Slammer – the SQL Server worm

David Litchfield, October 2010

Sophie's grandfather had just finished regaling us with a story about how he, during the second world war, had managed to avoid reveille aboard his ship every day he served when my mobile phone started buzzing in my pocket. We were all out celebrating Sophie's 23rd birthday at our local Chinese restaurant in Cheam, lingering over dessert - toffee banana and ice-cream. Looking at my phone, I excused myself from the table and took the call; it was my brother.

> "David, it's happened! Someone's released a worm."
>
> "Worm? Worm for what?"
>
> "Your SQL bug"

My stomach dropped. Telling Mark I'd call him back later I rejoined the table. Someone, I can't remember who, asked if everything was alright. "Not really," I replied, "I think there's going to be trouble." The day was the 25th of January 2003 and Slammer had just been released. Slammer was the fasted spreading Internet worm ever and it took advantage of a buffer overflow vulnerability in Microsoft SQL Server 2000 to replicate.

Of course, the story of Slammer started much earlier than that. For me it began on the 22nd of May 2002. Myself, my brother Mark and fellow co-founders of NGSSoftware Chris Anley and Sherief Hammad were on an client engagement for a bank in Frankfurt, Germany. The guy who'd hired us, we'll call him Mr M., wanted us to unleash bloody hell on his SQL Servers. They were locked down extremely capably and it would require something new to penetrate them so that was my task. Having heard of a tool called SQLPing, written by Chip Andrews, I wanted to know what it did so I downloaded it and fired it off against one of the targets in my mini-lab. Using Network Monitor I captured the traffic and noticed that all it did was simply send a single byte packet with a value of 0x02 to UDP port 1434 on the SQL Server. Well, the obvious question was what happens if we send a single byte with a value of 0x01 - or 0x00 for that matter and all the other values up to 0xFF (255)? I quickly wrote a short C program to investigate this. Once compiled, I fired it off against my SQL Server and by packet 0x08 the SQL Server had died. Now that really grabbed my attention. I restarted the server and this time started debugging it. On byte 0x08 the SQL Server crashed again and I disassembled the code to work out what was going on. In this particular case a call to the strtok() C function was made looking for a colon character in the user supplied data but, there being none, the function returned NULL. The fault in the code was not checking the return value of strtok() which, all going well, returns a pointer to the "token" being searched for. The code in SQL Server assumed a valid pointer was returned and the next line of code attempted to use this NULL pointer which caused the crash. So, next I adjusted my little C program to send 0x083A - byte 0x08 followed by a colon. The server still crashed this time in the atoi() function. atoi() take a string number and converts into a proper number and by not providing one in my send string server crashed again. I was beginning to see what was going on here - SQL server was looking for a hostname followed by a port number separated by a colon. So I gave it exactly that - but with an overly long host name and -

bang - an exploitable heap overflow occurred. But rather than simply stopping there and writing an exploit I went back and re-examined the code and found a number of other flaws. For example, if you send 0x0A as the first byte to a SQL Server's UDP port 1434 it'll respond with 0x0A to your source port. Thus, by spoofing the IP address of one SQL Server and setting the source port to 1434 and firing off an 0x0A to another SQL Server they'll start hammering each other with 0x0As in a storm of "pings". Another bug, and this is the key one as far as Slammer is concerned, happens when the first bye it 0x04. Anything after the 0x04 is sent to the sprintf() function in the process of building a registry key to open. The sprintf() function takes input and formats it as output into a destination buffer. In the case of SQL Server this is a fixed sized buffer on the stack and this lead to a classic stack based buffer overflow - one which was trivial to exploit. Coding an exploit up I sent a copy of it to the Microsoft Security Response Center (secure@microsoft.com) with a short write up of my findings then proceeded to own all of our client's SQL Servers. Job done.

Microsoft were extremely responsive and they informed me they had reproduced the issues and were working on a patch. I asked them if it would be ok to speak about these flaws at the upcoming Black Hat Security Briefings and they said they'd have the patch ready in time and gave me the go ahead. Sure enough Black Hat came and Microsoft shipped the patch. During the conference I warned that if people did not install this patch that these flaws could be the vector for the next big worm. 6 months later someone took what I said proved me right. They took more than that though. As part of the presentation I demoed my proof of concept code showing how easy it was to exploit the flaw. Whoever wrote Slammer used my code as a template and, at the time, this really gave me pause for concern. Slammer took down parts of the Internet because it spread so quickly, indirectly causing a denial of service attack and, in certain parts of the world, this wreaked havoc with essential services. For example, in Washington state, the 911 emergency services system went down. Whilst they reverted to good old paper and pen , this could’ve caused real problems for real people and it was a bit of a wake up call for me at the time. For me, what had been a fun, intellectual pursuit suddenly had real consequences and I had to carefully consider how I would go forward when publishing details of any more security flaws I found. They never did catch the person who wrote Slammer but I do have an interesting theory on this. A couple of years after Slammer was released I was asked by Microsoft whether, in my opinion, it had been written to be as small as possible and so I took a deeper look at the code. I concluded it had not. For example, part of the code set the ECX register to the value 0x9B040103 and it does so using the following instructions:

```
Machine code            Assembly code

33 C9                   xor   ecx,ecx

81 F1 03 01 04 9B       xor   ecx,9B040103h
```

This could be replaced with the shorter

```
B9 03 01 04 9B          mov   ecx, 0x9B040103
```

We can see that the method that Slammer uses is 3 bytes longer than it needs to be but that's not what's interesting. What is interesting is that half of the Slammer code uses the XOR method to set

register values whereas the rest of the code uses the MOV method. Coders generally have styles they stick to and, in the same way that radio operators during the second World War could be recognized by their distinctive style, or fist, a coder could be recognized by their style and they would tend not to switch between using XOR and MOV; and what's interesting about Slammer is that there are at least two styles at play here suggesting that there is more than one author. But that's just a theory.

As I write this in 2010, Slammer is still out there, nearly 8 years after release, still doing the rounds, so to speak. This is at worst an annoyance as, fortunately, Slammer had no destructive payload but it does suggest that there are still unpatched SQL and MSDE installs out there. This is incredible to me but unpatched systems are definitely few and far between. One positive aspect of Slammer was the effect it had on patching – prior to Slammer I'd guesstimate, from the results of penetration tests and so on, that 9 out of 10 SQL Servers were unpatched. Immediately after Slammer this reversed leaving 1out of 10 unpatched. Patching was 100% effective in preventing reinfection and so, in its own ironic way, Slammer helped make the Internet that little bit more secure. Another positive effect Slammer had was the sea change it helped cause at Microsoft as far as security was concerned. As I hear it, all development of Yukon (SQL Server 2005) was put on hold and everyone on the SQL team went back to SQL Server 2000 and pored through the code looking for flaws. And then they did the same for extant Yukon code. Doing this paid back massive dividends for Microsoft. The first major flaw to be found in SQL Server 2005 came over 3 years after its release – a heap overflow found by Brett Moore, triggered by opening a corrupted backup file with the RESTORE TSQL command. So far SQL Server 2008 has had zero issues. Not bad at all for a company long considered the whipping boy of the security world. As it happens, Slammer came 1 year and 10 days after Bill Gate's Trustworthy Computing Memo so it would seem that Microsoft were already on the right path and they'd have gotten to this state of affairs without the wee nudge from Slammer.