Microsoft

Life in the Digital Crosshairs

The dawn of the Microsoft Security Development Lifecycle

The dawn of Microsoft Trustworthy Computing

David LeBlanc, like all Microsoft staffers, occasionally got email blasts from Bill Gates. But never before had one mentioned him by name.

"I remember very clearly coming back to the office after a morning full of meetings," says LeBlanc, one of the first full-time security professionals involved in Microsoft's Trustworthy Computing initiative. "People were coming out of their offices and asking, had I seen Bill's mail?"

Sure enough, blinking away on LeBlanc's brand-new Windows XP office laptop was a message from the company's legendarily direct CEO. The security of Microsoft products was at risk, Gates wrote on Jan. 15, 2002. From that day forward, whatever software the company made had to be secure enough to earn a customer's trust.

"Trustworthy Computing is the highest priority for all the work we are doing," Gates wrote, defining a new initiative for the company. Over the next 1,500 or so words, he made it clear that the security and overall trustworthiness of complex software like Windows XP and Microsoft Office was now the job of every company employee. As he saw it, the integrity of not only these products, but the millions of lines of code the enterprise used to pay its staff and manage its finances, were part of the foundational infrastructure, like running water and heat, for modern computational life.

LeBlanc was impressed by Gates' inspirational tone. But what really made the email pop for him was that — about a third of the way through the message — Gates recommended that all the then roughly 50,000 worldwide Microsoft employees take a look at a certain book: LeBlanc's book. He and principal cyber-security architect Michael Howard had recently finished Writing Secure Code, a Microsoft-published text, and Howard had slipped Gates a copy at the end of a recent meeting.

The pair had written the book partly to fill a knowledge gap about what it took to write software with fewer, less severe vulnerabilities. The themes they laid out eventually helped define the Security Development Lifecycle — commonly called the SDL — that became the benchmark reference for how large groups can create as secure software as possible.

Writing Secure Code tackled basic principles like matching the right amount of data needed to fit into chunks of available memory or designing software without unnecessarily giving it the privilege to hijack an entire PC. It also detailed larger security concepts like anticipating risks before designing software and planning attack responses ahead of time. The details were complex, but the idea behind them was almost simplistic.

"I say it over and over: Software developers want to do the right thing," says LeBlanc, now a Microsoft principal software development engineer. "But they need to be shown exactly what that right thing is."

The spotlight that follows Bill Gates turned LeBlanc and Howard from scribes of a how-to manual for software developers into bestselling authors. Writing Secure Code became an instant bestseller on Amazon. "That was the exact moment when it all started, that Bill Gates memo," says Howard, now a principal cyber security architect at the company. "It was that big a deal. Everybody realized how they were going to do their jobs was going to be different."

"I remember very clearly coming back to the office after a morning full of meetings. People were coming out of their offices and asking, had I seen Bill's mail?"

David LeBlanc
Principal Software Development Engineer, Microsoft Windows

Microsoft goes Code Red

As bright as the limelight was, LeBlanc and Howard knew that a darker message loomed behind Gates' email: The company was under attack.

The world's software bad guys were no longer content to bash away at Microsoft's customers by the established means of breaching firewalls, subverting how data is transported around a network or gaining unauthorized access to computer terminals. Rather, this new generation of global network-savvy computer marauders was exploiting programming flaws in Microsoft software. In many cases, the software giant had released patches for these flaws weeks or months before, but computer users around the world often found them difficult to install.

On July 19, 2001, just six months before the Gates security email, a small firm called eEye Digital Security had noticed a nasty bit of self-replicating code — dubbed a worm. Internet lore says researchers named the bug "Code Red" for the flavor of Mountain Dew they were drinking at the time. Either way, this aggressive new form of digital infantry was quickly in business in a tiny, hidden crevice deep inside Microsoft Web servers that store, or buffer, data. Code Red took advantage of a so-called buffer overflow to store more data in a place than normal, giving attackers the means to deface a target website with "HELLO! Welcome to http://www.worm.com! Hacked By Chinese!" and to gain enough control over that machine to use it to spread the worm to other Web servers at will.

Not surprisingly, the story of out-of-control software straight out of a Tom Clancy novel gained instant media traction. One of the many news outlets that ran the story, ABC News, reported that more than 300,000 computers around the world were infected with Code Red in just two weeks — including critical computational infrastructure at the Department of Defense that was shut down to avoid attack.

"I think it's safe to assume that Code Red is the first of a new breed," Marty Lindner, a member of Carnegie Mellon University's Computer Emergency Response Team Coordination Center, told ABC News at the time. "And there will be more like it."

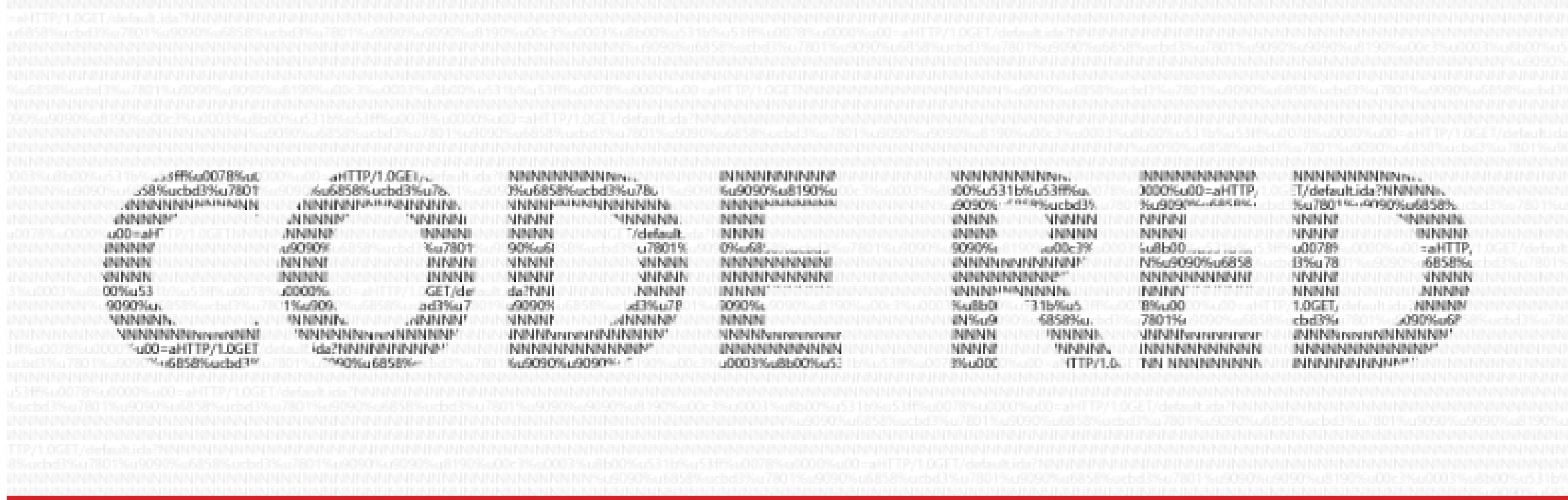
Lindner was right. Just six weeks later, Code Red was surpassed both in damage and in reach by a similar bug called Nimda. On Sept. 18, this particularly vicious bit of self-replicating software not only harvested emails en masse, but spread itself in shared files and as users clicked on infected public websites. It also took advantage of weaknesses in Microsoft's Web software products.

It did not help that Nimda struck just a week after the attacks of Sept. 11, 2001. Then U.S. Attorney General John Ashcroft went as far as issuing a statement quashing the suspicion that there was any connection between the two. But businesses had clearly had it with feeble Microsoft security. Chris Walker, a software engineer who managed early penetration testing efforts for Microsoft products, has vivid memories of being called into the office of Brian Valentine, then senior vice president for the Windows Core Operating System Division.

"I remember clearly him telling a room packed with Windows folks that the pain had to stop," says Walker. "He couldn't go talk to new customers without spending most of the time talking about security. And that was simply unacceptable."

Microsoft scrambled to issue patches and fix any issues it found. But security pros inside the company knew that reacting to attacks would not stop them. Nothing less than a ground-up security reboot was needed. "We all knew what the problems were," recalls Steve Lipner, then a director of security assurance focused mostly on threat management and mitigation. "But the real issue was, things were getting worse and worse. How were we going to get ahead of this?

"That's what we really had to go fix."



Internet lore says researchers named the bug "Code Red" for the flavor of Mountain Dew they were drinking at the time.

Tetris crashes a mainframe

Early application security professionals knew all too well why product security was so feeble. In these early days, not just Microsoft products, but every bit of computer software, did not prioritize security.

Arjuna Shunn came to the company as an in-house penetration tester, or pen-tester, whose job was to break into software before a bad guy does. Prior to his years at the company, he developed and managed a massive, multi-million-dollar server array — which amounted to working in a winter coat in a giant, refrigerated computer server facility. He and his team had just finished automating a complex management process on this server farm when, while waiting for tests to finish, they killed time playing a version of the video game Tetris. Their version — just for fun — was hacked to run on the tiny 6-inch screen that controlled the array of hard discs.

"We realized, quite by accident, that when that game crashed it exposed the 'root' control for the entire computer," Shunn says. Suddenly, that copy of Tetris was not so funny anymore. In stunned silence, Shunn and his colleagues realized that anybody with a free copy of a simple video game could take down millions of dollars of equipment and information. "That was an epiphany, spending the next three weeks fixing the system because of a mere video game."

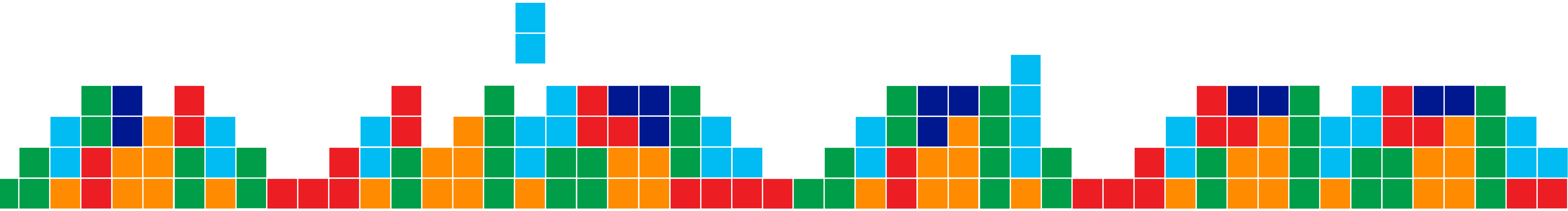
Even worse, there was no way in the late 1990s for a company like Microsoft to hold a meaningful cross-company conversation about software security. "We failed to find an existing taxonomy that could provide a framework for discussing Trustworthy Computing," Craig Mundie, then senior vice president and chief technology officer, wrote in a white paper on the topic that was circulated as late as October 2002. "There is no shortage of trust initiatives, but the focus of each is narrow."

That made the problem of securing Microsoft software almost incomprehensible. This was a truly massive company, with more than 8,500 developers on Windows alone who touched tens of millions of lines of code. But truly massive exposure emerged from nearly invisibly small problems. "Code Red, for example, was the result of an error in a single line of code," says Howard, co-author of Writing Secure Code. "But that's all it took — one line turned on that should have been off.

"That was how specific we all needed to start thinking."

"We realized, quite by accident, that when that game crashed it exposed the 'root' control for the entire computer. That was an epiphany, spending the next three weeks fixing the system because of a mere video game."

Arjuna Shunn
In-house penetration tester at Microsoft



Getting the AppSec band together

Microsoft, at least in the abstract, had committed real resources to the stoutness of its software from its earliest days.

There are solid accounts of security reviews, coding policies for individual products and even the occasional "Bug Bash," where coders would stop developing and focus intensely on fixing any mistakes they could find. Howard recalls that these early, non-centralized security efforts pioneered many core principles of modern secure software development at the company — including the basic, but critical, notion of finding mistakes in the code before the bad guys do. By the late 1990s, the security efforts began organizing themselves into small, unnamed security teams. These early pick-up bands of application security "studio musicians" would gig their way through various product groups at the company to raise awareness for software security, fix what they could and get developers in rhythm with the latest risks as they broke.

"As far as I know, that was the earliest effort inside the company dedicated entirely to application security," Howard says. "And I have to admit, it was fun work." Bashes were kept light-hearted. There were awards for finding the best bug, the worst bug and the bug written by the most senior person. "We made a big deal of having to fix the insecure code previously written by a vice president," Howard says. "You have to have these kinds of things to show that anybody at any level can make these mistakes."

Upper management began to see the value in investing in a full-time security force. Dave Thompson, who was vice president of Windows Server at the time and who recently retired after launching Microsoft Office 365, named these early security groups the Secure Windows Initiative, or the SWI. Security teams fluent in both the product being developed and the current state of application risk met in the morning and set a plan for the day. Then, depending on the threat level, they spent the rest of their day running automated tools, reviewing code by hand, re-engineering any security bugs they found and following up on past risks.

By all accounts, the Secure Windows Initiative made Microsoft's products safer. But all close to the effort knew that these small SWI teams were no match for an enterprise of Microsoft's scale. The company — and its products — were simply too big. "We could meet and code all day and night," Howard says, "and still not make progress in making the entire line of Windows products secure.

"It was tens of millions of lines of code we had to deal with."

"We made a big deal of having to fix the insecure code previously written by a vice president. You have to have these kinds of things to show that anybody at any level can make these mistakes."

Michael Howard
Principal Consultant Cybersecurity with Microsoft

Standing up by standing down

For all its limits, the Secure Windows Initiative remained the best security effort the company had for roughly the next 18 months. But even as early as 2001, it was clear these efforts were not the future of application security at Microsoft.

By mid-year, Howard and LeBlanc felt growing pressure to finish their book and formalize the process of how large groups could consistently write more secure software. "We joked about this, but it really was true. We had to get Writing Secure Code done to keep from going to the same meetings to answer the same questions," LeBlanc says. "There was not really a whole lot of knowledge in the industry about creating secure source code. We were laying down the rules as much for us as for our potential readers."

Toward the end of 2001, as the two were polishing off their final draft, a critical new approach for how massive armies of developers would make software more secure appeared in Howard's email inbox. It was a message from Loren Kohnfelder, a security lead on the then-radical new .NET Framework. The major idea behind .NET was to offer those who wrote software a set of consistent tools that could speed the application development process. But it would also attempt to centralize who was who on a network, no matter how big or complex that network was. .NET was a major focus of then-chief technology officer Mundie. And it was developed from the ground up as a showcase for how application security could be a major cornerstone of new Microsoft products. Kohnfelder was given the role of managing the security of .NET for a good reason: He was a serious security pro as Microsoft. His MIT thesis in the late 1970s had been to define the multi-headed system for managing trust on complex computer networks, called public-key infrastructure, or PKI. That system remains the backbone of Web security to this day.

Kohnfelder reached out to Howard with a serious problem. In spite of a project-wide commitment to secure software development in .NET, Kohnfelder and his team were seeing disturbing security flaws. "The issues they found were very specific to .NET," says Howard. "It took me several times through with Kohnfelder to even understand what was happening, but essentially it turned out they had to be very careful about how .NET code used a security feature called LinkDemand. If attackers knew what they were doing, they could exploit systems with something called a 'luring attack' to cause significant damage."

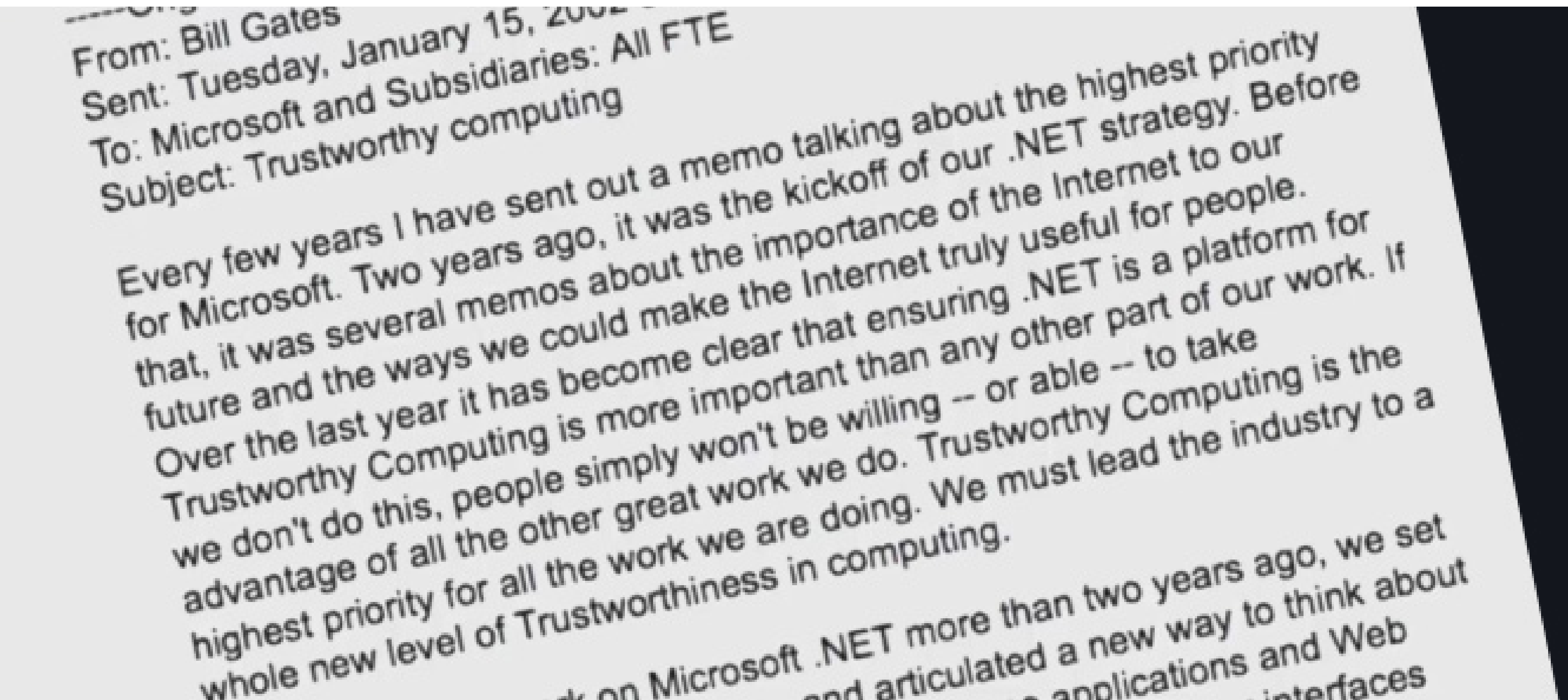
Considering the profile of .NET both inside and outside the company, Kohnfelder looked to Howard for ideas on how to be absolutely certain all the security issues — not just LinkDemand — were mitigated. Howard's answer was simple: a more in-depth Secure Windows Initiative-style Bug Bash. "But rather than lasting only one day, it would be done when it was done," he recalls, "With 'done' meaning the rate of incoming security bugs approached zero."

But since humans cannot bash bugs and write code at the same time, this non-stop, bug-blasting rave meant that all new development on .NET would halt — even though the framework was set to ship in just a few months. "There was a chance we could put the ship date in jeopardy," recalls Howard. For the first time in Microsoft's history, an entire product team was out of the software-writing business — and in the software security business.

"We all got the message: If you are going to take developers off code and put them on security, that affects schedule. And that affects your business," says LeBlanc, Howard's co-author. "It showed us there was real executive buy-in to the importance of security."

The Bug Bash that ends when its ends was even awarded its own Microsoftian jargon and swag. It was dubbed "The .NET Security Standdown" and t-shirts were made with the date it was set to begin. "The big joke, of course, was when we were supposed to start, there was a massive snow storm," Howard recalls.

"So nobody could get to work. It actually started a few days later."



From: Bill Gates
Sent: Tuesday, January 15, 200
To: Microsoft and Subsidiaries: All FTE
Subject: Trustworthy computing

Every few years I have sent out a memo talking about the highest priority for Microsoft. Two years ago, it was the kickoff of our .NET strategy. Before that, it was several memos about the importance of the Internet to our future and the ways we could make the Internet truly useful for people. Over the last year it has become clear that ensuring .NET is a platform for Trustworthy Computing is more important than any other part of our work. If we don't do this, people simply won't be willing -- or able -- to take advantage of all the other great work we do. Trustworthy Computing is the highest priority for all the work we are doing. We must lead the industry to a whole new level of Trustworthiness in computing.

k on Microsoft .NET more than two years ago, we set
nd articulated a new way to think about
applications and Web
interfaces

"We all got the message: If you are going to take developers off code and put them on security, that affects schedule. And that affects your business. It showed us there was real executive buy-in to the importance of security."

David LeBlanc
Principal Software Development Engineer, Microsoft Windows

Opening the window to Windows secure code

As 2001 wound down, so did the .NET Standdown. And the company tasted the first fruits of giving security its due.

The Secure Windows Initiative and the security push in Kohnfelder's group began formalizing several foundational principles that would later become the core for the Security Development Lifecycle. Among them was reducing the so-called "attack surface" by reducing the amount of code exposed to attackers, turning off unneeded features and making sure applications lack the privilege to take over an entire computer.

.NET was progress, but it was obvious that application security was still in its infancy at Microsoft. The focus quickly shifted from .NET to the efforts of security director Steve Lipner, who had spent much of his career developing theoretically secure computer software for the U.S. Air Force and Digital Equipment Corporation. After being recruited to manage Microsoft's Security Response Center, which had responded to security holes as they are discovered, Lipner began to pull together security-minded employees and resources from across the company.

"I really think he was the quiet power behind security at the company at the time," says Walker, who has since retired from the company. "He had the credibility to get the right people in meetings. He did not back them into a corner. And they listened."

Lipner, who these days is partner director of program management in Trustworthy Computing, began to organize the various security efforts in Microsoft's Windows development into a three-tiered system: a developer-focused security arm manned by Howard, a program manager-aimed group led by program manager Jason Garms and a testers group led by Walker.

"It was clear that the Bug Bash model was not going cut it," Lipner says. "We were stuck in the mode of reacting to whatever hit us."

By November, the company-wide pace of development for application security began to quicken. Lipner and his team figured it was a good time to use a smart Microsoft management technique: Take a moment to stop and think. It was widely known that every six months, Gates got in touch with his own, inner smart self by spending so-called "Think Weeks" at his Wyoming wilderness lodge. Similarly, company executives routinely had offsite meetings to plan in peace and quiet. Lipner, Walker, Howard, a lead program manager named Glenn Pittaway and a then security response lead Scott Culp, planned an all-day meeting at Robinswood House, a popular wedding and bar/bat mitzvah venue in Bellevue, Washington.

"It was a nice, wooded spot," says Pittaway. "And there we were, in this rural, wood-paneled room, and we could really get away for a few hours and just make sure we were doing the right thing. At some point somebody said — and as far as I know nobody remembers who — that if the standdown had worked for .NET, could we do it for all of Windows? Which, of course, was ridiculous because .NET was, at most, a few hundred developers and Windows was this 9,000-person supertanker."

Lipner realized quickly that in order to get the buy-in needed for such a product-wide security reboot, the growing security effort would have to find support from the highest levels of Microsoft's corporate command. Lipner began his internal due diligence with a direct meeting with Doug Bayer, who was then director of Windows Security and responsible for specific security features in Windows products, like how authentication is managed. That led to a meeting of Lipner and Bayer with Windows Server vice president Thompson. And that, in turn, led to larger meetings with Windows senior VP Valentine, and finally to a meeting with Jim Allchin, who ran the larger Platform Group.

Lipner had learned in his previous application security lives that absolute security was futile for large software packages like Microsoft Windows. What the company sought, rather, was, within the practicalities of shipping real products to real customers, to build a much more secure line of Microsoft offerings. To achieve this reasonable level of security, standing down nearly 9,000 Windows developers for at least a month was probably the company's only option. The higher Lipner went, the more the notion of a Windows-wide standdown took hold.

"There was no dramatic meeting that I can recall," says Lipner. "At some point the tenor of conversation went from talking about standing down to actually planning to stand down."

All this application security movement among the rank and file caught the attention of Microsoft's top management. During the second week of December 2001, Howard and Bayer got on the calendar of the most important Microsoft executive of them all. Bill Gates invited the two to his surprisingly modest office for a hands-on breakdown of modern security flaws. The executive was so fascinated by the details of how exploits took down contemporary Microsoft products that they chatted for more than two hours.

"I had a deck of about 30 slides," Howard remembers. "But we spent so much time talking, I don't think we covered more than 25 percent of them."

As the meeting came to an end, a surprising bit of old-fashioned paper and ink took over the discussion. Howard had brought along a copy of his and LeBlanc's newly finished book, Writing Secure Code, on the off chance the topic might come up. At the end of the meeting, he handed Gates a copy.

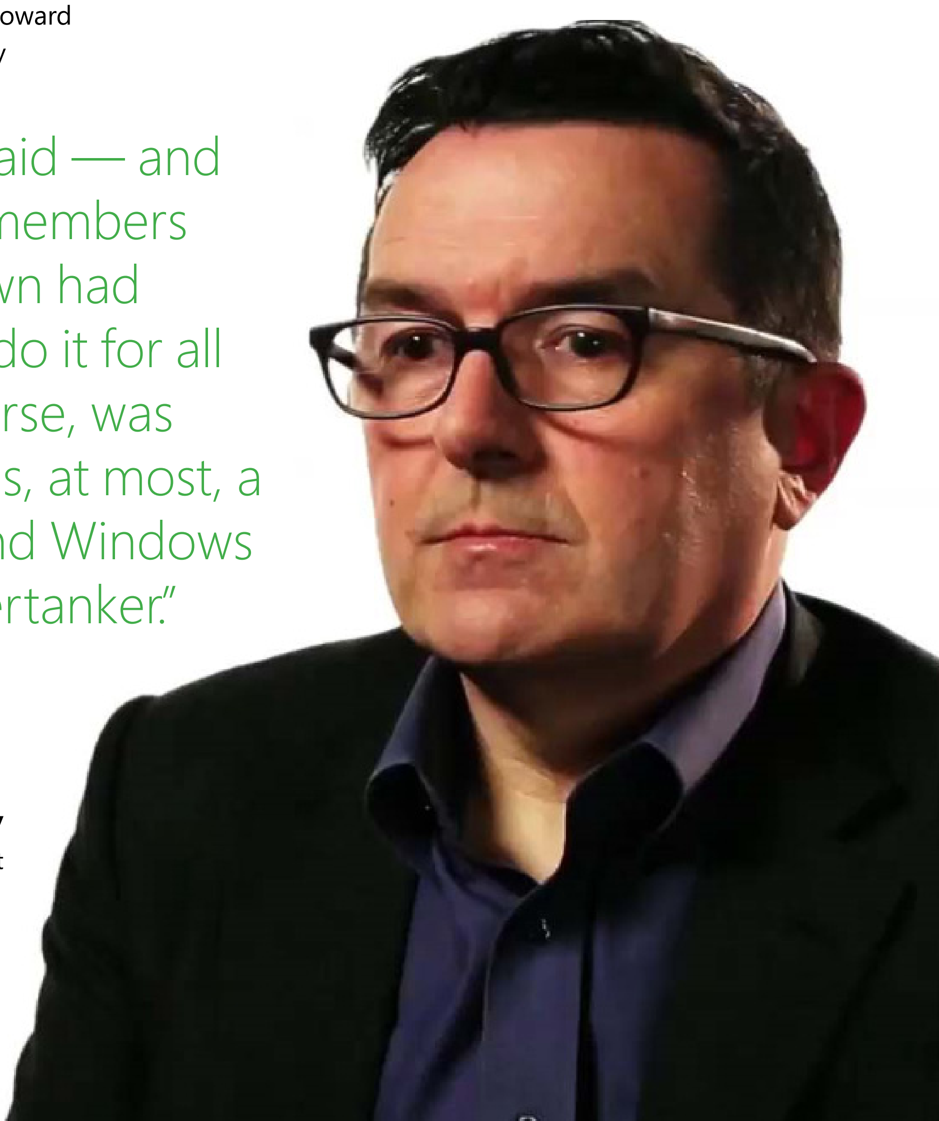
"I thought it would be a nice gesture," he says.

Gates digested what he had discussed with Howard and Bayer, and what he was hearing from executives like chief technology officer Mundie and vice president for the Platforms Group, Jim Allchin. Meanwhile, Lipner was huddling with his team. Their task was working essentially around the clock creating and prioritizing the security training materials and tools the software groups would need.

"Considering the work involved, the informal name of what we were doing was a 'security push,' and it stuck," says Lipner. "That is pretty much what it was. And to this day, that was what this period in the company is called: the Security Push Era."

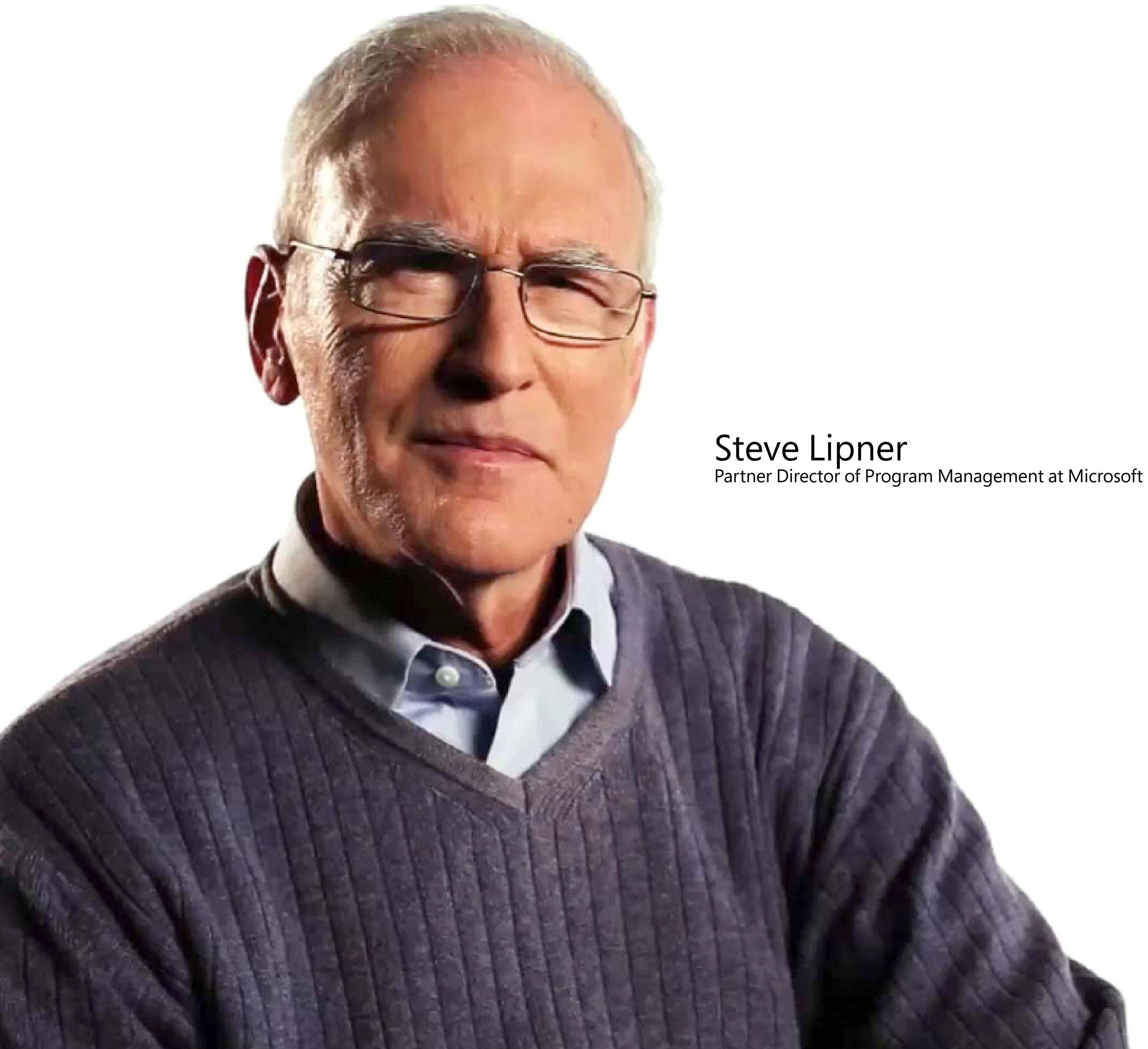
By January 2002, Gates' watershed company-wide security email rolled out. Howard and LeBlanc's book jumped to the top of Amazon's bestseller list and, seemingly overnight, the secure application software development era officially began at Microsoft.

"At some point somebody said — and as far as I know nobody remembers who — that if the standdown had worked for .NET, could we do it for all of Windows? Which, of course, was ridiculous because .NET was, at most, a few hundred developers and Windows was this 9,000-person supertanker."



Glenn Pittaway
Senior Director of Software Security at Microsoft

“Our approach to security was chaotic then. We were learning as we went.... It was a grind. We knew we needed a better way.”



Steve Lipner
Partner Director of Program Management at Microsoft

Fitting 8,500 developers into a 950-seat box

Despite widespread agreement that a Windows standdown was needed, the practical logistics of making it happen were daunting.

For starters, 8,500 developers, testers and program managers had to attend a four-hour software security training lecture on the security push process and building secure software — in a room that only sat 950 people.

“We had to do the presentations 10 times,” Lipner says. “We all got pretty tired by the last two or three.”

Then there was the problem of deciding what was a bug and how to fix it. To streamline the process, the team used the “War Room” on the third floor of the Windows development facility. There, developers would bring lines of potentially vulnerable code for review and debate. Was the code a bug at all? And if so, did it merit repair? While most issues were found and fixed by common consent, “There were times we would go to those meetings and lose,” says penetration tester Walker. “And we were not above going over that group’s head to find a vice president to approve the fix if we felt it should be there. There were fights.”

In those early days, the tools and texts Lipner and his team asked developers to use were still in development. There was Howard and LeBlanc’s book, and the company had some analysis tools like PREfix and PREfast. By all accounts, they were helpful. But many systems were in development. “People would throw some of the tools we gave them back at us because they really didn’t work,” says Lipner. “I knew how they felt; our approach to security was chaotic then. We were learning as we went.”

All these practical issues took time. The initial Windows standdown was budgeted for one month. It took two. Windows development picked back up by mid-2002, but the security battle was fought again and again over the next 24 months as a series of security pushes was rammed through the different units at the company, including SQL Server, Exchange Server and Office. And all the time, the budget meter was running: The cost in human resources was enormous. The oft-repeated internal figure was that in 2002, the company spent $200 million on the Windows Security Push alone.

But most of all, the security troops were getting threadbare. Though all involved say this was a three- to four-month period of remarkable focus and energy at the company, the level of work would often come in unsustainable chunks. All-nighters, back-to-back training sessions and endless email reviews were all in a day’s work during the most intense periods of some security pushes.

“We would have a morning meeting of the War Team that would update us on the bugs we fixed and then the progress of fixing other components,” Lipner says. “Then the day would consist of meetings to review the issues in various parts of the Windows development group. That would all end at 6:30 or 7 and I would go home, see my wife and have dinner — and then go to my study and try to catch up on 300 emails.

“It was a grind. We knew we needed a better way.”

Getting to writing something down

The Security Push Era wound down essentially under its own weight. At some point, managers ran out of major product lines to run pushes through. And those at the center of the company's emerging security culture realized Microsoft could only go so far to meet its security challenge when efforts were aimed only at specific products. What was needed was a day-to-day process for creating secure software. By late 2003, then-a-CTO Craig Mundie mandated exactly that. He called a meeting in a green room in the Microsoft Conference Center with, among others, Lipner and Mike Nash, then a company vice president.

"This security problem isn't going to go away," those who attended the meeting recall Mundie saying. "You need to step up and put something on the books that'll be permanent and formal."

For once, luck was on secure development's side at the company. Almost by accident, the formal documentation Mundie wanted was already coming together. Besides the presentation materials from various security pushes, there were droves of notes teams had collected detailing what worked and what didn't. For example, a globalization team essentially unrelated to the core Windows security process had updated the materials products needed for international markets. "We taught them how to do security, and they matched that learning to their expertise," Howard says. "Without any help, they created a unique product in the industry at the time. It was really awesome."

In early 2004, Howard and Eric Bidstrup, another program manager, began using a basic, 20-page document that had been created earlier to organize a secure application development liturgy. Progress, by all accounts, was swift. "This really was the early prescriptive phase of the application security culture," says former pen-tester Shunn, now a principal program manager for Trustworthy Computing. "So getting that down into a manual everyone could use turned out to be not that complex, compared to other challenges that were faced."

By mid-2004 it was clear to Lipner and his group that the existing written material could be organized into a legitimate security curriculum. The name Security Development Lifecycle was chosen as a riff on the generic title for creating code — the software development lifecycle. And in a strange bit of Microsoft jargon, this early version was called SDL Version 2 — since Version 1 was the then-unnamed Security Push Era of the previous two years. Once again, security director Lipner began walking the process through the company to gain formal buy-in for baking the Security Development Lifecycle into how Microsoft created all its software. By March of 2004, Lipner found himself in CEO Steve Ballmer's executive office meeting room at Microsoft headquarters.

Lipner recalls being in the room when the leadership team formally decided to authorize the creation of a procedure that would mandate how code would be securely created from then on. "I remember Ballmer turning to me and making clear to the entire leadership team that we weren't going to talk about this again," Lipner says. "It was now policy."

Sometime around July 2004, the formal Security Development Lifecycle, or SDL, went into active distribution. All the combined software security wisdom of the past half-decade was in place: The initial "find the bugs and kill 'em" ethic of the early Bug Bashes. The smart design concepts first tested in the .NET Standdown, which created design requirements for secure software. And the initial training and tools developed during the Security Push Era. It works now as it worked then. Once the SDL is running, various groups adapt its specific components for their needs. Important new features are injected into the knowledge base every six months to a year, such as when major new concepts like fuzz testing and improvements in threat modeling must be included.

"You realize that you are never going to get it perfectly secure," says Lipner. "You make things better, you make vulner-abilities harder to exploit and apply a strategy of continuous improve-ment. And if you stick to that, things really do im-prove."

7 Phases of SDL

1
Training
The basic concepts for building trusted software starts with education of developers.

2
Requirements
Defining a broad set of security & privacy standards from the start helps a team apply important guard rails throughout a project.

3
Design
To reduce the number of costly patches post-launch, security measures specific to the product are integrated into the software's overall structure.

4
Implementation
Thorough testing and analysis of the software product teams at this stage significantly reduce time-consuming fixes later.

5
Verification
Now in beta, the software undergoes rigorous checks on many levels, including security reviews more strict than the implementation phase.

6
Release
A few months before its public release and essentially written in full, the software goes through an independent final security review that checks all previous and current security issues.

7
Response
After the product is shipped the focus shifts to responding to any reports of vulnerabilities that emerge. Teams track and respond to any incidents to help protect customers, and any findings are fed back into the SDL to help improve future products.

The SDL Battleship sails a new course

Once Microsoft started using the Security Development Lifecycle, there was no stopping it.

The approach gave teams a formal start and end point for creating secure code. And even when previously unknown security issues were discovered, the approach still gave teams a well-proven process for solving these emerging problems.

"Obviously, there is a procedure for getting new ideas plugged into the SDL," says LeBlanc, who co-authored with Howard the book that ultimately became part of what evolved into Microsoft's development cycle. "But really, what happens is very human." LeBlanc explains, for example, that his pet peeve is programmers who use potentially risky application programming interfaces — or APIs, as they are known in the software trade — without fully understanding the risks they hold. LeBlanc takes it upon himself to track lists of such banned interfaces. As that list grows, he reaches out to those in Trustworthy Computing to talk through issues, including modifications of interfaces to be banned by the Security Development Lifecycle.

"There has to be a business review, of course," says LeBlanc. "But essentially, developers are making the argument for security in their specific areas as they go."

Over the years, these incremental shifts have been collected into roughly a dozen upgrades. Privacy features were added in 2005. Automated support for tracking the SDL process itself was injected a few years later. And 2010 saw a new version of the SDL that supported "agile development" styles popular among Web developers.

The process is not — and never will be — foolproof. As long as customers demand complex software products, the creators of those products will struggle to make them perfectly secure. Security breaches will happen. So will malware attacks. In fact, much of Writing Secure Code is as germane today as it was at the dawn of the millennium.

But even so, those who purchase from, collaborate with or track Microsoft's software security progress agree that the company has fresh security street cred.

First of all, Microsoft's in-house Exploitability Index, which tracks the ease with which vulnerabilities in its products can be compromised, saw that between July 2012 and June 2013, the most easily exploitable vulnerabilities declined by 40 percent.

Even those who were once the most critical of Microsoft products have been vocal about the company's improved security posture. Marc Maiffret, chief technology officer for BeyondTrust, the Phoenix, Ariz.-based software and IT security firm — who, in part, made his reputation detecting the original Code Red exploit that decimated Microsoft products years before — wrote in a New York Times op-ed piece that Microsoft had "changed its software development process to make security a core part of the program."

And it is clear that, despite all the pain and massive investment, Microsoft made a clever business bet on security, says Jeff Williams, founder of Aspect Security, the Columbia, Md.-based security firm, and one of the early writers of the Open Web Application Security Project Top Ten list of Web software vulnerabilities. "Bill Gates is a super-smart man. And he realized that security was an area where the company could win," Williams says.

Without a doubt, the SDL has been positive for companies that adopt similar approaches. There are well-documented success stories at Cisco Systems; Des Moines, Iowa-based utility Mid-American Energy; Liberty Lake, Wash.-based smart-meter maker Itron; and even the Government of India. Executives at America's foremost technology companies have also been public backers of the principles of the SDL.

"Our Secure Product Lifecycle is analogous to Microsoft's Security Development Lifecycle," says Brad Arkin, chief security officer at Adobe. "We value this process and the information it helps to protect."

There is palpable evidence that the SDL is adding value to even the newly minted corners of the security ecosystem. New York, N.Y.-based Canary, which is rolling out a crowdfunded consumer electronics home security device, gives credit to programs like the SDL for raising security awareness among a new generation of smart consumer electronics.

"Some of these are fundamental problems that could have been avoided with proper programmatic disciplines, which is why SDL and similar processes are invaluable building blocks," says Ken Garland, Canary's senior development operations engineer.

But for those inside Microsoft's security culture, the best success stories seem to run on a smaller, more personal scale. Lipner, for example, participates in the Chief Security Officer Council, a Microsoft event at which major customers come to give feedback and seek advice. About three or four years ago, Lipner gave a talk about new developments in the SDL and, afterwards, an attendee came up to ask a question.

"Here was someone from a company I knew of — a fairly major software vendor — but neither I nor anyone in my team that works with companies adopting the SDL had ever been in touch with them," Lipner says. "It just blew me away: Here was a moderate-sized business that I have never dealt with directly that had just picked up the SDL guidance from the Web, ordered a couple of copies the book — and then just gone off and done it.

"That just blew me away."

"That is when I realized that if anyone is going to remember my work, it is not my time doing theoretical security projects," he says. "It was what I did with the SDL, where we finally figured out how to take concrete steps to make real software more secure."

"Is the SDL perfect? Of course it's not," he says. "But it's simply a bigger contribution to real-world security.

"It just is."

"That is when I realized that if anyone is going to remember my work, it is not my time doing theoretical security projects. It was what I did with the SDL, where we finally figured out how to take concrete steps to make real software more secure."

Steve Lipner
Partner Director of Program Management at Microsoft

1.000.000

Since its inception in 2004, and the external release of SDL tools and resources in 2008, Microsoft's SDL guidance has been downloaded more than 1 million times and reached more than 150 countries. From small developer shops to large enterprises, many are seeing benefits from a "baking security in" approach.