

# [NT] SQL Server Text Formatting Functions Suffer from Buffer Overflows

*Source:* <http://www.derkeiler.com/Mailing-Lists/Securiteam/2001-12/0106.html>

---

*From:* [support@securiteam.com](mailto:support@securiteam.com)

*Date:* 12/24/01

From: [support@securiteam.com](mailto:support@securiteam.com)

To: [list@securiteam.com](mailto:list@securiteam.com)

Date: Mon, 24 Dec 2001 17:35:24 +0100 (CET)

The following security advisory is sent to the securiteam mailing list, and can be found at the SecuriTeam web site: <http://www.securiteam.com>

-- promotion

When was the last time you checked your server's security?

How about a monthly report?

<http://www.AutomatedScanning.com> -- Know that you're safe.

-----

SQL Server Text Formatting Functions Suffer from Buffer Overflows

---

## SUMMARY

SQL Server 7.0 and 2000 provide a number of functions that enable database queries to generate text messages. In some cases, the functions create a text message and store it in a variable; in others, the functions directly display the message. Two vulnerabilities associated with these functions have been discovered.

The first vulnerability results because of a flaw in the functions themselves. Several of the functions do not adequately verify that the requested text will fit into the buffer that's supplied to hold it. A buffer overrun could occur as a result, and could be used either to run code in the security context of the SQL Server service or to cause the SQL Server service to fail. SQL Server can be configured to run in various security contexts, and by default runs as a domain user. The precise privileges the attacker could gain would depend on the specific security context that the service runs in.

The second vulnerability results because of a format string vulnerability in the C runtime functions that the SQL Server functions call when installed on Windows NT 4.0, Windows 2000, or Windows XP. Although format string vulnerabilities often can be exploited to run code of the

## Securiteam: [NT] SQL Server Text Formatting Functions Suffer from Buffer Overflows

attacker's choice, that is not true in this case. Because of the specific way this vulnerability occurs, the C Runtime code would always be overrun with the same values regardless of the attacker's inputs. As a result, this vulnerability could only be used as a denial of service.

An attacker could exploit the vulnerabilities in either of two ways. The most direct way would be for the attacker to simply load and execute a database query that calls one of the affected functions. Alternatively, if a web site or other database front-end would accept and process arbitrary queries, it could be possible for the attacker to provide inputs that would cause the query to call an affected function with the appropriate parameters.

Because the two vulnerabilities have different root causes, there are separate patches for each. Microsoft recommends that the SQL Server patch be applied to all affected servers. However, we recommend that customers carefully weigh whether they need to apply the C runtime patch. We make this recommendation for two reasons:

- \* The C runtime vulnerability only allows denial of service attacks, so the threat it poses is somewhat lower.

- \* The C runtime plays a crucial role in the operating system itself.

While we are confident that both patches are well tested, if there were a regression error in the C runtime, the effects would likely be serious and widespread.

### DETAILS

Mitigating factors:

- \* The effect of exploiting the first vulnerability would depend on how the SQL Server service was configured. SQL Server can be configured to run in a security context of the administrator's choosing (by default, it runs as a domain user). If best practices are followed, and the service is configured to run with the least privileges necessary, it would limit the worst-case damage an attacker could achieve.

- \* The second vulnerability could only be used for denial of service attacks. It could not be used to run code on the machine.

- \* The second vulnerability could only be exploited against SQL Server when running on Windows NT 4.0, Windows 2000 or Windows XP.

- \* Both vectors for exploiting the vulnerabilities could be blocked by following best practices. Specifically, untrusted users should not be allowed to load and execute queries of their choice on a database server, and publicly accessible database queries should thoroughly filter all inputs prior to using them.

Patch availability:

Download locations for this patch

- \* SQL Server:

- \* SQL Server 7.0:

<http://www.microsoft.com/Downloads/Release.asp?ReleaseID=35066>  
<http://www.microsoft.com/Downloads/Release.asp?ReleaseID=35066>

- \* SQL Server 2000:

## Securiteam: [NT] SQL Server Text Formatting Functions Suffer from Buffer Overflows

<<http://www.microsoft.com/Downloads/Release.asp?ReleaseID=35067>>  
<http://www.microsoft.com/Downloads/Release.asp?ReleaseID=35067>

C Runtime:

\* Windows NT 4.0 and Windows 2000:

<<http://www.microsoft.com/Downloads/Release.asp?ReleaseID=33500>>  
<http://www.microsoft.com/Downloads/Release.asp?ReleaseID=33500>

\* Windows XP:

<<http://www.microsoft.com/Downloads/Release.asp?ReleaseID=35023>>  
<http://www.microsoft.com/Downloads/Release.asp?ReleaseID=35023>

What are the vulnerabilities discussed in this bulletin?

This bulletin discusses an attack scenario against SQL Server that actually results from two completely unrelated vulnerabilities:

- \* The first vulnerability results from a flaw in SQL Server itself.
- \* The second vulnerability results from a flaw in the C runtime in Windows NT 4.0, Windows 2000, and Windows XP that is exposed primarily through SQL Server.

What is the scope of the first vulnerability?

The first vulnerability is a buffer-overflow vulnerability. An attacker who successfully exploited the vulnerability would gain significant control over the database, and perhaps over the server itself. In the worst case, this could enable the attacker to add, delete, or change data in the database, as well as potentially allowing the attacker to reconfigure the operating system, install new software on it, or simply reformat the hard drive.

The scope of the vulnerability would be significantly reduced if best practices were followed. In particular:

- \* SQL Server can be configured to run with less than total privileges on the server. If this were done, it would also have the effect of limiting what an attacker could do via this vulnerability.
- \* In order to exploit the vulnerability, the attacker would need the ability to load and run a database query (which best practices recommend against), or would need to locate a query that did not correctly check all inputs before using them.

What causes the vulnerability?

The vulnerability results because several functions provide by SQL Server contain unchecked buffers. By calling any of these functions with specially chosen parameters, an attacker could cause a buffer-overflow condition to occur.

What are the affected functions?

The functions that contain the unchecked buffers are associated with creating, and in some cases displaying, text messages. SQL Server provides several functions that enable a database query to create formatted text. In some cases, the text is simply stored in a variable in order to be printed later; in others, the text is immediately displayed in response to

an error condition.

What is wrong with the functions?

The functions implicated in the vulnerability all have the same defect: they do not check that the text the query has requested will actually fit into the buffer that has been provided. Because of this, an attacker could provide text that overruns the buffer and overwrites memory within the SQL Server process itself.

What would this enable an attacker to do?

Depending on the specific text the attacker chose, either of two effects could occur:

- \* If the text were random data, the SQL Server process would fail.
- \* If the text were carefully selected, it could be possible for the attacker to alter the SQL Server software while it was running.

If the attacker provided random data as the text, what would be required in order to restore normal operation?

The administrator would need to restart the SQL Server service.

If the attacker provided carefully selected data and altered the SQL Server software, what could the new software do?

It would depend on the privileges the SQL Server service had to begin with. By default, SQL Server normally runs with unexceptional privileges (specifically, those of a domain user). An attacker who successfully exploited this vulnerability against such a server would gain control over the database, but little else.

On the other hand, if the administrator had configured SQL Server to run with higher privileges, a successful attacker could potentially gain additional privileges. This is a good example of the importance of adhering to the principal of least privilege.

How might an attacker exploit this vulnerability?

There are several ways an attacker might exploit the vulnerability. The most direct one would be for the attacker to construct a query that deliberately calls one of the affected functions and overruns its buffer. However, it would only be possible for the attacker to do this if the server had been configured to allow untrusted users to load and run queries of their choice. Best practices recommend that only trusted users be allowed to do this.

What if the attacker could not load and execute a query?

It might still be possible to exploit the vulnerability. Suppose, for instance, that a web site provided a service that involved searching a database. If the database search called one of the affected functions, it might be possible for an attacker to construct a query that would cause the function to be called in such a way as to exploit the vulnerability. Clearly, this would require intimate knowledge of the internals of the database query.

## Securiteam: [NT] SQL Server Text Formatting Functions Suffer from Buffer Overflows

On the other hand, if the database query were poorly implemented, the attacker might not need to know its internal details in order to exploit the vulnerability. One frequently exploited implementation error occurs when a database query does not adequately check its inputs before using them. In some cases, this would make it possible for an attacker to embed database commands – including, in this case, calls to the affected functions – within database query parameters. This underscores the need to always check input parameters before using them.

What does the patch do?

The patch eliminates the vulnerability by implanting proper checking into the affected functions.

What is the scope of the second vulnerability?

The second vulnerability is a denial of service vulnerability. Although the attack scenario for this vulnerability is the same as for the first vulnerability, it could only be utilized to cause the SQL Server service to fail, and only when running on Windows NT 4.0, Windows 2000, or Windows XP. The administrator could immediately restore service by restarting the SQL Server service.

As in the case of the first vulnerability, exploiting this vulnerability could be difficult. The attacker either would need the ability to load and run a database query on a SQL Server, or would need to locate an existing query – one that did not check its inputs — on the machine. As discussed in detail below, we recommend that the patch for this vulnerability only be applied to systems that the administrator judges to be at very high risk.

What causes the vulnerability?

This vulnerability occurs because the C runtime in Windows NT 4.0, Windows 2000, and Windows XP are subject to a format string vulnerability.

How is this vulnerability related to the first vulnerability?

In terms of the implementation error, this vulnerability is completely unrelated to the first one. That vulnerability results because of an implementation error in SQL Server; this one results because of a flaw in the C runtime. However, in terms of the attack scenario, they are identical. Both could be exploited through SQL Server, in exactly the same way. In fact, from an attacker's perspective, it would appear that there was only a single vulnerability.

What is the C runtime?

The C runtime is the set of executables and files that provide support for programs written in the C programming language. All Windows platforms ship with a runtime for C, as well as other languages. The problem results because of a format string vulnerability affecting one of the functions in the C runtime that ships with Windows NT 4.0, Windows 2000, and Windows XP.

## Securiteam: [NT] SQL Server Text Formatting Functions Suffer from Buffer Overflows

What is a format string vulnerability?

A format string vulnerability occurs when a function that accepts formatted text for printing does not properly validate it before using it. In some cases, it is possible to either overwrite program code or take other action via the formatting codes.

If it is possible to overwrite program code via a format string vulnerability, why do you say this could only be used for denial of service attacks?

It is true that format string vulnerabilities can, in many cases, be exploited to run programs of an attacker's choice. However, this is not universally true. In the case of this vulnerability, program code can indeed be overwritten, but with the same values every time, regardless of the data that was provided to the function. The values do not lend themselves to running code, and the result is that the worst that could happen is that the SQL Server service could be made to fail.

If this vulnerability involves the C runtime, why are you discussing it solely in terms of SQL Server? Couldn't someone just write a C program and use it to exploit the vulnerability?

Yes. However, if an attacker wrote a C program to exploit this vulnerability, the only effect would be to cause the program to fail. It couldn't be used for any broader effect. It's only when the vulnerability is exposed through SQL Server that it becomes a concern, and even then only because it provides a way to cause SQL Server to fail.

How could an attacker exploit this vulnerability?

The exploit scenario for this vulnerability is indistinguishable from that of the first vulnerability. The attacker would need to create a SQL Server query that requested a particular text message, or locate an existing query that would let him give it arbitrary SQL commands.

Could the attacker tell which vulnerability he was exploiting?

There are no external indications that would allow the attacker to identify beforehand whether he was exploiting this vulnerability or the one discussed above. Clearly, though, the attacker could judge which vulnerability had been exploited by observing the effect, and seeing whether it was possible to make code run.

You've advised that the patch for the first vulnerability be applied to every system running SQL Server, but only advised that administrators consider applying this patch. Why?

First, the scope of the first vulnerability is much more serious than this one. However, there's a larger answer. The C runtime is fundamental to many operating system functions, including the ability to boot the system at all. Because of this, we believe the threshold for applying the patch should be higher.

Are you saying you don't trust the patch?

Securiteam: [NT] SQL Server Text Formatting Functions Suffer from Buffer Overflows

No. The fix is well understood and has been thoroughly tested. Just the same, when building a security patch, timeliness must be our first concern. As a result, we cannot perform the same level of testing as would be performed for a service pack or a new product version. We are confident in the patch, but do believe that it's prudent to only apply it to servers that are truly at risk. For all other cases, we recommend waiting until the next service pack, which will contain the fix.

My company writes applications using C, and we redistribute the C runtime with our applications. Can we redistribute this patch as part of the runtime?

No.

What does the patch do?

The patch eliminates the vulnerability by instituting proper validation of the formatted string printing functions in the C runtime.

ADDITIONAL INFORMATION

The information has been provided by <mailto:[secnotif@MICROSOFT.COM](mailto:secnotif@MICROSOFT.COM)>  
Microsoft Product Security.

=====

This bulletin is sent to members of the SecuriTeam mailing list.

To unsubscribe from the list, send mail with an empty subject line and body to:

[list-unsubscribe@securiteam.com](mailto:list-unsubscribe@securiteam.com)

In order to subscribe to the mailing list, simply forward this email to: [list-subscribe@securiteam.com](mailto:list-subscribe@securiteam.com)

=====  
=====

DISCLAIMER:

The information in this bulletin is provided "AS IS" without warranty of any kind.

In no event shall we be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages.

- 
- **Previous message:** [support@securiteam.com](mailto:support@securiteam.com): "[NT] PGP Plugin for Outlook Can Send Unencrypted Messages"
  - **Messages sorted by:** [\[ date \]](#) [\[ thread \]](#) [\[ subject \]](#) [\[ author \]](#) [\[ attachment \]](#)