Global Information Assurance Certification Paper

Making Your Network Safe for Databases

Duane Winner

July 21, 2002

GIAC Security Essentials Certification Practical Assignment -- version 1.4

Introduction

Databases are critical components of an information infrastructure today. It is doubtful that you will ever encounter a commercial web site that does not communicate with a database server behind the scenes. And it is almost certain that any e-commerce site or any other web site that collects information from visitors stores information in a back-end database server.

The problem with databases however, is that they may be easily overlooked when security procedures are implemented. Often, the focus is targeted towards securing the web server and little thought goes into how the database may be vulnerable because all of the 'front-end' security has been implemented. But databases usually contain a company's most valuable information assets, and if compromised, could wreak havoc. While much press is given to denial-of-service attacks and web-site defacements, attacks on database servers can result in even more severe consequences than just a public relations problem or lost revenue because of downtime. To illustrate this point, below are some examples of several high-profile organizations that have had their databases compromised over the past few years.

If you are charged with administering a network that contains a database server, there are a number of steps you can take to help protect the data from being compromised. Properly configured, you can help prevent your organization's information assets from falling into the wrong hands.

If it could happen to them…

Over the past few years, there have been many documented cases of organizations that have reported their databases compromised. In most cases, the theft of data has resulted in more than just embarrassment. At best, a company will lose time and money due to expenses occurred from conducting forensics to determine the extent of the theft. Companies have been blackmailed, or threatened with blackmail, and in some cases, have had to discontinue providing services to consumers due to lack of confidence in the system. Depending on the sensitivity of the data that is stolen, companies could face damaging lawsuits.

In December 2000, Egghead.com, a computer products retailer, announced that its customer database may have been compromised and that up to 3.7 million credit card accounts may have been stolen [1]. Egghead later claimed that its

customers' credit cards were not compromised, but the investigation to determine the extent of the breach cost millions [2].

Nearly a year previously, an intruder thought to be from Russia and known as "Maxus" attempted to blackmail cdUniverse, an online music retailer, after he exploited a security hole on their web site and stole credit card information from the database. He posted thousands of users' credit card details on the Internet after his extortion demands were refused [3]. To see a copy of the credit card black market website that Maxus ran, visit http://databases.about.com/gi/dynamic/offsite.htm?site=http://www.pc%2Dradio.com/maxus.htm.

In November 2001, Playboy.com was hacked, and intruders stole the credit card information of visitors, and proved it by sending threatening email messages to the customers that displayed all of their credit card information [4].

It was announced in the spring of 2001 that 98,000 people had their credit card information compromised when criminals broke into Bibliofind, division of Amazon.com that assisted users in locating rare and out-of-print books. To make matters worse, they managed to maintain their free access to the database for 4 months before being discovered [5]. As a result, Bibliofind ended up refraining from acting as a financial transaction service for its clients, and instead continued as a matching service only [6].

The FBI reported in March of 2001 that over 40 banking and retailer's web sites were attacked and compromised by Russian and Ukrainian hackers. Blackmail was usually involved after the theft of credit cards [7].

At two different times during 2001, Indiana University's computers were hacked and private individual information was stolen, including social security numbers and addresses [8].

James Middleton reported for vnunet.com in January of 2002 that Evans Data, a U.S. market research firm, conducted a study and found that 10 percent of databases had been compromised during 2001, based on 750 companies surveyed. Over forty percent of banking and financial services "reported incidences of unauthorized access and data corruption, while 18 per cent of medical/healthcare and telecoms firms reported similar breaches." [9]

All of these incidents illustrate that databases are treated as treasures for hackers to steal. While denial-of-service attacks and web site defacements can be very costly to a company that relies on e-commerce for revenue, these sorts of attacks provide very little financial incentive to hackers. Database theft, on the other hand, can be quite lucrative for an enterprising hacker who doesn't get caught. Once they steal the data, as illustrated in a few examples above, hackers may attempt to blackmail a company by threatening to post customers'

credit cards online.

But another even more disturbing development has begun to gain publicity, which is organized cyber-crime. There is a growing black market for stolen credit card numbers, just as there has been with physical cards. Only now hackers can sell their stolen 'goods' online and make a profit without having to physically snatch a purse or break into a glove box.

If you think these incidents are anomalies that gained a lot of publicity because the companies are high profile, just take a look at http://isc.incidents.org/port_details.html and enter the default port for whichever database you are running. The biggest target is Microsoft SQL Server. On some days there are over *half a million* reports of attacks upon servers running Microsoft's database products. Not even coming close are other databases, but there have been days when there have been over 600 reported attempts on MySQL databases. And if you punch in their respective port numbers on incidents.org, you will see that Sybase and Oracle are not exempt, either.

Following are some guidelines that should be followed if you are implementing a database-driven web site. They start with most simple and work their way up to the most paranoid. If you find yourself in a situation where you cannot implement every one of these guidelines, at least attempt to implement as much as possible. Most of these guidelines are relatively inexpensive and easy to implement, and should be seriously considered if your organization covets the security and confidentiality of it's data.

1: Give the database server and the web server their own hardware

One of the biggest mistakes that can be made when implementing a web site with a back-end database is to install the database server on the same box as the web server. While there may be a seemingly good argument for doing so, the risks should always outweigh the advantages. It may be done for convenience, lack of resources to procure a separate server for the database, or for performance reasons.

Whether it is full-fledged database application, like Microsoft SQL Server, Sybase or Oracle, or just a Microsoft Access database, if a hacker gains control of your web server, then they will also have access to your database resources. And a special note should be made regarding Microsoft Access. Many web sites use Microsoft Access databases. Considering the security flaws found in Microsoft's Internet Information Server (IIS) recently, and how worms like Nimda and Code Red could open a back door and provide administrator privileges and file system access to the entire server, special care should be given to using Microsoft Access, which is essentially a flat file, and could be easily stolen if somebody broke into the web server.

The bottom line is if a hacker breaks into the web server, then they will have access to your database. If the cost of purchasing a separate computer for your database outweighs the risks and consequences if your data is stolen, then by all means, go ahead and put the web server and the database server on the same box.

Some may also argue that desirable performance is obtained by co-locating the web server on the database server on the same computer. But this argument is doesn't hold much water when considering bandwidth models. Most internal network segments operate (at the very least) 10Mb per second; more commonly at 100Mb. Gigabit service is even available. So consider that while most Internet connections operate at less than 10Mb per second, your web server should have much more available bandwidth to perform fast queries. If there is to be a bottleneck, it is more likely to occur between the browser client and the web server rather than the web server and the database server [10].

2: Don't put the database server in the DMZ

Chances are that you have a demilitarized zone (DMZ) configured for your web server and any other resources that you need to make available for public access. It may seem logical at first to just install your database server in the DMZ and trust your firewall to prevent attacks against it. If you configure your firewall to only allow certain traffic to hosts that you want to be public, and drop any traffic directed towards the database server, then you may feel confident that your database is safe.

But are you positive that the database server is 100% safe from other servers in your DMZ? Just because the firewall is configured to drop packets directed towards the database server, the reality is that some forms of outside traffic are being allowed into your DMZ. Are you confident that your mail server is so secure that it could not be compromised and used as a launch pad to attack your database server? Depending on the size of your company, you may not have control over all of the servers in your DMZ, so you may be relying on basic trust that the administrators of the mail servers, the web servers, the DNS servers and any other servers in the DMZ have done their job to secure their boxes.

"But I'm using network address translation (NAT), and the database server doesn't even have a public IP address," you may say. It doesn't matter if one of the other servers behind your firewall is compromised. The hacker now has access to the same network with private IP addresses as your database server. This may seem, in essence, admitting that your own network is 'untrusted'. It is a matter of perspective. Some may argue that, in reality, any network is, to a certain degree, untrusted; i.e., there is always a certain degree of threat and vulnerability once you plug a computer into any network. No network or resource

is invulnerable.

So what are the options if you want to follow this suggestion, and remove your database server from the DMZ?

The first option would be to install a second firewall and configure it to protect a separate network that is separated from your DMZ. By configuring it to allow only very restricted traffic from privileged resources, the database server can be secured against attacks staged from a compromised server in the DMZ that should never have access to the database server in the first place (such as the DNS or mail server).

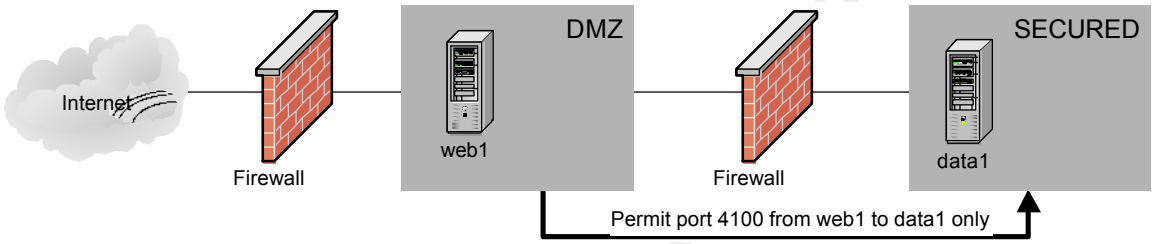

Figure 1

As shown in figure 1 above, the second firewall separates the database server from the rest of the DMZ and only allows specific traffic from the web server. In this case, the database server is running a Sybase server on Linux, so the firewall would be configured to allow only traffic from ‘web1’ to ‘data1’ over port 4100. If the firewall does its job correctly, if another server in the DMZ is compromised, an attacker will be prevented from launching an attack on ‘data1’ from that server.

Another recommendation that can be made if you follow this suggestion, is rather than install a second firewall that runs the same firewall software as the perimeter firewall, consider purchasing a different type of firewall than the one you already have protecting your DMZ. For instance, your first firewall may be a Check Point firewall. Instead of installing a second Check Point firewall to protect your private network, you may want to consider mixing it up a little and install a different firewall product, such as a Cisco PIX or a Raptor firewall. The reason for this is that by varying the types of protection in your network hierarchy, you will make it more difficult for a potential hacker to get all the way into your network. Suppose a hacker exploits an unknown (or unpatched) vulnerability in your Check Point firewall on the perimeter and gains access to some resource in your DMZ. If they attempt to proceed further, they will be surprised if they find that the next gatekeeper in line is not the same as what they already encountered on the outside.

There may be a reason that installing a second firewall would be impractical. One, you may simply not have the funds to do so. Another possibility may be that you don’t have the skills or resources to add additional technology to your

network. If you are a small or one-man operation, then it may not be advisable by complicating your network by adding additional technology. Firewall products take a certain level of knowledge about the products to properly configure and baby-sit. Having one or two improperly configured firewalls could be worse then not having a second firewall at all. You may also have so few resources in both the DMZ and the secured network that it would be difficult to justify purchasing a firewall just to separate one server from another.

If adding a second firewall is not a practical solution, then another option would be to add functionality to your perimeter firewall by adding another network interface. Many organizations already do this to separate their DMZ from their internal local area network/LAN (or corporate Intranet). The type of firewall you are using may limit this. A software-based firewall (such as Check Point) on an NT or Unix box is limited by the number of PCI slots available. A hardware-based solution, however, may be more limited (such as Check Point on a Nokia, or a Cisco PIX); in which case, your hardware profile (or licensing) may restrict the number of network interfaces.

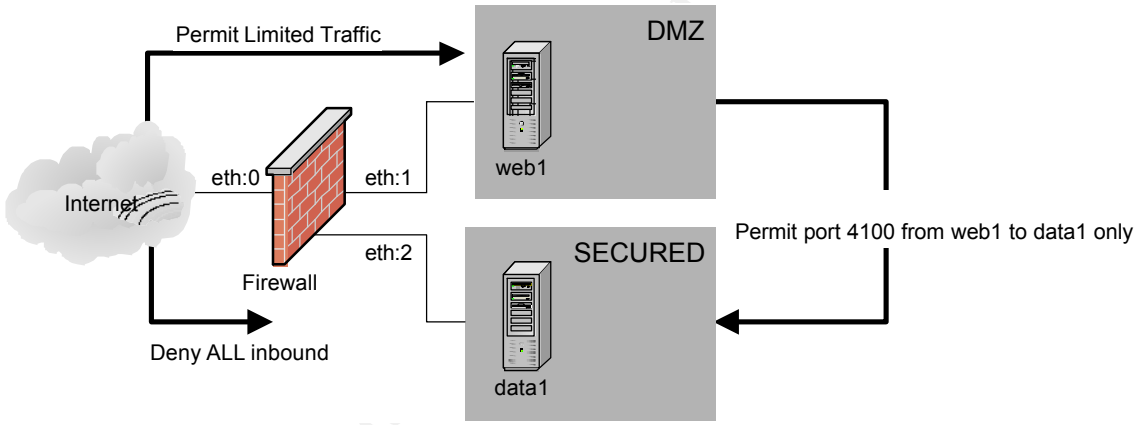

Figure 2

As illustrated in figure 2, the perimeter firewall has three network interfaces: eth:0, which connects to the Internet, eth:1, to which the DMZ is connected,, and eth:2, to which a second private network is connected (called ‘secured’ in this example).

The firewall is configured to limit outside traffic into the DMZ for public access, but absolutely no outside traffic is permitted into the SECURED network. But the firewall is configured so that port 4100 is allowed from ‘web1’ in the DMZ to ‘data1’ in the SECURED network, again using Sybase as the example.

A special note should be made regarding this technique, however: You may already have a similar configuration already set up for your existing DMZ and local area network (LAN). In that case, you should add a fourth network card to create the ‘SECURED’ network segment. But instead of adding a fourth network card, you may find it tempting to install your new database server (or even use

an existing corporate Intranet database) in your LAN, and permit restricted traffic to this database from the web server in the DMZ. This is *highly* discouraged. A corporate LAN should have outbound access only, and inbound access, whether from the Internet or the DMZ should *never* be allowed in. If the web server is compromised, then an attacker could conceivably gain access to your entire internal corporate network.

A third option, if neither of the above two options are feasible, is to leave the database server in the DMZ and rely on a local software-based firewall on the server. This is not the ideal solution, but is better than nothing. If you work for a small organization, and the database server is the only server you have that would benefit from creating a separate network segment, it may difficult to convince your boss to spend several thousand dollars for another firewall to protect one server.

If your database server is running on a Linux platform, then you could utilize ipchains or iptables (standard open-source software that comes with almost all Linux distributions). These are firewall software packages that perform the same duties as most commercial firewall products, and can be run locally to protect a single machine. In fact, their functionality is robust enough that they could also be utilized as a firewall gateway in option 1 above if cost is a concern, since both Linux and ipchains/iptables are free. An older unused Pentium computer (or possibly even a 486) with two network cards could be used as a low-cost second firewall gateway. The most significant difference between ipchains (the older) and iptables (the newer) is that ipchains performs packet-filtering only, while iptables performs stateful inspection, which is something that most newer commercial firewalls do, such as Check Point and Cisco PIX.

If your database server is running a Microsoft Windows platform, then you could install personal firewall software such as Black Ice Defender or Tiny Personal Firewall, both of which, although designed for workstations, may provide an adequate level of protection on your server. Windows 2000 also has native packet filtering rules that can be configured to protect the server.

3: Replace network hubs with switches

By using a switch instead of hub, you will greatly reduce the possibility of data being sniffed and captured as it traverses the network.

Hubs are simply connectors for all of the network media, and any node that is connected to a hub will have access to any data that is passed on that network. Because each host that is connected to a hub has equal access to the media and the information that is passed between any other hosts on the network, data could conceivably be ‘sniffed’ as it passes from one host to another.

Let's say that you have implemented suggestions 1 and 2 in this paper. You have provided the database server it's own hardware, separating it from the web server, and you have installed a second firewall and isolated the database server from the DMZ by locating it behind the new firewall. So far, so good, right?

But let's say that an intruder finds his way into your mail server in the DMZ and is undetected. He then sets the network adapter on the mail server into promiscuous mode and installs software that will capture all traffic that occurs in the DMZ. Even though the database server is not in the DMZ, the web server is passing traffic to and from the database server and the mail server is now capturing credit card information and relaying it back to the intruder's computer. *He does not have to break into either the web server or the database server to steal the data.*

Simply replacing your DMZ hub with a switch will go a long way to stymieing unauthorized sniffing on your network. The reason a switch is better is because when two hosts are communicating via a switch, a "virtual circuit" is created between the two hosts. Hosts connected to other ports on the switch do not see or even sense the traffic that is not directed towards them, making it nearly impossible to capture data.

Another benefit of using a switch is that you will obtain faster bandwidth between two hosts that are communicating.

Will this make your network completely safe at this point? Not quite. It may seem like a long shot, but a dedicated and determined hacker could still compromise your network if they can break into the switch. Most switches are configurable via a telnet console. Many switches can also be configured to forward all traffic to a single port for analysis. There are legitimate reasons for this; for instance, running intrusion detection software (IDS) in the DMZ is problematic unless you can forward all traffic to a single a port on the switch. However, if someone broke into your mail server, and from there also finds a way to access the switch, he/she could forward all traffic in the DMZ to the port that the mail server is connected. Now the mail server could capture traffic that moves between other hosts.

Granted, this would be difficult to accomplish, but the possibility still exists. So now what? How can you be absolutely certain that data traveling your network will not be captured? This brings us to the next recommendation, encryption.

4: Encrypt data between the web server and the database server

You may already be saying to yourself, "I'm already using SSL, so the data is secure."

First off, if you've read this far, then you probably already know that there are issues beyond normal SSL communication that need to be addressed. But this is actually a common attitude and misconception regarding Secure Sockets Layer. Many people think that by configuring a secure web site that utilizes SSL, the data is now encrypted, it is now safe, and their job is done as far as security is concerned. They often seem to overlook the fact that SSL-enabled web sites *only encrypt the data between the web browser and the web server.* If it is credit card transactions that are being made, the data will be encrypted as it travels the Internet, but is no longer encrypted once the web server passes it on to the database server, and vice versa; any queries the web server makes to the database server will be unencrypted [11].

So now what? How do you encrypt the data between the web server and the database server? The options are varied, and depend to a certain extent on the type of database server you have implemented and what version you are running.

If your database supports native encryption, then the web application could be written to talk to the database via SSL. For instance, Sybase has introduced SSL into its Adaptive Server Enterprise database as of version 12.5. The next release of MySQL (a stable version 4 still has not been released as of this writing) will incorporate SSL capability. PostgreSQL now has native SSL support as of version 7.2.1. Microsoft SQL Server 2000 supports SSL. Oracle also supports SSL encryption.

Using native SSL will require that whoever is writing the web application can properly utilize the SSL capability that the database provides and incorporate that into their code. If this is not feasible, or your database does not provide SSL capability, then the responsibility will fall on you as a network administrator to provide encryption between the two hosts.

There are two techniques that can be used to implement encryption between the web server and the database server. Both of these options are completely independent of the database and the operating system.

SSH Port forwarding

The first option is to use Secure Shell (SSH) port forwarding. SSH is a replacement for telnet that encrypts the sessions, but it can also be configured to listen for other types of traffic on the host, forward that traffic through the encrypted SSH session, and pass it back to its appropriate port on the other end. It is fairly easy to implement and has been ported to both Unix/Linux platforms as well as Microsoft Windows.

Initiating a normal SSH session is rather simple and if you were to open an SSH session from web1 to data1, the basic syntax would look something like this:

```
#ssh user@data1
```

This command will simply provide a shell console to data1 while logged on at web1. But there are additional switches that can be used with the above command to provide a 'tunnel' for other types of traffic. Following is the basic command that would be used to set up an SSH port-forwarding tunnel for Sybase traffic.

```
# ssh –L 4100:data1:4100 user@data1
```

Running the above command on web1 will open an SSH session to data1, but will also set up a listening port on the loopback interface on web1, and will listen for any traffic slated for port 4100. It will pass this traffic through the encrypted SSH session to data1, then the SSH daemon on data1 will decrypt the 4100 traffic and pass it on to the listening Sybase port.

Note that for the listening port on web1, you do not have to use port 4100. You make things a little more difficult for would-be intruders by using an unusual port number. For instance,

```
# ssh –L 35842:data1:4100 user@data1
```

This will instead listen on port 35842 on web1, but will still get translated to port 4100 for Sybase on data1. Your web application will simply need to know that any calls it makes to Sybase will be made to port 35842 instead of 4100.

The nice thing about SSH port forwarding, however, is that you can use it in either direction. The above examples use what is called "local forwarding," hence the '-L' switch in the command. You can also use "remote forwarding." Instead of opening the SSH tunnel from web1 to data1, you could initiate the tunnel from data1 to web1, telling web1 to listen for Sybase traffic and pass it back through the SSH tunnel to data1. The following command would accomplish this:

```
# ssh –R 4100:127.0.0.1:4100 web1
```

Using this method, you could configure the firewall that is protecting your secure network to deny ALL inbound traffic, permitting nothing inside, even from the DMZ, ensuring that no host outside of the secured network can initiate a connection to any resource inside the secured network.

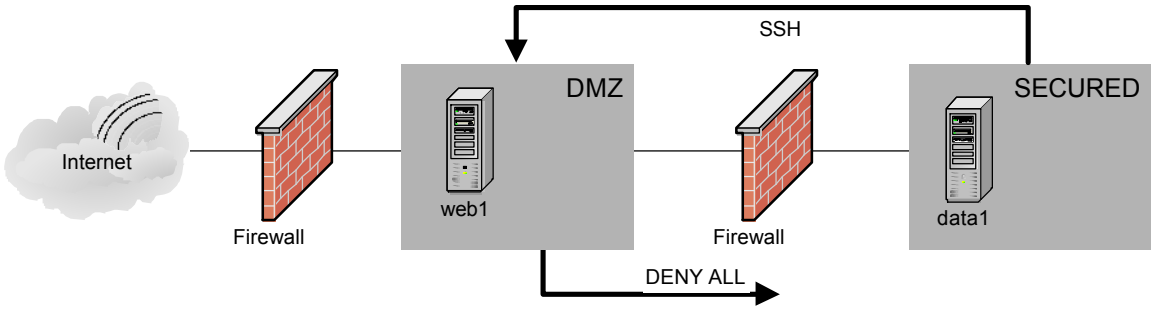

Figure 3

In figure 3 above, the second firewall is configured to deny all inbound traffic to the secured network. This being the case, using local SSH port forwarding from web1 to data1 will not work because even traffic from DMZ resources will not be permitted in. But as long as SSH traffic is permitted outbound from data1 to web1, then data1 can initiate the SSH session and create the encrypted tunnel.

Using the above SSH command on data1 with the '-R' switch, which specifies 'remote forwarding', data1 will instruct the SSH daemon on web1 to set up port which listens for 4100 traffic on its loopback interface, and passes that traffic back through the SSH session to data1 for decryption to Sybase.

Please note however, that using this command, you must specify '127.0.0.1' in the syntax, rather than 'web1' or the IP address of web1. Otherwise, you will set up a listening port on web1 that will listen for Sybase traffic on its *outside interface* and pass it on to data1 indiscriminately! Meaning that for all effective purposes, web1 will be acting as the database server out in the open! By specifying '127.0.0.1' or 'localhost', you will ensure that Sybase traffic will only be accepted from local processes on web1 alone.

This technique will also work with the single firewall approach (as illustrated in figure 2). You would just configure the sole firewall to deny any inbound traffic to the secure network, not only from the outside, but from the DMZ as well.

SSH is available for all Linux/Unix platforms as openssh, the free open source release at http://www.openssh.org. For Windows platforms, you can either obtain the commercial version from http://www.ssh.com or install Cygwin (a free Unix environment for Windows) and run openssh within Cygwin. Cygwin can be downloaded from http://www.cygwin.com.

Stunnel

While SSH port forwarding can be effective, some may be hesitant to utilize this technique due to the nature of the SSH connection. SSH port forwarding cannot be set up to run automatically as a service or a daemon. As a result, it generally requires manual intervention to set up the connection. And if anything disrupts

the SSH connection, the tunnel usually collapses and will have to be set up again before database communication can resume. (NOTE: This is not to say that SSH port forwarding is unreliable or worthless. I have personally had SSH tunnels running for more than 2 months without being disrupted.)

A good alternative is Stunnel, which is open source software designed to utilize Secure Sockets Layer (SSL) and add SSL encryption to any service that does not have native encryption capability. It can be obtained freely from http://www.stunnel.org.

There are a couple of advantages make Stunnel a compelling choice. First, it can run as a daemon on both the client and the server, and you can write startup scripts so that the listening daemon will start automatically if either server is rebooted. Also, because if follows the SSL standard, it can utilize all of the features of SSL, including a Certificate Authority (CA) and client certificates (in addition to server certificates). It also utilizes the security features of TCP Wrappers if running on a Unix/Linux platform.

The one disadvantage compared to SSH port forwarding is that it cannot perform the equivalent of remote forwarding. As a result, you must have the firewall protecting the database server configured to permit at least one port from the web server to the database server.

The operation of Stunnel is very similar to SSH port forwarding. In fact, you could almost call it SSL port forwarding, because it essentially does the same thing as SSH local port forwarding.

After installing and configuring the Stunnel software on both the database server (the Stunnel server) and the web server (the client), the commands to initiate the tunnel are almost as simple as SSH port forwarding.

On the client, the basic command would be:

```
#stunnel -P/tmp/ -c -d 127.0.0.1:4100 -r data1:4101
```

On the server, you would run a corresponding command like this:

```
#stunnel -P/tmp/ -p /root/stunnel.pem -d 4101 -r localhost:4100
```

On the client, you tell Stunnel to listen for port 4100 (Sybase) on 127.0.0.1 (the loopback) and intercept this traffic, encrypt it and pass it on to data1 using port 4101. (This is an arbitrary port of your choice, but it must be different from standard target port on the server.)

On the server, you essentially do the reverse, telling Stunnel to listen to the corresponding tunnel port you chose on the client (4101), decrypt this traffic, and

pass it on to the standard listening port for the database (4100). The "-p /root/stunnel.pem" parameter specifies the server certificate that it must present to the client; similar to a server certificate that is presented when you visit an SSL-enabled web site.

Stunnel is available for both Unix/Linux platforms and Windows.

Encryption is a valuable asset that will help protect your data, but many may be tempted to omit encryption because they think its overkill, unnecessary, or will slow down transactions too much. It's true that you will lose some speed when using encryption, but again it all boils down to risk analysis. And keep in mind that many attacks are initiated from the inside, as opposed from originating out on the Internet. Also, many institutions these days, primarily financial and banking, and also some healthcare as well, must follow very strict security rules that demand that all transactions, whether they are internal or external, must be encrypted [12].

Conclusion

These practices will go a long way towards protecting your database server from threats. Will implementing these actions ensure that your database is invulnerable? Absolutely not – there are no networks that are completely invulnerable. However, implementing these measures will make it extremely difficult for an intruder to steal data that is either in your database or is in transit through your network.

While the examples in this paper used a simple web server-to-database server scenario for public access in the DMZ, these practices should also be considered even if you are implementing a system for internal company access only. Leaving a database server that contains payroll or confidential employee information connected to the corporate LAN with no additional firewall protection or encryption could pose a big problem if you have an enterprising (and unethical) employee who spends their weekends learning how to sniff network traffic or how to take advantage of the latest Microsoft SQL Server vulnerability.

Also, keep in mind that this is by no means a comprehensive checklist of everything you can do to secure your databases. These practices focus on securing the network alone, and don't even address additional measures you should be taking, such as running intrusion detection software and vulnerability testing. Then there are the obvious action items that were not mentioned above, such as making sure that your operating systems and applications on the web server and database servers have the latest patches and security fixes applied. Many of successful break-ins of Microsoft SQL server could have been avoided if the latest patches were applied as soon as Microsoft made them available.

The weakest link at this point is the web server itself, because after all, that is the public resource that outsiders are being allowed to access. That being the case, you will need to ensure that all precautions have been taken to protect and harden the web server.

If you are one of those unlucky souls who runs the entire shop, meaning you are the network administrator, the web programmer and the database administrator, you will need to educate yourself about the security implications of these other areas as well. Otherwise, you should talk with your programmer/web developer and DBA and make sure that they have done their jobs. After all, wouldn't you be terribly upset, after doing all that work to secure the network, only to discover that your web developer has written code for the web server that submits credit card information to the database server using the 'sa' account and password? This would be the equivalent of buying the best security system and locks that money can buy to protect your house, but then leaving the key underneath the front step doormat.

Footnotes

[1] Lemos, Robert and Charny, Ben, "Egghead cracked; data at risk",
URL: http://zdnet.com.com/2102-11-526642.html

[2] Lemos, Robert, "Analysts: Egghead's inquiry cost millions",
URL: http://zdnet.com.com/2100-11-527001.html?legacy=zdnn

[3] Carrington, Damian, "Net thief grabs credit cards",
URL: http://news.bbc.co.uk/hi/english/sci/tech/newsid_597000/597828.stm

[4] Katayama, Fred, "Playboy.com gets hacked",
URL: http://money.cnn.com/2001/11/20/news/playboy

[5] Greene, Thomas C., "Amazon division hacked, thousands of CCs exposed",
URL: http://www.theregister.co.uk/content/archive/17384.html

[6] Sieberg, Daniel, "Hackers tap credit card info at Bibliofind",
URL: http://www.cnn.com/2001/TECH/internet/03/05/bibliofind/index.html

[7] Knight, Will, "Hackers steal one million credit cards",
URL: http://www.zdnet.co.uk/news/2001/9/ns-21473.html

[8] Delio, Michelle, "Hoosier Favorite Hack Victim?",
URL: http://www.wired.com/news/print/0,1294,44501,00.html

[9] Middleton, James, "Databases a soft touch for hackers",
URL: http://www.vnunet.com/News/1128592

[10] Farrow, Rik, “Web Servers: No Place to Hide”,
Network Magazine, February 6, 2002.

[11] Tippett, Peter, “The Crypto Myth: If you assume SSL is essential to Internet security, guess again.”
URL:
http://www.infosecuritymag.com/articles/may01/columns_executive_view.shtml

[12] MacVittie, Lorie “Cover Your Assets, Web Style”,
URL: http://www.networkcomputing.com/1314/1314ws1.html

References:

1. Chuvakin, Anton, “Your money or your life! Which would you rather lose to hackers: private customer data or your website?”,
URL: http://www.securitywatch.com/RES/June25.html

2. Gutzman, Alexis D., “Alternative Payments for the Newly Cautious”,
URL: http://dc.internet.com/news/print.php/941411

3. Olavsrud, Thor, “Egghead.com Gets Hacked”,
URL: http://www.internetnews.com/dev-news/article.php/10_543591

4. Richtel, Matt, “Credit Card Theft Thrives Online as Global Market”,
URL: http://www.nytimes.com/2002/05/13/technology/13CARD.html

5. Chapple, Mike, “Database Insecurity: Is Your Credit Card Safe?”,
URL: http://databases.about.com/library/weekly/aa121500a.htm

7. Aterma, Timo and Kleimola, Johannes, “Using publicly available tools and sniffers in hacking”,
URL: http://www.hut.fi/~jjkleimo/kurssit/tik110452/toolkits.html

8. Articsoft White Paper, “Does SSL protect your, or is it a condom that is open at both ends?”
URL: http://www.articsoft.com/wp_ssl_condom.htm

9. Anonymous/(‘orange’), “Intranet Security 101”,
URL: http://www.hackinthebox.org/article.php?sid=3661

10. Meinel, Carolyn, “Are You Giving Away Your Databases: Why Database Theft Is a Serious Problem”,
URL: http://www.messageq.com/communications_middleware/meinel_2.html

11. ECTalk Discussion Transcript, “Storing Credit Card Details”,
URL:
http://ecommerce.internet.com/community/best/article/0,,10183_484911,00.html

12. URL: http://isc.incidents.org/port_details.html

13. URL: http://www.openssh.org

14. URL: http://www.ssh.com/

15: URL:http://www.cs.uchicago.edu/info/services/ssh_tunneling

16. URL: http://www.stunnel.org/

17. Barrett, Daniel J. and Silverman, Richard, “SSH, The Secure Shell: The Definitive Guide”, O’Reilly & Associates, 2001.