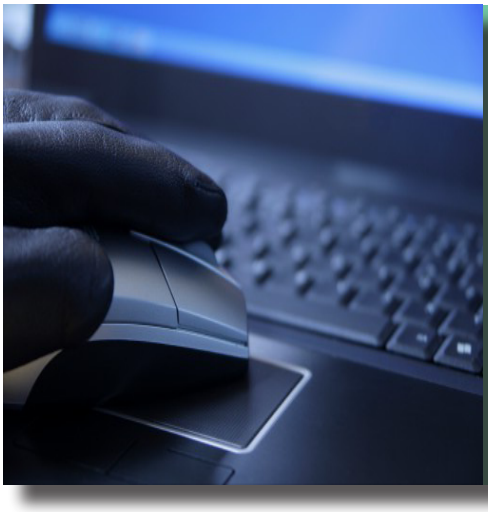
BREACH



AN ANATOMY OF A WEB HACK: SQL INJECTION EXPLAINED

BY DAVID STROM
FOUNDING EDITOR-IN-CHIEF
NETWORK COMPUTING MAGAZINE

JANUARY 2009

Breach Security, Inc. | Corporate Headquarters | 2141 Palomar Airport Road, #200 | Carlsbad, CA 92011 | USA
tel: (760) 268-1924 | toll-free: (866) 205-7032 | fax: (760) 454-1746 | www.breach.com

INTRODUCTION

Allow me to show you how to hack into your own website. You don't need any specialized tools other than a web browser and you don't need any specialized skills either. It doesn't take much time, and the payoffs could be huge: an intruder could easily obtain a copy of your most sensitive data in about the time it takes to read through this analysis.

"...an intruder could easily obtain a copy of your most sensitive data in about the time it takes to read through this analysis."

While there are many exploits, none are as simple or as potentially destructive as a SQL injection. This isn't something new, but what is new is how frequent this attack happens and how easily you can protect your network with relatively little effort and cost.

Let's walk through what is involved with a SQL injection exploit, using examples of both a website that we found at random as well as one that had previously been compromised with the hackers publicly describing their methods over the Internet. We will show you the consequences of doing nothing and leaving this front door wide open for anyone to walk into your data center. Finally, we will talk about ways that you can prevent this from happening in the future and what choices you have to protect your websites and networks.

TODAY'S THREAT LANDSCAPE

It used to be most attacks happened through email or remote penetrations into the network. With the improvements in anti-virus protection and greater deployment of firewalls, today's threat landscape is a very different one. In a word, it is all about applications and, in particular, web-based applications that tie into traditional databases and interact with browsers across the Internet.

"Web applications are inherently vulnerable."

Clearly, websites are easy targets. They are exposed to the general Internet, indexed efficiently by Google™ and other search engines to show vulnerable areas and often overlooked by security specialists. Most intrusion prevention and detection systems allow all web traffic free reign on organizational networks, so that any exploit involving a web server and that originates from a random browser are harder to detect and block. Also, some exploits, such as the SQL injection one that we'll show you shortly, don't involve much in the way of sending and receiving traffic across a network. This makes them harder still to detect and prevent.

HOW DATABASES AND WEB SERVERS WORK TOGETHER

Before we begin our tutorial on SQL injection, a few words about how web and database servers work together. Most databases can be thought of as a collection of tables of information: customer address records in one table and purchase records and inventories of products in two others. For a typical ecommerce customer, a browser is used to start the purchasing process by first finding some product of interest. This kicks off a query of the inventory to see if in stock, and then the web server produces a shopping cart page that shows what items are about to be purchased.

Because web servers are stateless, there has to be some way for the server to track what path a browser uses as the shopper moves from page to page on the site and queries these particular databases on their journey to check-out and purchase. Application developers employ a number of techniques to track user session state, including setting cookies on a user's machine, using

downloaded programs that save the browser state, using customer logins and passwords and storing the web queries in special tables of their own.

While we have used the example of an ecommerce site, the issue is more general. There are websites that don't sell anything, but allow visitors access to information, such as the IRS to download particular tax forms, the local community center to see what times it is open to the public or the local library to see if a book is on its shelves. In each case, the browser is requesting information from a particular database, and the web server is used to collect, organize and present the information. As Web 2.0 technologies such as AJAX become pervasive, there is more and more information that is presented to browsers dynamically and more information that originates from databases.

In order for all this to work, the request for information is translated from the web server into commands that the database server understands. Think of the web browser as an extended keyboard that is operated across the Internet, but directly connected to your database server in your data center. Indeed, that is a good picture to keep in mind, because by the time a SQL injection exploit is finished, it will seem as if the hacker is directly typing in commands to your servers.

The problem is that web developers tend to think that database queries are coming from a trusted source, namely the database server itself. But that isn't always the case, and a hacker or even a casual browser can often take control over the web server by entering commands that appear to be valid SQL commands in the right places. The trick is finding the right places.

There are two situations where the web and databases intersect that are relevant for our discussion on SQL injection:

- Places that directly enter database parameters into the URL itself, or
- Fill-in forms on web pages that will take this information and pass it along to the database server via the HTTP POST command.

Think about this for a moment. There are probably dozens, if not hundreds of places across your various websites that fit these two situations. Can you test them all to make sure your developers did everything possible to lock things down? What about a simple form that sends a password back to a particular email address in case a (what you might have thought legitimate) user forgot it? This is a simple and very effective way for a hacker to penetrate your defenses and assemble a very complete picture of your database structure, table names and field names.

Let's look at the first situation and give you more context. Before we do, we first need to talk about SQL command syntax. Every database server has a similar series of commands to query its tables, narrow down results to a few specific entries and combine information from one table to another. While a general tutorial on SQL database syntax is outside the scope of this paper, there are a few commands that are easily explained. Here is a typical SQL statement:

```
SELECT some-data FROM some table WHERE some-condition-is-met OR something-else-here
```

Here the `WHERE` specifies a condition, such as user name is equal to "John Jones" or an account matches a particular ID number. When this command is given to the database server, it returns the results of this query with all matching items that meet this condition. The `some-data` portion of the command provides the view of the results with particular database fields, such the "name, address, phone number" fields in a customer database. If you are thinking of tables, each row in the table is a different data item and each column in the table is the different field for each particular item.

Now what does this have to do with the web? When we put this data on the web, we use URLs and forms to assemble the query statements. Here is what a typical URL might look like on a website that has a catalog of things for sale, and just after a user has clicked on a particular item to get more information about it:

http://www.yourwebsite.com/askforparticularitem.asp?id=6969

We'll get to what happens when you make changes to the URL in a little bit but, for now, realize that the item or ID number in the URL gets taken by the web server and placed into a SQL query such as the following:

```
SELECT name, description, price FROM itemtable WHERE id=6969
```

The SQL server returns the name, description and price details as a result, and then passes this information back to the web server, where it is displayed on the page. The same thing happens with a web form: the parameter gets passed from the form to the database server, and then back to the web server for presentation and display.

ANATOMY OF A HACK

Let's assume that a bad guy wants to get into your databases and steal your customers. How does it work? Simple, they first find you on the Internet via Google or some other search engine.

Most database servers can be easily found on Google by searching for the right keywords. There are just a few typical search terms, such as `login.asp, asp?id=, php?id=` and other statements indicating database queries that are being passed from the web server to the database, including web forms as we mentioned earlier.

So we do a search on `gov: php?id=` and one of the results is this site, dealing with Scotland's biodiversity resources:

http://www.biodiversityscotland.gov.uk

Now let's browse around the site until we get to a page that has specific information—we will see a URL like something below:

http://www.biodiversityscotland.gov.uk/pageType2.php?id=8&type=2&navID=30

A hacker would use this information to begin his hack by changing the last part of the URL to the following:

http://www.biodiversityscotland.gov.uk/pageType2.php?id=999

And the page would be returned with a slew of error codes. Now, before you get frustrated because of the errors, realize that we are finished with our experiment. The goal with SQL injection is to get error pages, reading the codes like a fortune teller reads tea leaves or lines on your hand. These error codes provide a great deal of rich and important information about the internal structure of the database, the location of resources and even, in some cases, the names of the database fields or other content that can be used to further construct more compromising queries.

So what do we know from this very simple experiment? First, that the site is running a program called SiteMark and also running MySQL as its database server. Second, we have the directory where the PHP scripts are located. And even the lines of the program that are generating the errors are shown for us here. All we did was send a single page request to get all this information! We didn't have to learn any new skills, download any hacker tools or do anything other than type in a single line of text.

Now, admittedly this isn't a site that is going to attract a significant number of criminals, looking to crack into details about Scotland's rich ecological history. But it shows how easily a hacker can discern internal details about a web application simply by playing around with a search engine.

MORE SERIOUS STUFF

Let's take things one step further and demonstrate how databases are actually compromised. We'll use for our guide a Russian web site that posted hacking instructions on how to compromise a U.S. state government website:

http://www.xakep.ru/post/29550/default.asp

While the site is in Russian, you can translate it automatically using Babel Fish® or some other service to get the drift. What you see here is a step-by-step series of instructions on how RI.GOV was penetrated. Of course, since this hacking manual was published, the exploit was removed, but this is a very real demonstration of an actual exploit. The Rhode Island government site contains detailed taxpayer information on property records and other information that could be used by cyber-criminals to perpetrate identity theft.

There are two key elements to the Rhode Island attack that are explained in Russian. First, is the way that a hacker adds text to a normal SQL query statement using various characters and commands to trick the database server into delivering actual results, rather than just error messages.

Depending on the actual SQL query, you can add database commands to the end of the URL or inside an appropriate web form field:

```
'
or 1=1--
" or 1=1--
or 1=1--
' or 'a'='a
" or "a"="a
') or ('a'='a
```

Each line of text typically starts with a single quote, includes an `OR` statement and then a statement that is always true (1=1). Taken all together, what we are doing here by appending this text is telling the database server to deliver everything, since we are asking for either a single record or else "1=1"—meaning, give us all records.

The second element to the Russian instructions are how to take the feedback and error codes generated by the Rhode Island site and use it to further investigate the data structures of the site. The instructions show you how the hacker was able to obtain a series of passwords, pass them through a password cracking utility and gain total access to the database. Subsequent screen images show the hacker using a secure telnet login and gaining access to the server and then to the data itself.

RECAPPING WHAT WE HAVE LEARNED SO FAR

So what have we learned? All SQL injection attacks begin by modifying or adding text to a vulnerable area on a website: either a URL with an embedded hard-coded query statement or inside an otherwise innocuous web form. Information is then passed to the database server and acted upon.

There are two different attack strategies: manipulation of the SQL query string and compromised logins, and both were shown in the Russian instructions. The most well known attack is to modify the `WHERE` clause of the user authentication statement so the `WHERE` clause always results in TRUE. A second method is by adding information to the string and therefore a series of new commands, such as by using an `Execute` command in Microsoft SQL Server.

Through a series of attempts, a hacker can gain control over the server with a series of educated guesses and using well-known techniques to send information to the database server and monitor the replies. Once a bad guy has this information, s/he can begin to manipulate your authentication defenses and, at worst, gain total control over your data. But even if this doesn't happen, at the very least, a hacker can find out all sorts of things about your network and database structure.

DIFFERENCES IN DATABASE SERVERS

The various database servers have different features when it comes to two critical areas: the ability to send multiple commands together and the ability to send an executable command. Taken together, this makes it easier for hackers to manipulate query strings and compromise these servers.

PRODUCT	MULTIPLE	EXECUTE
ORACLE®	NO	NO
MICROSOFT® SQL SERVER	YES	YES
POSTGRE SQL	YES	NO

As you can see, Microsoft's SQL Server is vulnerable in both areas.

POTENTIAL PROTECTIVE MEASURES

So what can you do to protect your data? Well, short of disconnecting your corporate network from the Internet, there are two basic strategies you can follow.

1. **First, lock down your database servers to remove any potential backdoors.**
   You should do this in any event, as just basic best security practices. While SQL injection comes in through the front door, you should still practice safe computing and treat your database servers as yet another operating system that requires hardening, similar to what you do with your other servers.

2. **Do a better QA job and make sure you validate your input scripts and strings that are sent to the database.**
   It goes almost without saying that developers can't test everything. But you can do a better job of looking at all of your web scripts and programs and testing them for exploits. Do you have range limits for all queries? Do you have the appropriate access rights for all users, including the default rights for web users? Do you apply input validation checks for all places that accept input from the web?

   One common mistake is not checking stored procedures, thinking they are outside the bounds of SQL injection. Not true. Wherever possible, you should restrict the access of web-based applications to stored procedures and make sure you filter any data that is passed to them to eliminate potential exploits.

   The key takeaway here: it is best to validate all inputs that come from the web, keeping in mind what SQL injection can do.

BETTER PROTECTIVE MEASURES: WEBDEFEND® FROM BREACH SECURITY™

The problem with all of the above protective measures is that they only work for the moment in time that you implement them. As soon as a developer creates a new web application, you are once again vulnerable. They also are a lot of work to maintain and to enforce, and will require a fair amount of skill to implement properly. These measures also assume that your original web and database developers are still around, still remember what they did and can find their way through their code to make the necessary protective measures. That is oftentimes a tall order, especially given the rate of change in websites these days.

There is a better way, and that is why we recommend Breach Security's WebDefend web application security appliance. It is a very cost-effective solution, takes a few minutes to set up, doesn't require any special skills to maintain and will keep your data safe and prevent any SQL injection exploit from ever happening, no matter what applications you create down the road.

WebDefend can complete the application development lifecycle by providing real-time, continuous web application protection for production applications with passive vulnerability assessment. WebDefend can block attacks from hackers before they are able to gather enough information to launch a successful targeted attack. WebDefend is able to identify and correlate the events that show an attacker is researching the site, giving organizations the power to see and block sophisticated targeted attacks on the application.

WebDefend protects web applications from SQL injection attacks by performing validation on all user input to the application. Each input field or query parameter within the application is identified, typed and specified in the security profile. While validating traffic against an application's security profile, WebDefend will check all user input to ensure that it is the correct data type and has the

appropriate data length. Also, it will check to see that the data does not include any special characters or SQL commands. WebDefend will prevent any SQL injection attacks against a web application by ensuring that user input is only data with no attempts to circumvent an application's normal behavior.

WebDefend is the best solution to protect high-value web applications and the data behind them from targeted web-based attacks. WebDefend provides comprehensive web application protection through an architecture designed to address the spectrum of modern web application threats. Automated, behavior-based security profiles are created and maintained for each web application, ensuring that unique application vulnerabilities are successfully addressed. This positive security model ensures that only acceptable behaviors are allowed, thereby protecting against even unknown threats to the application.

SUMMARY

So we have shown you how easy it is to penetrate the average website and gain information about the underlying database structures inside a network firewall. The notion of SQL injection isn't new, but is still widely misunderstood and many sites are still vulnerable to attack. By using a security appliance such as WebDefend, along with beefing up security practices for web-accessible data, you can fill these holes in your network security as well as provide for tailored web application-specific security and comprehensive protection against the array of potential web-based threats.

ABOUT THE AUTHOR

David Strom is a freelance writer and consultant who has written two books and thousands of articles on networking, security and computing topics. He was founding editor-in-chief for *Network Computing* magazine and has created and run the editorial operations of dozens of web magazines over his career. He has worked in IT departments of private industry and government, and can be reached at david@strom.com.

ABOUT BREACH SECURITY

Breach Security, Inc. is the leading provider of real-time, continuous web application integrity, security and compliance that protects sensitive web-based information. Breach Security's products protect web applications from hacking attacks and data leakage and ensure applications operate as intended. The company's products are trusted by thousands of organizations around the world, including leaders in finance, healthcare, ecommerce, travel and government.

For more information, contact an authorized Breach Security representative or visit www.breach.com.