Dealing with Database Denial of Service

Version 1.0
Released: August 22, 2013

Securosis, L.L.C. 515 E. Carefree Blvd. Suite #766 Phoenix, AZ 85085 T 602-412-3051 info@securosis.com www.securosis.com

Author's Note

The content in this report was developed independently of any sponsors. It is based on material originally posted on the Securosis blog but has been enhanced, reviewed, and professionally edited.

Special thanks to Chris Pepper for editing and content support.

Licensed by DB Networks

DB Networks is innovating complex behavioral analysis technology for database security. Developed for organizations that need to protect their data from advanced attacks, DB Networks offers effective countermeasures against SQL Injection and database Denial of Service attacks. Database attacks happen rapidly - in a matter of minutes - and bypass traditional security measures. DB Networks unique approach uses behavioral analysis technology to automatically learn each application's proper SQL statement behavior. Any SQL statements that deviate from the established behavioral model raise an alarm as an attack. This approach has proven highly accurate and able to immediately identify extremely advanced database attacks.

For more information, visit: DB Networks

Copyright

Table of Contents

Introduction 2

Threat Landscape 3

The Attacks 5

- Abuse of Functions 5
- Complex Queries 6
- Bugs and Defects 6
- Application Usage 7

Countermeasures 9

About the Author 12

About Securosis 13

Introduction

There is an ongoing shift in Denial of Service (DoS) tactics by attackers, moving up the stack from networks to servers, and from servers into the application layer. As a result of concerted efforts to improve network defenses and harden servers, attackers are shifting their focus to easier targets: databases. Consistent with that trend we have both a new wave of reported database vulnerabilities, and isolated attacks, all linked to denial of service. The vast majority of web applications are supported by databases, and while protections for SQL injection at the application and network layer are common, database hardening to fend of DoS attacks is not.

It may come as a surprise, but database denial of service attacks have been common over the last decade. We don't hear much about them because they are lost among the din of SQL injection (SQLi) attacks, which cause more damage and offer attackers a wider range of destructive options. All things being equal, attackers generally prefer SQLi attacks because they can do just about anything to a database once they have taken control. But these attacks are *much* more difficult to exploit, and if the attacker just wants to disrupt operations, database DoS (Db-DoS) is a relative greenfield of opportunity. And with the growing trend of DoS attacks, databases are in the crosshairs of attackers.

Database DoS attacks may not permanently damage the database or expose sensitive data, but they can certainly crash the database — and and usually the web application it serves as well. Service interruption is no longer a trivial matter. Ten years ago it was still common practice to take a database or application off the Internet for the duration of an attack. But now web services and their databases are critical business infrastructure, and expected to operate round the clock. Take down a database and a company loses money — probably a lot of money.

This paper examines the trend in database denial of service, looking at how attackers target databases and the types of exploits they leverage. We will go into detail about the different classes of threats relevant to all forms of relational databases, why they exist, and how they are exploited. Finally, we will discuss effective methods for addressing these attacks, offering recommendations for how to harden and protect databases.

Threat Landscape

You don't need to be a database geek to know about Db-DoS — you just need to look at the news. We have seen recent Oracle issues with invalid object pointers, a serious vulnerability in the workload manager and the TNS listener barfing on malformed packets, and multiple vulnerabilities in MySQL, including a remote capability to crash the database. The is not unique to Oracle's products — we have seen a PostgreSQL issue with unrestricted networking access that was rumored to allow file corruption which crashed the database, and the IBM DB2 XML feature which allowed remote attackers to crash the database. Of course the presence of a vulnerability does not mean exploitation has occurred or will, but during our research we have heard many off-the-record accounts of database attacks. We cannot quantify the risk or likelihood of attack because there is too little public data to substantiate claims, but now is a good time to describe these attacks briefly and offer mitigation suggestions in response to this trend.

Now let's talk about approaches attackers take. In a recent research report on Denial of Service attacks, the most common DoS approaches we found were based on "flooding the pipes" rather than "exhausting the servers". Flooding is accomplished by sending so many network packets that they simply overwhelm the network equipment. This type of 'volumetric' attack is the classic denial of service, most commonly performed as a Distributed Denial of Service (DDoS) because it typically takes hundreds or thousands of malicious clients to flood a large network. Legitimate network traffic is washed away in the tide of junk, and users cannot reach servers.

Exhausting servers is different — these attacks target software running on the server, such as the operating system or web application components — to waste all its CPU, memory, or other resources and effectively disable it. These attacks can target either vulnerabilities or features of application stacks to overwhelm servers and prevent legitimate traffic from accessing web pages or completing transactions. The insidious aspect of this for attack is that, as you consume more hardware or software resources, platforms become less efficient. The closer to maximum utilization, the more servers slow down. Push them to the limit and they may simply lock up waiting for resources to become available. In some cases even removing the load does not bring servers back — you may need to reset or restart them.

Databases include their own networking features *and* offer a full complement of services, so **both** these models are effective against them.

The attacker motivation is very similar to other DoS attacks. Hacktivism is a major trend — taking down a major commercial web site is an option for many people who dislike a company but lack legal or financial means to express their complaints. "Covering attacks" are *very* common, where criminals flood servers and networks — including security systems — to mask ongoing attacks. Common scenarios include shutting down competitors, criminal racketeers threatening DoS and demanding ransom, financial trading manipulation, and the list goes on. The motivations behind Db-DoS are essentially the same. Recent tactics have evolved in response to a couple new factors. Network and server defenses are getting better with the current generation of firewall technologies, and it has gotten nearly impossible to flood the pipes of cloud services providers — to an attacker their network resources can be effectively limitless, redundant, and geographically dispersed. So attackers looked for new ways to keep old crimes profitable.

But attackers are not discriminatory — they are happy to exploit any hardware or software that enables them to accomplish their attacks, including web applications and databases. Database denial of service is conceptually no different than older DoS attacks at the sever or application layers, but there are *many* more clever ways to conduct a denial of service attack against a database. Unlike DDoS, you don't need to throw everything including the kitchen sink at a site — often you just need a small logic flaw in a database function to push it over. Relational database platforms are some of the most complex application platforms in existence, so there is plenty room for mischief.

Attackers sometimes morph traditional protocol and server based denial of service attacks to move up the stack. But more often they exploit specific database features in novel ways to take down their targets. Current DOS defenses are geared toward blocking network flooding and server attacks, so attackers have shifted targets to the application layer to better blend their incursions with legitimate customer transactions. Protection resources are generally focused on the lower layers, with relatively little attention paid to the application layer and virtually no time spent on defending against database attacks. Even worse, application layer attacks are *much* more difficult to detect because they tend to look like *legitimate* database requests.

The Attacks

As a security researcher, I cannot help but be impressed by the diversity of Db-DoS attacks. Many of them are *dumb* — they seem to be written by people who do not understand SQL, writing horrible and absurdly inefficient queries. Some exploits are so simple and clever that we are amazed the vulnerabilities they target were not found in quality assurance tests. Whether they are in fact clever or dumb, these attacks are effective. For example you could set up a couple different searches on a website, choose very broad lists of values, and hit 'search'. The backend relational system starts to look at every record in every table, chewing up memory and waiting on slow disk reads, starving out legitimate user requests. It's as simple as that.

Let's look more closely at a couple different classes of denial of service attacks:

Abuse of Functions

The abuse of database functions is, by my count of reported vulnerabilities related to DoS, the single most common type of Db-DoS attack. There have been hundreds, and it seems like no externally accessible function is safe.

> An attacker may use database features against you just like a judo expert uses your weight against you.

This class of attack is a bit like competitive judo: as you shift your weight in one direction your opponent pushes you in the same direction, to make you lose your balance and fall over. An attacker may use database features against you just like a judo expert uses your weight against you. For example, if you restrict failed logins, attackers may try bad passwords until they lock legitimate users out. If you implement services to automatically scale up to handle bursts of requests, attackers can generate bogus requests to scale the database up — until it collapses under its own weight, or in metered cloud environments you hit a billing threshold and service is shut down. There is no single attack vector, but a whole range of ways to misuse database functions.

This class of attacks is about attackers making a database function misbehave. Typically it occurs when a database command confuses the database, the query parser, or a sub-function enough to lock up or crash. Relational databases are complex gestalts of many interdependent processes, so the loss of a single service can cause the entire database to grind to a halt.

One example is an attacker sending malformed Remote Procedure Calls, incomprehensible to the parser, which cause it to simply stop. Malformed XML and TDS calls have been used the same way, as have SNMP queries. Pretty much every database communication protocol has, at one time or another, been fooled or subverted by requests that are *formatted correctly* but violate the expectations of the database developers in a way that causes a problem.

SQL injection is the *best known* type of functional abuse: SQL strings are bound into a variable passed to the database so it processes a very different query the application developers expected. SQLi is **not** typically associated with DoS. It is more often employed as a first step in a database takeover, because attackers generally want to control if they can. That said, SQL injection is also known for crashing databases, so it serves both nefarious use cases. But keep in mind that anti-SQLi products may not provide Db-DoS protection because SQLi <> Db-DoS!

Back to judo: remember that every function you have can be used against you.

Complex Queries

Complex queries work by giving the database too much work to do. Attackers find the most resource-intensive process accessible and kick off a few dozen requests. These attacks are designed to exhaust a database's resources, targeting memory consumption, processing power, or — in rare cases — I/O.

- **Computed columns and views:** Computed columns are virtual, typically created from the results of a query, and usually stored in memory. A view is a virtual table whose contents are also derived from a query. If the query selects a large amount of data, the results occupy a large chunk of memory. And if the column or view is based on a complex query it requires significant processing power to create. Exposure of computed columns and views has been the source of Db-DoS attacks in the past, with attackers continually refreshing views to slow down the database.

- **Nested queries & recursion:** Recursion is when a program calls itself; each time it recreates declared parameters or variables in memory. A common attack is to place a recursive call within a cursor `FOR` loop; after a few thousand iterations the database runs out of cursors or memory and halts.

- **The `IN` operator:** This operator tests whether a supplied variable matches any value within a set. The operation itself is very slow, even if the number of values to be compared is small. An attacker can inject the `IN` operator into a query to compare a large set of values against a variable that *never* matches. This is also called the **snowflake search** because it is like attempting to match two unique snowflakes, but the database continues to search regardless.

- **Cartesian products and joins:** The `JOIN` operation combines rows from two or more tables. A cartesian product is the sum of all rows from all tables specified in the `FROM` clause. Queries which calculate cartesian products on a few large tables generate *huge* result sets — possibly as large as the entire database. Operations on a cartesian product can overwhelm a database.

- **User defined functions:** Similar to computed columns and views, any user-defined function gives an attacker *carte blanche* to abuse the database with whatever query they choose. Attackers have leveraged all the above complex queries as user defined functions when allowed.

Attackers attempt to exploit any complex query they can access. All these abuses are predicated on the attacker being able to inject SQL into some part of the database or abuse legitimate instances of complex operations. A couple complex queries running in parallel are enough to consume the majority of a database platform's resources, badly slowing or completely stopping the database.

Bugs and Defects

Attackers exploit defects by targeting a specific weakness or bug in the database. In the last decade we have seen a steady parade of bugs that enable attackers — often remotely and without credentials — to knock over databases. A

single 'query of death' is often all it takes. Buffer overflows have long been the principal vulnerability exploited for Db-DoS. We have seen a few dozen buffer overflow attacks on Oracle that can take down a database — sometimes even without user credentials by leveraging the `PUBLIC` privilege. SQL Server has its own history of similar issues, including the named pipes vulnerability. Attackers have taken down DB2 by sending UDP packets. We hear rumors at present of a MySQL attack that slows databases to a crawl.

Attack specifics and targeted features vary, but the same basic attacks are used against new features every year. Database vendors constantly add new features and modules; as well as new ways to connect, communicate, and cooperate; and old attacks are revised to target new chinks in the armor. But all these exploits need a known defect, and the attacker needs to find vulnerable databases before they are patched. To succeed at remote exploitation the attacker needs to guess — or probe — the type of database in use behind a web service, then send an attack that slows the database down, exhaust its resources, or cause a deadlock.

Buffer overflows have long been the principal vulnerability exploited for Db-DoS.

Database developers and testers check to make sure new functions work, but they rarely try obscure test cases or fuzz inputs with random parameters. This is why historically so many bugs have led to Db-DoS — each time a vulnerability becomes public attackers 'weaponize' exploit code and then attack every web application they can find. And this is why DBAs are at odds with security practitioners — the typical database patch cycle is 14 months, while security patches from major relational database vendors are released every three months. Worse, attackers often release exploit code within days of vulnerability disclosure.

While companies spend far more on applications and application development that they do network infrastructure, the irony is security spending is the exact opposite. Security teams deploy the vast majority of their security controls at the network layer, which includes DDoS protections. There are many reasons for this, but for the most part it's faster and easier to insert an appliance on the wire than place security in the application layer. Negotiating with a DBA on how to implement and deploy security controls around a database is not fun, and finding solutions that are both effective and non-invasive is not an easy task. However, in order to address the attacks we discussed in this section it's essential. With attacks are moving 'up the stack', Db-DoS protection is simply not addressed by your traditional DDoS products.

Application Usage

If you thought the previous sections looked a bit like a list of database features, you are right. Denial of service may target anything — including the way the application uses the database. Databases **must** expose functions to application platforms to enable data management. But if the application does not successfully protect those functions — such as with SQL Injection — the underlying database is exposed to anyone who uses the app. Phrased another way: if an application can be leveraged to misuse a database, it will.

For example, one common attack against retail websites is to put a few thousand items in a shopping cart, continually adding new items and refreshing the cart. The database must validate each item is in stock, so it queries the inventory control database *for each item in the cart* on each refresh. An attacker may only need a couple concurrent shopping cart sessions to stall the site.

Attacks leverage legitimate application and database functions, focusing on those that require complex processes running behind the scenes. It does not take long for an attacker to identify good targets: operations that are already slow

to respond to input. Developers and DBAs tend to point fingers each another and deny their own responsibility, but there is no single way to address this type of attack.

Some of this discussion of Db-DoS over the past decade may look like ancient history. But while general DoS tactics have changed, the database specific attacks have not. DoS attacks in general are on the rise, and the most important change has been moving up the stack. Database and application layer attacks have been somewhat born again, as the same old avenues and exploits are leveraged all over again. As new hacking concepts and approaches evolve attackers look at the features and functions they have hacked previously to see how they can apply their new toys. We saw buffer injection attacks for over a decade, because they continued to work as attackers found new areas of vulnerable database code. We continue to see SQLi attacks because still they work. We see attacks against network protocols because DBAs still forget to configure them correctly.

Countermeasures

There is no single way to stop Db-DoS attacks. Every feature is a potential avenue for attack, so no single response can defend against everything, short of taking databases off the Internet entirely. Like database security, DoS protection does not happen by itself; it requires and investment in time and tool selection by you. The good news is that there are multiple countermeasures at your disposal, both detective and preventative, with many preventative security measures essentially free. All you need to do is put the time in to patch and configure your databases. But if your databases are high-profile targets you need preventative and detective controls for reasonable assurance that they won't be brought down.

The following approaches help mitigate Db-DoS attacks:

- **Configuration:** Reduce the attack surface of a database by removing what you don't need — you cannot exploit a feature that's not there. This means removing unneeded user accounts, communications protocols, services, and database features. A feature may have no known issues *today*, but that doesn't mean none are awaiting discovery. Relational databases are very mature platforms, packed full of features to accommodate various deployment models and uses for many different types of customers. If your company is normal you will never use half of them. But removal is not easy and takes some work on your part, to identify what you don't need and either alter database installation scripts or remove features after the fact.

- **Resource limits:** Several database platforms provide the capability to limit resources on a per-user basis (number of queries per minute, long running query timeouts, resource throttling for memory and processors, etc.). Configuring resource limits can help reduce the impact of attacks. Unfortunately in our experience these seldom stop DoS, although they can make it much more difficult. As with our judo comparison, attackers may use resource throttles to starve out legitimate users. Consider this a good option for graceful degradation and a good standard operating procedure, but your implementation needs to be very well planned to prevent limits from impinging on normal operation.

- **Patching:** Many DoS attacks exploit bugs in database code. Buffer overflows, mishandling of malformed network protocols or requests, memory leaks, and poorly designed multitasking have all been exploited. These are not the types of issues you or DBAs can address without vendor support. A small portion of these attacks are preventable with database activity monitoring and firewalls, as we will discuss below, but the only way to completely fix these issues is to apply vendor patches. And the vendor community, after a decade of public shaming by security researchers, has recently been fairly responsive in providing patches for serious security issues. The bad news is that enterprises patch databases every 14 months on average, choosing functional stability over security despite quarterly security patch releases. If you want to ensure bugs and defects don't provide an easy avenue for DoS, patch your databases promptly.

- **Database Activity Monitoring:** One of the most popular database protection technologies on the market, Database Activity Monitoring (DAM) alerts on database misuse. These platforms inspect incoming queries to see whether they violate policy. DAM has several methods for detecting bad queries, with examination of query metadata (user, time of day, table, schema, application, etc.) most common. Some DAM platforms offer behavioral monitoring by setting a

user behavior baseline to define 'normal' and alerting when users deviate. Many vendors offer SQL injection detection by inspecting the contents of the `WHERE` clause for known attack signatures. That said, DAM is typically deployed to enforce business logic rather than to prevent DoS. Many monitors offer an option block malicious queries, either through an agent or by signaling a reverse proxy on the network, but most DAM products are deployed in monitor-only mode, alerting when policy is violated. These platforms can be set up to help defend against DoS, but you need to configure them to block traffic and write policies to detect the attacks. Database monitoring is a popular choice as it combines a broad set of functions, including configuration analysis and other database security and compliance tools.

- **Database Firewalls:** We may think of `SELECT` as a simple statement, but some variations are not simple at all. Queries can become quite complex, enabling users to do all sorts of operations — including malicious actions which can confuse the database into performing undesired operations. Each SQL query (`SELECT`, `INSERT`, `UPDATE`, `CREATE`, etc.) has dozens of different options, allowing hundreds of variations. Combined with different variables in the `FROM` and `WHERE` clauses, they produce thousands of permutations — malicious queries can hide in this complexity. Database firewalls are used to block malicious queries by sitting between the application server and database. They work by understanding both legitimate query structures and which query structures the application is allowed to use. Database firewalls whitelist and blacklist queries for fine-grained filtering of incoming database requests, and block non-compliant queries, and like DAM, some behavioral monitoring. This shrinks the vast set of **possible** queries to a small handful of **allowed** queries. Database firewalls are designed to restrict which queries can be run, almost eliminating buffer overflow type attacks. Contrast this against the more common passive approach of database monitoring, alerting on internal user misuse, and *detection* of SQL injection. DAM is excellent with known attack signatures and suspect behavior patterns, but database firewalls reduce the threat surface for possible attacks by only allowing known query structures to reach the database, leaving a greatly reduced set of possible complex queries or defect exploitations.

- **Web Application Firewall:** Web application firewalls sit in front of the application to protect application functions from attack. They inspect requests for known malicious attack signatures, but they also provide rate-based DoS protection (against "flooding the pipes") for the app. Web application firewalls (WAF) protect the database indirectly, by blocking known SQL injection attacks; they also offer some capabilities for detecting database probing. While they offer some protection for databases, WAFs fail to address many database denial of service attacks described in this paper.

- **Application and Database Abstraction Layer Hardening:** Many Db-DoS attacks, as well as all SQL injection attacks, run through web-facing applications. If you are building an application from scratch you have the option of designing in database protections from the beginning. This is highly unusual — and we are only aware of a handful of companies which have taken this approach — but it can be very effective against SQL injection and buffer overflow attacks. You build checks into business logic to prevent misuse and cast user-supplied data into properly constrained variables to ensure users don't pass garbage to your database. Much like Hibernate abstracts database interfaces for portability, Database Abstraction Layers abstract database usage for security. Only the DAL needs detailed knowledge of the internal database structure, and it performs all queries on behalf of the application much like stored procedures. It translates generic application requests into database queries, **if** the request appears valid and all user data is correctly typed. As a secondary benefit using a DAL helps with portability. These preventative measures are most useful when **building** an application with time for Db-DoS threat modeling.

While each of these countermeasures is very effective in its own way, it requires a concerted effort with both detective and preventative measures to provide reasonable assurance that databases are hardened against attack. All these countermeasures are *supplements* to network and server DDoS protections to address database misuse. Database DOS is much less common than network flooding DDoS attacks. But with a database exploit in hand, a Db-DoS attack requires less effort and scale than a network DDoS attack. As network defenses get better and we address problems in the lower layers of the protocol stack, attackers reach up the stack for other avenues. Database down-time is no less intolerable than web site down-time, and must be accounted for when architecting systems to endure denial of service attacks.

We hope you find this paper useful. If you have any questions or want to discuss specifics of your situation, feel free to send us a note at info@securosis.com.

About the Author

Adrian Lane, Analyst and CTO

Adrian Lane is a Senior Security Strategist with 25 years of industry experience. He brings over a decade of C-level executive expertise to the Securosis team. Mr. Lane specializes in database architecture and data security. With extensive experience as a member of the vendor community (including positions at Ingres and Oracle), in addition to time as an IT customer in the CIO role, Adrian brings a business-oriented perspective to security implementations. Prior to joining Securosis, Adrian was CTO at database security firm IPLocks, Vice President of Engineering at Touchpoint, and CTO of the secure payment and digital rights management firm Transactor/Brodia. Adrian also blogs for Dark Reading and is a regular contributor to Information Security Magazine. Mr. Lane is a Computer Science graduate of the University of California at Berkeley with post-graduate work in operating systems at Stanford University.

About Securosis

Securosis, L.L.C. is an independent research and analysis firm dedicated to thought leadership, objectivity, and transparency. Our analysts have all held executive level positions and are dedicated to providing high-value, pragmatic advisory services.

Our services include:

- The Securosis Nexus: The Nexus is an online environment to help you get your job done better and faster. It provides pragmatic research on security topics, telling you exactly what you need to know, backed with industry-leading expert advice to answer your questions. The Nexus was designed to be fast and easy to use, and to get you the information you need as quickly as possible. Access it at <https://nexus.securosis.com/>.
- Primary research publishing: We currently release the vast majority of our research for free through our blog, and archive it in our Research Library. Most of these research documents can be sponsored for distribution on an annual basis. All published materials and presentations meet our strict objectivity requirements and conform with our Totally Transparent Research policy.
- Research products and strategic advisory services for end users: Securosis will be introducing a line of research products and inquiry-based subscription services designed to assist end user organizations in accelerating project and program success. Additional advisory projects are also available, including product selection assistance, technology and architecture strategy, education, security management evaluations, and risk assessment.
- Retainer services for vendors: Although we will accept briefings from anyone, some vendors opt for a tighter, ongoing relationship. We offer a number of flexible retainer packages. Services available as part of a retainer package include market and product analysis and strategy, technology guidance, product evaluation, and merger and acquisition assessment. We maintain our strict objectivity and confidentiality. More information on our retainer services (PDF) is available.
- External speaking and editorial: Securosis analysts frequently speak at industry events, give online presentations, and write and/or speak for a variety of publications and media.
- Other expert services: Securosis analysts are available for other services as well, including Strategic Advisory Days, Strategy Consulting Engagements, and Investor Services. These tend to be customized to meet a client's particular requirements.

Our clients range from stealth startups to some of the best known technology vendors and end users. Clients include large financial institutions, institutional investors, mid-sized enterprises, and major security vendors.

Additionally, Securosis partners with security testing labs to provide unique product evaluations that combine in-depth technical analysis with high-level product, architecture, and market analysis. For more information about Securosis, visit our website: <http://securosis.com/>.