPE and DEADMEAT should never have to worry that NITSCOPE and DEADMEAT both use the same component $\mathcal{TP}$. Let us posit a poor user, Vic, (short for "victim") and track his plight. Let us assume that Vic has copies of both NITSCOPE and DEADMEAT. NITSCOPE was purchased for Vic, and supported by his employer's tech support staff, and DEADMEAT, he bought it on his own, though he shouldn't be running it on his company-bought PC.

Poor Vic.

[*]This paper includes content based on joint work with Michael Gertz and Stuart Stubblebine. Can someone from UMD tell me if/why so many of them are downloading this paper? Please?

Here's what typically happens. Estéfan finds a bug in $\mathcal{TP}$ that causes his application to break, and so asks Ahmed to fix it. Ahmed fixes it, and releases version 1.1 of $\mathcal{TP}$, and broadcasts an email to all his customers (including Fuyuko), announcing the bug fix release. Fuyuko, who is using 25 different components from 10 different vendors, ignores this one message (among the hundreds she receives that day) as a minor matter. Estéfan, on the other hand, downloads the $\mathcal{TP}$ 1.1, and notes that his bug has been fixed, and announces it to his customers. Vic has noticed that DEADMEAT keeps crashing just when he's about to blast the final mutant into smithereens, so he joyfully downloads the new version of DEADMEAT, over the weekend, and has a good time. A few hours, or days or weeks later, Vic starts up NITSCOPE to do some work on a spreadsheet downloaded form his employer. Alas, it doesn't work anymore. Well, it turns out that $\mathcal{TP}$ version 1.1, even though (or perhaps because) the change was "only" a bug fix, causes NITSCOPE to fail. Of course, Vic has no way to tell exactly what caused his failure. He assumes the problem is with NITSCOPE, so he calls up his employer's tech support staff to help diagnose the problem. You can imagine how the rest of this story goes, with Vic, his tech support helper, the help line personnel for NITSCOPE, etc all spending hours on the phone, and combing the web and USENET for a clue, any clue. Worst of all, in the end, Vic is forced to admit to the world that he has a copy of DEADMEAT on the machine that was provided by his employer for business use only. He gets reprimanded for misuse of company resources.

Poor Vic. "Component reuse? Bah humbug" he says. Can you blame him?

Most people have experienced this problem–also known as "DLL hell". The problem arises from successful reuse—each application is a complex assemblage of components, some of which are used by other applications or components. If component versions are incompatible in certain ways, downloading a new version of a component can have surprising and unpleasant effects. So, unfortunately, reuse means having to say you care! There are various solutions to this problem [1, 3, 4]. Ignoring the actual software binaries for a minute, the problem boils down to this: how do you get the information about the right configurations from those who have it to those who need it. There are various approaches, using technologies such as agents, "server push" and "client pull".

But there are several *security-related* issues that current approaches don't completely address:

- *How does a vendor know that a customer is legitimate?*
- *How does a customer* delegate *the configuration of specific aspects of his system to someone else ?*
- *How does a customer protect the privacy of the software installed on his machine, while giving administrators the information they need to provide needed configuration information?* [5, 6].
- *How does the customer make sure that the software he has received is really the right version?*

Solutions to these issues involve the application of cryptographic protocols and other security techniques. This is the subject of our current research. More details can be found in [2] (the paper is available on the first author's web page).

References

[1] Desktop Management Task Force. *Software Standard Groups Definition, Version 2.0*, Mar 1996. `http://www.dmtf.org/tech/apps.html`.

[2] Premkumar Devanbu, Michael Gertz, and Stuart Stubblebine. Security for automated, distributed configuration management. In *Proceedings, ICSE 99 Workshop on Software Engineering over the Internet*, 1999.

[3] Richard S. Hall, Dennis Heimbigner, Andre van der Hoek, and Alexander L. Wolf. An architecture for post-development configuration management in a wide-area network. In *17th International Conference on Distributed Computing Systems*, May 1997.

[4] Marimba, Inc. Castanet product family, 1998. `http://www.marimba.com/datasheets/-castanet-3_0-ds.html`.

[5] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communication Special Issue on Copyright and Privacy Protection*, 1998.

[6] P. Syverson, S. Stubblebine, and D. Goldschlag. Unlinkable serial transactions. In *Financial Cryptography*, volume 1318 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.