How to Test Software Security?

Adrian Kosmaczewski

2007-08-27

Howard and LeBlanc give a very complete answer to this question in their classic "Writing Secure Code" book:

Most testing is about proving that some feature works as specified in the functional specifications. If the feature deviates from its specification, a bug is filed, the bug is usually fixed, and the updated feature is retested. Testing security is often about checking that some feature appears to fail. What I mean is this: security testing involves demonstrating that the tester cannot spoof another user's identity, that the tester cannot tamper with data, that enough evidence is collected to help mitigate repudiation issues, that the tester cannot view data he should not have access to, that the tester cannot deny service to other users, and that the tester cannot gain more privileges through malicious use of the product. As you can see, most security testing is about proving that defensive mechanisms work correctly, rather than proving that feature functionality works. In fact, part of security testing is to make the application being tested perform more tasks than it was designed to do. Think about it: code has a security flaw when it fulfills the attacker's request, and no application should carry out an attacker's bidding.

(Howard & LeBlanc, 2003, page 568).

This impressive description is followed by a complete test plan for security, that can be summarized as follows:

1. Decompose the application in components: networking, user interface, database, etc.;
2. Identify the component's interfaces: network sockets, dialog boxes, databases, the Windows' registry, microphones, or even the clipboard! This will define the "attack surface" of the system;
3. Rank the above interfaces by potential vulnerability;
4. Identify the data accessed by each interface: network packets, environment variables, user input, command-line parameters, etc.;
5. Attack each interface with ad hoc techniques: network tools, badly-formed command line parameters, invalid XML data, long strings, malformed URLs, cross-site (XSS) scripts, etc. (Howard & LeBlanc, 2003, pages 570 to 613)

The framework proposed above shows clearly that security testing has a life of its own. “Classical” testing concentrates efforts in the functionality to be delivered by the final system, while security testing has an orthogonal set of goals, many of which are usually not considered part of the final deliverable, but might appear as fundamental later, when the application is deployed in the “real world”.

I consider security testing actually as one aspect of the broader concept of secure design: not taking security in the design process is a recipe for disaster. Software applications that have been designed with security in mind are actually able to deliver secure features, are strong when they are attacked, and are able to recover from or report about those attacks. The canonical comparison is that of the BSD family of operating systems and Windows Server; in sensible environments, BSD-derived operating systems (like OpenBSD, Mac OS X or NetBSD) are considered as virtually invulnerable, given the strong focus on security that they have been given from the very beginning:

OpenBSD believes in strong security. Our aspiration is to be NUMBER ONE in the industry for security (if we are not already there). (…) Like many readers of the BUGTRAQ mailing list, we believe in full disclosure of security problems. (…) Our security auditing team typically has between six and twelve members who continue to search for and fix new security holes. We have been auditing since the summer of 1996. The process we follow to increase security is simply a comprehensive file-by-file analysis of every critical software component. (…) Code often gets audited multiple times, and by multiple people with different auditing skills.(…) Our proactive auditing process has really paid off. Statements like “This problem was fixed in OpenBSD about 6 months ago’’ have become commonplace in security forums like BUGTRAQ.

(OpenBSD)

A search for “secure operating system” in Google brings OpenBSD as the third result (as of May 17th, 2007)

OpenBSD is often noted for its code auditing and integrated crypto, but the security features go far beyond this. OpenBSD was built from the ground up on the model of being a fabric woven with security in mind, not a patchwork of bug fixes and security updates. This has led to OpenBSD finally being recognized today for what it is: the most secure operating system on earth.

(ONLamp.com, 2000)

The most interesting statement in the OpenBSD citation above is

Our security auditing team typically has between six and twelve members who continue to search for and fix new security holes.

This shows that the OpenBSD team takes security very seriously at all stages of the production of the system. I think that more generally, when security is designed as a system feature rather than an afterthought, security testing cannot be distinguished from the overall testing effort, even if it consists of a special set of procedures.

BSD is not dying, as this funny video tries to convince us!

References

Howard, M.; LeBlanc, D., “Writing Secure Code, 2nd Edition”, ISBN 0-7356-1722-8, Microsoft Press, 2003

ONLamp.com, “An Overview of OpenBSD Security”, [Internet] http://www.onlamp.com/pub/a/bsd/2000/08/
(Accessed May 17th, 2007)

OpenBSD, “OpenBSD Security”, [Internet] http://www.openbsd.org/security.html
(Accessed May 17th, 2007)