

# A black-hat approach to testing security

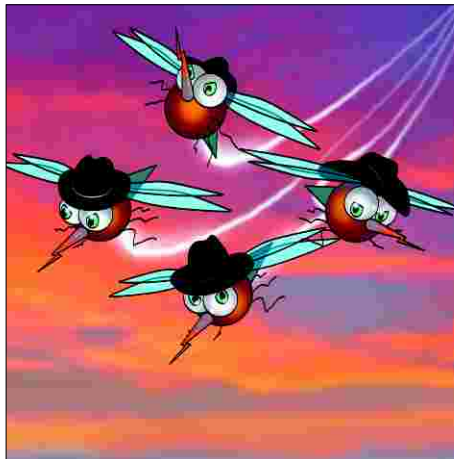
Bugs in code exist and will continue to exist. While in general testing can reveal a large portion of them, it tends not to catch them all. As a result, even the best tested software will contain bugs. When these bugs relate to security, the consequences can be very serious. That's why security testing probably has the highest ROI of all current testing methods.



## Mario de Boer of Collis

### About the author

Mario de Boer has held various information security positions within the Dutch Government and at Njama, a company he founded. He currently heads the security testing team at the Dutch-based IT development and testing company Collis, in Leiden.



**Should we care** about security bugs? In many cases we have no option. First of all, some software absolutely has to be secure. Home banking applications, copy protection schemes, software for routers or mobile services; all these are examples of applications that need careful security considerations from the requirements stage right through to maintenance. Security objectives nowadays are often increasingly governed by legislation or regulations. If the product has holes, you might face legal action, with economic consequences. But I would also care about bugs if I were the producer of a software product, because security has become a market discriminator. Customers will refuse to buy software without sound security practices. So you can't say you don't care because you virtually have to.

Security bugs are errors in the design or implementation of software that might be exploited to make the system act in a malevolent way. As web servers are prone to public attack, all software is susceptible to malicious attacks. Let me illustrate what I mean by two real life examples. Company X is releasing a database and software on CD-Rom and the amount of data accessible depends on the license. When asked what security measures have been incorporated - an important design requirement - the company asserts that modern technology has been applied, but does not provide any further details. The 'protection' turned out to be a copy protection scheme and in fact the protection against illegal licensing was dead easy to break. In this case - and this is a common problem - the security was provided by a third party. The whole issue was not quite understood by Company X, and thus the 'solution' was taken for granted, which was clearly a costly design error. Another example is published in the Master's thesis of one of my students, Sieuwert van Otterloo [1]. He discovered that in the source code of PGP, there was a small error with potentially big consequences. The error lay in part of the code that generated RSA keys. Two prime numbers,  $p$  and  $q$ , were being generated to satisfy certain assertions. The obvious way to implement this is to code the part for  $p$ , apply copy-and-paste, and replace the  $p$  in the copied part by  $q$ . The error made arose when one of the instances of  $p$  was not replaced, resulting in code in which prime  $q$  need not satisfy all assertions. After careful consideration, this did not lead to an exploitable flaw, yet it exemplifies a clear implementation error.

The examples above point to a need for two special skills in dealing with security in QA and testing: a Black-Hat [2] approach to analysing systems, and a thorough and broad knowledge of security techniques.

**QA in secure development - becoming standard practice**

These days virtually no software vendor would omit to build security into their development lifecycle. The practice has become pretty standard throughout the industry, certainly for the large vendors, who have all tightly integrated Security QA into their existing lifecycle. Best known is probably Microsoft's Trustworthy Computing Security Development Lifecycle [3]. Common to Security QA development programmes is the need for education for the developers backed up by a central security team, whether internal or third-party.

For developers who have not yet integrated security measures into their lifecycle, initiatives such as CLASP [4] (Comprehensive, Lightweight Application Security Process) by Secure Software may provide a valuable start. It lists some of the activities that are needed to assure the right security quality. Another interesting source for information is Corsaire's Secure Development Framework [5]. In fact, all in all there appears to have been a significant industry-wide effort to make development lifecycles evolve into more mature processes where security is paid due attention at every stage of the way. I wish the same were true for security testing.

**Security testing - still an immature field**

Although security QA has been given ample attention by the software vendors, security testing is still an immature field. Security testers often rely on ad-hoc approaches, the results of which depend on the expertise - or the marketing power - of the tester. Security testing tools often lack speed, extensibility, and ease of use.

At Collis, our approach to security testing follows the proven methodologies well known to professional testers. In Table 1 we provide an overview. Security testing is a time-boxing process, and the aim is to find as many of the important bugs in a given time as possible. The best way to achieve this is to conduct a risk analysis that will result in a prioritised list of things to test. Note that the items on the list can go beyond the application itself. Assets typically ending up on the shortlist of

security testing are the parts of code dealing with interfacing such as libraries, the network, human interaction, together with the physical and logical environment. Microsoft uses the term 'attack surface' to refer to this part of the application exposed to attacks. The complete list of parts of the system that need to be security tested is referred to in this article as a Target under Test.

**Static and dynamic security testing combined with manual review**

The approach we advocate combines static and dynamic security testing with manual review. Static security testing is aimed at uncovering bugs without executing the application. Design errors and requirements lacking can be discovered in the early stages of the development process by reviewing the documentation. This is why static testing achieves the highest return on investment (ROI) when actual flaws are found. Other static methods include source code review and reverse engineering. Both of these can and should be supported by automatic tools, of which there are a number of useful products available. When using these tools, however, the testers should be alert to typical problems that arise when using source code reviewing tools. These include high rates of false positives, low speeds, non-extensibility and the problem of determining the coverage and completeness of the tests. Provided that the selected tools are piloted carefully prior to deployment, and monitored

carefully throughout, they are great time savers. Having said that, bad tools are worse than no tools at all, and it is my belief that automatic tools will never fully replace manual review. Microsoft's Michael Howard shares his interesting pragmatic approach to manual security code review in an MSDN article [6].

In some cases source code is unavailable for a part of the Target under Test and manual review is necessary. But thanks to advanced reverse engineering techniques now emerging, the whole code reviewing process has become only slightly more cumbersome than in cases where the source code is available. With compiled code too, even, finding coding errors such as integer or buffer overflows can be automated. Of course, reverse engineering can only be performed in those cases where it does not infringe law or license agreements.

Dynamic security testing techniques aim to find security faults while executing the application. The technique most often mentioned in relation to these testing is fault injection. Injecting faults entails presenting the application with unexpected input from the user-interface, the file system, the environment (logical and physical), the network and the API's etc. There are tools that support this kind of investigative test. Since this type of testing is a typical example of exploratory testing - whereby test cases are defined while testing - many experienced security testers will have their own methods for injecting faults, together

	Action	Method	Deliverable
Test management	Determine scope	Risk analysis	Target of Test
	Perform static testing	Automated testing: Source code review Reverse engineering	Intermediate test reports containing bugs and regression tests
		Manual testing: Document review Source code review Reverse engineering	
	Perform dynamic testing	Automated testing: Fault injection Environment simulation	
Manual testing: Exploratory testing			
Finalization	Reporting	Test report	

Table 1 Simplified security testing process

with toolboxes that automate the process. When depending to a large degree on exploratory testing, the overall quality of the testing depends heavily on the experience and skills of the tester. Any security flaw should be carefully documented and a regression test should be compulsory to prevent the fault from being reintroduced in future releases. Retesting is even more important for security than for other tests, since new families of vulnerabilities and exploitation techniques are being discovered all the time. Remember the saying: "Old software never dies; it just becomes insecure." [7]

To revisit the examples provided at the start of this article, the poor security design in the database application should have been noted during the manual review of the documentation or the manual testing phase. It would not have escaped the approach I've outlined here. The second example, PGP, illustrates the benefits of performing a structural source code review. The emphasis should be on 'structural':

discovering a potential disastrous fault of one bit (replacing p by q) in 30 MB of source code [8] cannot be achieved by following an ad hoc approach. Detailed knowledge of security is necessary to fully understand the ramifications of such an error.

#### Why it's worth the trouble

The investment required for setting up security testing is low in relation to total testing time. The problem is hiring the right people. Security testing requires experience

and the right blend of expertise: knowledge of software vulnerabilities, software development, structural testing and a Black-Hat's view. Even if you involve only one of these ambidextrous testers in your development or testing project, it can make a large difference. For when you compare the costs of security testing with the costs of security vulnerabilities, you'll probably find that security testing has the highest ROI of all current testing methods ■

**Subscribe to**  
Professional Tester Magazine

boer@collis.nl

1. Sieuwert van Otterloo, Master's Thesis, University of Utrecht, <http://www.bluring.nl/pgp/pgp.pdf>
2. In information security, like in traditional Westerns, the good guys are commonly referred to as White Hats, and the malicious hackers at the Black Hats. A Black-Hat approach entails trying all you can to crash products, exploit holes and in general circumvent security measures.
3. Microsoft Computing Security Development Lifecycle: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsecure/html/sdl.asp>
4. Securesoftware: CLASP (<http://www106.ibm.com/developerworks/rational/library/content/RationalEdge/oct04/ieiga/index.html>)
5. Corsaire: Secure Development Framework: (<http://www.corsaire.com/white-papers/040220-secure-development.pdf>)
6. Expert tips for finding security defects in your code, Michael Howard, MSDN Magazine, <http://msdn.microsoft.com/msdnmag/issues/03/11/SecurityCodeReview/default.aspx>
7. Attributed to an unknown author in <http://www.cerias.purdue.edu/secprog/class1/1.security%20overview%20and%20patching.ppt>.
8. The version of PGP for Windows which has been researched (6.5.8) contains 30 MB of source code

CONTEMPORARY SOFTWARE

## Spotting the bug just got easier

With our Automated Testing software

As the dependency on Websites increases, more and more companies are in need of reliable and accurate Web Testing Software

Unprecedented ease of use, accuracy, scalability, power and performance.

Easy to Use, Cost Effective, Browser Based Solutions for Load and Performance Testing

- Wizard driven tools
- Robust Scripting
- Integrated Change Management
- Market leading Bug Tracking

**Contemporary Software**

Call Now on 0845 345 6846

Download evaluation at:

[www.contemporarysoftware.co.uk](http://www.contemporarysoftware.co.uk)

Demand the NDS difference

### Demanding Testers Needed!

NDS is a leading supplier of digital pay-TV solutions for the secure delivery of entertainment and information to television set-top boxes and IP devices. We are currently recruiting talented individuals to join the UK R&D division.

We have opportunities in both our Staines and Chandlers Ford offices for experienced testing specialists with the following skills:

- > UNIX/LINUX
- > Oracle (inc SQL)
- > XML
- > IP & MPEG streaming

If you are interested in joining a dynamic environment that uses cutting-edge technologies such as Oracle 10G and Redhat ES 3.0 in the delivery of mission critical real-time systems, send your CV with details of your availability and salary expectations to [testers@nds.com](mailto:testers@nds.com)

For more information about NDS visit [www.nds.com](http://www.nds.com)

