

## Microsoft compiler flaw, Cigital responds

*Source:* <http://www.derkeiler.com/Mailing-Lists/securityfocus/bugtraq/2002-02/0239.html>

---

*From:* Gary McGraw ([gem@cigital.com](mailto:gem@cigital.com))

*Date:* 02/15/02

From: "Gary McGraw" <[gem@cigital.com](mailto:gem@cigital.com)>  
To: "'[bugtraq@securityfocus.com](mailto:bugtraq@securityfocus.com)'" <[bugtraq@securityfocus.com](mailto:bugtraq@securityfocus.com)>  
Date: Fri, 15 Feb 2002 10:37:07 -0500

Brandon Bray claims that Microsoft Visual C++ compiler team "invented" the idea of placing a canary on the stack to try to prevent certain kinds of buffer overflows. Mr. Bray should familiarize himself with Crispin Cowan's StackGuard [Cowan]. Also of interest are various attacks against the original technique, most notably the Emsi vulnerability explained in Phrack 56 [Phrack].

We never made a claim that the use of the flawed /GS feature exposes code to "more attacks" as suggested in a bugtraq post. All we have done is point out that the /GS feature is itself susceptible to attack and should not be relied on to improve software security. The short term solution is quite simple: don't use the feature, or if you insist on using the feature at least know the risks! In other words, developers should be made aware of problems with the /GS feature so they know the risks they run if they choose to use it. Perhaps future versions of the /GS feature will be better constructed.

Why might developers rely on the broken /GS feature to protect their code? We refer you to Mr. Bray's article "How Visual C++.NET can Prevent Buffer Overruns" [Bray], where the title itself says it all. Also note that in Chapter 13 of Mike Howard and David LaBlanc's book is the important qualifying statement (p. 346) "doing so catches only stack-based buffer overruns that overwrite the function return address." That it does. But as our toy program shows, simply getting caught by killing the canary is not sufficient. We can "get caught" and still go on to carry out an attack. Two stage attacks are both possible and dangerous. By now, everyone should know this. To be fair, Mike Howard has always been very careful in his claims about the /GS option. Good for Mike.

There are well known ways to prevent a two stage attack. In our technical writeup, we mention three [Cigital], one prominent one being work done by IBM [IBM]. Also see Crispin's new work on PointGuard. (We expect these will be "invented" soon too.)

With regard to feature performance, a classic criticism against Microsoft is that there is a tendency to make hard tradeoffs like "efficiency versus

## SecurityFocus Bugtraq: Microsoft compiler flaw, Cigital responds

security" rather poorly. Mr. Gates promises to change this when he says "So now, when we face a choice between adding features and resolving security issues, we need to choose security." We applaud this sea change and wish Microsoft the best in this endeavor. Unfortunately, Mr. Bray's claim "if the security checks perturbed the code such that it was noticeably slower, I truly doubt anyone would use [it]." demonstrates the wrong attitude. Our hope is that cooler minds will prevail (go Mike!).

There is clearly room for improvement in the /GS feature. We believe that you should make use of some of the "extremely well documented" ways of protecting the canary, or just drop the whole idea. If you could "easily change" the /GS architecture to thwart our attack, then why the heck didn't you? This is not really rocket science. Don't make performance against security tradeoffs without informing the people who end up using your feature.

Just for the record, we agree with your stated position that the real solution is to educate developers about security. The book "Building Secure Software" by Viega and McGraw is one decent place to start. [BSS]

Instead of worrying about protecting flawed native code from exploit, we think that Microsoft should take this opportunity to emphasize "managed code" and encourage people to adopt this approach. Java intruded the world to the power of type safe languages. .NET "managed code" can help make this modern approach more pervasive. (BTW, this from the guy who wrote "Java Security" in 1996).

Code Safely,

Gary McGraw and Chris Ren

### REFERENCES

[Cowan] Automatic Detection and Prevention of Buffer-Overflow Attacks", Crispin Cowan, Calton Pu, David Maier, Heather Hinton, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, and Qian Zhang, in the 7th USENIX Security Symposium, San Antonio, TX. January 1998.

<<http://www.immunix.org/stackguard.html>>

[Phrack] Bypassing Stackguard And Stackshield

<<http://www.phrack.org/show.php?p=56>>

[Bray] How Visual C++ .NET Can Prevent Buffer Overruns

<<http://www.codeproject.com/tips/gsoption.asp>>

[Cigital] Microsoft Compiler Flaw Technical Note

<<http://www.cigital.com/news/mscompiler-tech.html>>

[IBM] GCC Extension For Protecting Applications From Stack-Smashing Attacks

<<http://www.trl.ibm.com/projects/security/ssp/>>

## SecurityFocus Bugtraq: Microsoft compiler flaw, Cigital responds

[BSS] John Viega and Gary McGraw "Building Secure Software", Addison–Wesley  
2001. <<http://www.buildingsecuresoftware.com>>

---

- *Previous message:* [Andrew Griffiths: "codeblue remote root"](#)
- *Next in thread:* [David LeBlanc: "ITS4 from Cigital flawed"](#)
- *Reply:* [David LeBlanc: "ITS4 from Cigital flawed"](#)
- *Reply:* [Paul L Daniels: "Outlook \r exploits – ripMIME fix."](#)
- *Messages sorted by:* [\[ date \]](#) [\[ thread \]](#) [\[ subject \]](#) [\[ author \]](#) [\[ attachment \]](#)