# An HTML Mangling Test case

On the 18th of October 2004, Michal Zalewski (http://lcamtuf.coredump.cx) posted on the Bugtraq security mailing list (http://www.securityfocus.com/archive/1/378632/2004-10-13/2004-10-19/0) the results of a small experiment he'd carried out. The experiment involved feeding randomly generated HTML pages into some of the web's most widely used browsers. The results were somewhat alarming. All of the open source browsers Michal tried crashed when faced with this kind of input. Only one browser stood up at all well against this type of stress test. Which one? A round of applause please for, er, Microsoft's Internet Explorer.

A Slashdot discussion followed the release of this information and there was much argument about what the test proved, what it didn't prove, and whether we should all switch to using Internet Explorer immediately on the grounds of improved security. If you like reading 1,000 post Slashdot threads go right ahead. This article will look at the test case, consider why it's a good one and what the free and open source software communities can learn from it.

# The Test case

Let's start by examining the test case. Michal's original code (http://lcamtuf.coredump.cx/soft/mangleme.tgz) is written in C, which is a little tricky for many people to deal with. To make it more accessible I rewrote it in Perl. If you want to give the Perl version <link here> a try, copy the script to an executable file called mangler_pl.cgi, place it in your web server's cgi-bin directory (/srv/www/cgi-bin on my SUSE Linux system) and point your web browser at it: http://localhost/cgi-bin/mangler_pl.cgi. You might want to finish reading this article first because your browser might not last too long!

The script is really very simple. It chooses, at random, one from a list of HTML tags, then chooses, again at random, one of the attributes which that tag would expect. A random value for the attribute is then generated. The value might consist of a random number, a text string of random length, or maybe one of a number of keywords which HTML renderers take a special interest in: "javascript:", "file:", "_self" and so on. Several attributes might be added to the tag in this manner, then a whole lot more tags will be generated in the same way. Occasionally, random characters might be inserted into the document at strategic points, quotes or brackets might be inserted, and so on. The final output, therefore, contains valid HTML tags, which have some of their expected attributes, and thus the HTML renderer in the browser gets to work. When it does so it encounters a whole stream of randomised garbage which it then tries to make sense of.

The test is automated by a clever use of an HTTP refresh command in the header of the generated page. When the browser has read a page from the CGI script and has completed rendering it, it automatically and immediately refreshes itself, so it gets another random page, then another and another. The browser might need a manual refresh or restart occasionally when it goes a bit wobbly, but for the most part can be left automatically rendering page after page.

The results of running this test case are distinctly sobering for anyone who uses an open source web browser: they all crash in some way sooner or later. Normally a generated web page causes a repeatable crash – that is, if you feed the same web page into the browser twice it will crash in the same way both times. Sometimes, however, the browser might crash on a page the first time, then display it correctly on a second attempt. This seems to imply a creeping memory corruption sort of problem as the test

runs, and I noticed that Konqueror in particular seems susceptible to this. It's not just the open source browsers either - the test case causes the significant closed source browser in the free software world, Opera, to crash on a regular basis too.
Most of the fuss from Michal's original post came from his assertion that Microsoft Internet Explorer stood up better than all of the free software browsers, but a little experimentation shows that this doesn't quite tell the full story. I tested various versions of Internet Explorer on Windows2000 and Windows XP and all of them, up to and including Internet Explorer 6 with Windows XP service pack 1, crashed as regularly as Mozilla and Konqueror did. The difference comes with Windows XP service pack 2: with this installed, Internet Explorer seems pretty solid, although in a later posting to Bugraq (http://www.securityfocus.com/archive/1/379207/2004-10-20/2004-10-26/0) Michal noted that this version of Internet Explorer did eventually crash after a 3 hour run.

## Some Serious Implications

Michal posted his original findings to the Buqtraq security mailing list because there are significant security implications of these results. Whenever a program crashes, and particularly when a program crashes as a result of externally sourced input data, the implications are *bad*. It is often an easy step for an attacker to take the randomly generated HTML and craft it more carefully into something that still makes the browser crash, but crash in a way that allows the attacker to take control of it.
Michal's program actually turns up several different types of problem, all of which produce a denial of service (i.e. a user's web browser crashes) and many of which might prove exploitable by people with evil intent. As Michal put it: "Because I did not ask CERT or NIPC for patronage … this will likely not generate any media splash, but I for one consider my findings to be far more chilling than a wave of tabbed browsing URL spoofing flaws and similar recent browser issues."
Security implications aside, the other significant issue raised by the test case is the resilience displayed by the latest version of Internet Explorer to this sort of random input. There are any number of explanations for this, and the Slashdot discussion probably mentions most of them. The most obvious, though, is that Microsoft is now concentrating on security issues, and we're seeing a direct result of this. Microsoft is aware that people launch this type of attack on its products because with no source code to pour through, blackhat crackers know that slinging random data at a program is a good way to find bugs in it. Since it's a known methodology of attackers, perhaps Microsoft now codes for, and performs testing for, resistance to this type of attack.

## Lessons for OSS Projects

There is nothing new or even particularly clever about this type of test case. Throwing random, but pointedly nasty bits of input at a program has long been known to unveil subtle bugs. There's nothing quite like a randomiser for inventing forms of input that no human would ever think to enter. What is interesting about this test case is that, clearly, no one has actually tried anything like it with any of the open source web browsers. Perhaps open source developers are so used to having the code to look at in order to find bugs that they forget some of the other techniques available to them.
Another observation is that despite the number of people who have access to the source trees of these browsers, no one has spotted that these bugs exist. It could be that web browsers are now just so complicated that not many people look at the source anymore; perhaps some bugs are just too deep to spot, no matter how many eyes are on the code; perhaps it's just a case of using the right tool for the job.
Whatever the explanation, it can't be good that such a simple test reveals so many

problems with so many projects. Maybe open source applications of all types should be looking at diversifying their testing procedures?
Also worth noting is that this isn't a complicated or particularly rigorous test. It's a bit of code knocked up by one hacker, and it only looks at the most obvious part of the browser – the HTML renderer – and even then it only works with a small subset of HTML tags and their attributes. It doesn't test stylesheets, Javascript, XML processing, or any of the other advanced features we now take for granted from our browsers. One has to wonder what other horrors lurk in those much newer and more complicated parts of the code bases.

## Conclusion

This test case raises immediate questions about the security in our web browsers, and demonstrates bugs that are, in at least a few cases, potential remote exploits.
Hopefully patches to the open source browsers and Opera will follow pretty quickly.
Let's not forget either, that many of these insecure HTML rendering engines are used in other applications such as email programs, as well as web browsers.
The test case also demonstrates that Microsoft might be getting their act together with regard to security issues. On this one, admittedly narrow, point, Internet Explorer is better than anything the open source community has to offer. This idea can't sit well with any open source software enthusiast.
Finally, the test case raises important questions about the quality of testing which all the open source web browsers have been subjected to. It's hard to avoid the phrase "not good enough". But instead of pointing the finger at the application developers, maybe all of us open source and free software enthusiasts should be asking what we can do to help with the testing of the software we use.