SQL Injection

Despite the fact that Structured Query Language (SQL) Injection has been a well-known security threat for several years, this attack method continues to be associated with many Account Data Compromise (ADC) cases involving e-commerce merchants. Therefore, it is important to understand exactly what SQL Injection is, how it can be detected, and how to correct security vulnerabilities that may enable successful SQL Injection attacks.

SQL is the language used to communicate with common database systems. If a web application isn’t securely coded, hackers may be able to “inject” malicious code into an application which could allow them full access to a database system. SQL Injection attacks can be very easy to execute. An attacker only needs a fundamental understanding of the SQL language, access to a standard web browser, and internet access. To illustrate the point, an attacker can simply walk in to a coffee shop, rent time on an internet kiosk, and attempt to compromise a merchant system halfway across the globe.

Understanding the Problem

A proper understanding of SQL Injection begins with knowing how Web-based applications work. When visiting an online merchant, the e-commerce functionality you encounter usually consists of a three tiered architecture consisting of Web servers, application servers, and a database system.

Web Servers

Web servers are the front end of the overall web infrastructure and provide the connection with the customer’s web browser. Thus, Web servers tend to be generally accessible to anybody on the internet. These systems send the data received from the user’s browser to the application servers.

Application Servers

Application servers provide the business logic of the overall web site. These servers interface between the Web front end and the database back end. These application servers can manage a customer’s shopping cart, calculate shipping costs, and provide delivery times, etc.

Database Systems

Database systems store much of the content, pricing, and other information available on the merchant’s Web site. Most importantly, the database servers may store customers’ account information, such as payment card data and shipping addresses. These systems should not have direct access to the internet and should only be accessible from the application server and the internal network for administrative purposes.

Almost every action taken when shopping online triggers a matching request from the application server to the database server behind the scenes. In order for these matching requests to take place, the servers must use a common language — SQL is this common language. Potential intruders know that application servers and
database servers both use SQL to complete database queries and typically have access to Web servers from the internet. This enables intruders to use a Web server as a “front door” to attempt access to sensitive information stored in database servers within an e-commerce merchant’s network.

Since the hacker can view and interact with Web servers from the Internet and Web servers communicate with the targeted database servers via SQL, the hacker will seek ways to manipulate SQL commands to interact with database servers. Specifically, hackers will look to “inject” unexpected characters into SQL queries in hopes of gaining access to stored information or potentially to gain administrative access to the database itself. A primary concern about SQL Injection vulnerabilities lies in the multitude of potential variations that a hacker can use when attempting an attack. When a valid end user searches an online merchant’s Web site, variables in a SQL request are passed to the database server, thereby initiating queries. In a typical Web services model, the resulting output from each query is then passed along directly to the valid end user who made the request. But when the expected SQL variables are replaced by a savvy hacker with unexpected variables or characters, it is sometimes possible to make the database perform an unexpected action and expose sensitive information. Thus, by injecting unexpected variables in SQL commands the hacker may now have access to the information housed in the database. After gaining information about the database through SQL Injection queries, hackers will typically attempt to query the database through SQL Injection to retrieve sensitive information, such as payment card data.

Detecting SQL Injection Attempts

Now that the problem is understood, the focus must turn to prevention. In order to prevent SQL Injection, IT security professionals need to understand and identify any potential ‘red flags’. Even when initiated by a talented hacker, SQL Injection attacks can be easy to recognize, provided there are some basic logging capabilities enabled. These basic logging capabilities must cover two critical areas: the Web server logs and the database server. Web servers are the interface with the Internet; as such they are often the initial touch point for SQL injection attempts and the first place to look. Since these database servers are not directly accessible, all information retrieval attempts must first go through the Web server. For this reason, the Web server logs tell deal of information. IT professionals should examine the Web server logs regularly. Hackers will use application security holes in these types of Web pages to pass these SQL Injection attempts to the protected database servers. However, even very basic Web server logging can record these attempts and will typically reflect the actual SQL queries employed by the hacker. This information will not only help identify that an attack is underway but, in the event of a compromise, can also help in understanding exactly what sensitive information has been accessed.

The next place to look is the database server. Just as your Web server logs can identify the signs of attempted or successful SQL injection, your database server logs can do the same.
thing. IT professionals should monitor the log files on the internet related database servers regularly, looking for unusual or excessive SQL queries or other suspicious activity. In addition to the database server logs, unusual spikes in memory or usage should also be viewed suspiciously.

Identifying The Vulnerability

Just as SQL Injection attempts can be easily detected via logs, the vulnerability can be just as easy to diagnose before it happens.

Many Web pages vulnerable to SQL Injection can be identified using vulnerability scanning tools provided by Approved Scanning Vendors (ASV). A list of ASVs can be found at: https://www.pcisecuritystandards.org/approved_companies_providers/approved_scanning_vendors.php. These tools can help determine exactly how vulnerable a site is to SQL injection and can help identify means to prevent a potential compromise. In addition, free and open-source tools exist that can be used to regularly check applications for SQL injection vulnerability.

Preventing SQL Injection

Even with the ability to detect and identify SQL Injection, it currently remains a powerful and prevalent tool within the hacking community. This leads to a simple question: why? Perhaps the main reason is that SQL Injection vulnerabilities are frequently overlooked by security administrators because this type of vulnerability is not a problem that can be fixed with a patch or a simple update. SQL Injections are enabled by an improperly coded web application or misconfigured database server. Getting to the root cause of this exposure is a matter of properly understanding the fact that SQL Injection vulnerabilities exist at the Web application layer.

Taking a proactive approach within the Web application development lifecycle and with the development team is the best approach to prevent SQL injection and other application vulnerabilities. Establishing standards for code and application development while educating development teams on the dangers presented by SQL injection vulnerabilities goes a long way to protecting both the environment and the end consumer.

Input Validation is the primary standard to add to the development lifecycle. The validation of the variables or inputs submitted in a Web-based request is the key to preventing SQL Injections. Although relatively straightforward, input validation controls are best implemented during the application development process because they are highly customized for each Web application. Due to this high degree of customization, proper input validation controls cannot simply be downloaded and applied after the application has gone into production. In fact, retrofitting these validation controls can be an expensive undertaking. It is estimated that retrofitting changes into applications or projects can cost up to 3X the amount of the original implementation. Make sure secure coding and vulnerabilities scans are part of the lifecycle requirements and business as usual.

A Web Application Firewall (WAF) may also be used to protect against SQL injection attacks. A WAF is a special type of firewall that can protect web infrastructure from various types of common web based attacks in addition to SQL Injections. These systems are generally placed in front of the web servers to filter out malicious attacks before they can enter any point of the web infrastructure.

As neither secure coding nor WAFs are 100 percent fool proof, ideally both practices should be used simultaneously to optimize risk reduction.

Once the SQL injection vulnerabilities have been addressed, it is important to consider that unexpected points of exposure can always occur. Application updates, freshly installed software, and operating system maintenance patches (common to daily IT operations) can often create new vulnerabilities where they previously did not exist. For this reason it is critical that the proactive application security measures mentioned here are reinforced with regular security testing and vulnerability scanning. In the world of data security, information is not only power, it is protection.

PCI Requirements On Securing Web Applications

The PCI Data Security Standard, Payment Application DSS and the requirements therein were primarily built off the results of known vulnerabilities and attack vectors found in Account Data Compromises (ADCs) within our industry. Specifically, PCI DSS requirements 6.5 and 6.6 and corresponding PA-DSS requirements are focused security controls driven by the high number of ADC’s caused by SQL injection flaws.

PCI DSS requirement 6.5 states that applications must be developed with secure coding methodologies, including the prevention SQL injection flaws.

DSS requirement 6.6 states there are two options to deter the impact of web application vulnerabilities. Either, secure coding reviews or a Web Application Firewall may be used to meet the intent of this requirement. (However, as previously mentioned, utilizing both mechanisms simultaneously are ideal.)