SQL Injection in Microsoft Environments

I. Introduction

Attacks on web applications have become one of the leading information security concerns over the past few years. The Web Application Security Consortium's "Web Application Security Statistics 2008" report analyzed 12,186 sites and found that "more than 13% of all reviewed sites can be compromised completely automatically." and "about 49% of web applications contain vulnerabilities of high risk level (Urgent and Critical) detected during automatic scanning." [1]

SQL Injection attacks are a large part of this concern and can be used for a myriad of malicious activities including: subverting authentication, stealing data, compromising servers, planting malware and pivoting to attack other internal corporate resources. One of the more interesting facts about SQL Injection is the length of time they have been around. Rain Forest Puppy first introduced the concept publicly in 2000 with the release of his paper "How I Hacked Packetstorm."[2] While these attacks have been around for nearly ten years, as an industry we seem to be struggling more and more with these issues. In 2008, the Asprox botnet began using large scale SQL injection in order to spread malware.[3] More recently, SQL Injection has been credited with the theft of nearly 130 million credit card numbers in the Alberto Gonzalez case.

The challenge for security professionals is getting organizations to see the danger SQL Injection can bring and realize it takes a holistic approach to solve the problem. There is no silver bullet security technology which can prevent SQL Injection. Solving the problem requires a coordinated effort across the development, operations and security teams. Adding to the challenge, is the fact that many of these web vulnerabilities exist in internally developed systems and, therefore, must be identified and remediated internally. There is no magic patch.

II. SQL Injection Detection and Exploitation

While our focus is primarily on Microsoft environments, SQL injection is not limited to Microsoft SQL. Environments based upon other technologies such as MySQL, Oracle and PHP can also be susceptible to SQL injection. For purposes of this discussion we are assuming that you have an understanding of basic manual SQL injection attacks such as "AND 1=1". Those looking for a nice guide to SQL injection fundamentals should consider reading "SQL Injection Walkthrough" at http://www.securiteam.com/securityreviews/5DP0N1P76E.html.

Since nearly all organizations have an existing web infrastructure, detecting SQL injection in existing applications is critical. SQL injection scanners are a great place to start when looking at SQL injections. Scanners are a great way to take care of the "low hanging fruit" and automate tasks which would normally take time to find manually. The important thing for organizations to understand is that scanners are not perfect and will have both false positives and false negatives. The following tools are each available at no cost and serve various purposes in detecting SQL injection.

[1] http://projects.webappsec.org/f/wasc_wass_2008.pdf
[2] http://www.wiretrip.net/rfp/txt/rfp2k01.txt
[3] http://www.darkreading.com/security/app-security/showArticle.jhtml?articleID=211201082

HP Scrawlr (SQL Injector and Crawler)
http://www.communities.hp.com/securitysoftware/blogs/spilabs/archive/2008/06/23/finding-sql-injection-with-scrawlr.aspx

Hewlett Packard Scrawlr is a good starting point for the SQL injection beginner looking to play with a scanner. The tool is simple to use and interpret through the use of its Windows GUI interface. Scrawlr will crawl a website and attempt to find vulnerabilities in URL parameters. The tool will return the page found vulnerable, the parameter and database table names. Scrawlr is limited in scope as it limits scans to 1,500 pages, does not support forms input, javascript or authentication, nor does it perform blind SQL injection. Scrawlr is by no means the most advanced or capable tool, in fact it is very basic, however but it is a great tool to quickly test for SQL injection in URL parameters and, more importantly, a great start in looking at SQL injection.

SQLMAP - http://sqlmap.sourceforge.net/

Sqlmap is an open source project led by Bernardo Deamele A. G. which is designed to find and exploit SQL injection vulnerabilities. The tool supports MySQL, Oracle, PostgreSQL and MS-SQL back end databases. Sqlmap is written in Python and can be run from any operating system with Python support. While sqlmap is a more complicated tool, it is also much more powerful.

Sqlmap has a myriad of options which are fully explained in its documentation. The examples shown here are designed to help explain the power of this tool. Sqlmap has five levels of verbosity of output which is set using the –v parameter. This is handy depending up the level of detail you want to see or possibly provide back to developers. Sqlmap doesn't scan or crawl and entire website like traditional tools, but rather can specify a specific target URL or generate targets based upon logs from Burp and Webscarab or results from Googledork. For example, you could use Googledork to search for any page containing a "id=" parameter and use those URLs as targets for sqlmap to attempt to exploit.

After targets have been determined, sqlmap can specify many details about how to connect to the target URL including: method to be used (get or post), http header (cookie, referrer and user-agent), authentication options and various performance settings. By default, sqlmap will attempt to exploit parameters via blind SQL injection, however various other techniques can be set such as testing for multiple stacked queries, time based blind SQL injection and various union query options.

Once the target has been identified and parameters set, sqlmap performs various enumeration and exploitation against vulnerable pages. Sqlmap can determine numerous details about the backend database including: database type/version, operating system type/version, database names/tables, database users and even their password hashes. Exploitation options are equally robust. Sqlmap has the ability to provide the end user a sql shell, execute an operating system command, deliver an operating system shell, deliver meterpreter or vnc and even attempt privileged operating system escalation.

Sqlmap is a feature rich tool which can take you through all the steps involved in the SQL injection process. Given the vast array of exploitation options, sqlmap can certainly be used for malicious purposes, however having this exploitation ability is a fantastic way to showcase these vulnerabilities to skeptical audiences. Developers cannot argue about false positives after watching someone get shell through a vulnerable parameter. Also, examples like this help show management how serious web application vulnerabilities can be. Again, it takes all the technical jargon out of the discussion when someone can see all their database contents being gathered remotely.

IDS/IPS

Many organizations have existing Intrusion Detection/Preventions systems in place which have traditionally been focused on addressing network based attacks. Today's systems also include rules which try to attempt various web application attacks such as SQL injection. Emerging Threats has a list of SNORT rules for known vulnerable web applications. [4] Commercial vendors such as Juniper and ISS provide more generic SQL injection rules such as detecting a SQL command chain in a URL.[5] Intrusion detection/prevention systems are not the answer to solve SQL injection, but can add an additional line of defense. They also may help in showing staff that there are attackers regularly probing web sites looking for these vulnerabilities.

III. SQL Injection Mitigation

Preventing SQL injection attacks takes a coordinated effort across the development, operational and security teams. Security teams can help by implementing defense in depth technologies such as web application firewalls, however organizations should consider these a complimentary piece of the puzzle, not a silver bullet. There are numerous WAF options both commercial and open source. Microsoft environments should consider evaluating URLScan, a free filter from Microsoft which can provide some WAF-like filtering for IIS web servers.

Database Server Hardening

Operations teams can help by hardening database servers. While this won't mitigate all SQL injection attacks, it can significantly reduce the scope of the attack. First, the account used to run the database server should have as little access as possible. Second, accounts used to connect to the databases should have minimal access. Try limiting accounts to read-only where possible or only giving access to specific tables. Third, if stored procedures are used, limit access to only those required by the application. Microsoft provides further details in the following best practices guides for securing SQL server.

MS SQL Server 2008 - http://technet.microsoft.com/en-us/library/bb283235.aspx
MS SQL Server 2005 - http://technet.microsoft.com/en-us/library/bb283235(SQL.90).aspx

[4] http://www.emergingthreats.net/rules/emerging-web_specific_apps.rules
[5] https://services.netscreen.com/restricted/sigupdates/nsm-updates/HTML/HTTP:SQL:INJECTION:CMD-CHAIN-1.html

Stored Procedures, Parameterized Queries, Escaping User Input and Input Validation

While network and database protections can help, the best methods for SQL injection mitigation lie in writing secure code and limiting a user's ability to manipulate database queries. Stored procedures, parameterized queries, escaping user input and input validation are four options for limiting SQL injection vulnerabilities.

Stored procedures involve predefining SQL queries and storing them inside the database. Instead of the application creating queries, the application would call a stored procedure in order to query the database based upon input provided. This can help limit an attacker's ability to manipulate the SQL query; however it may still be possible when complex queries are used. Input validation may still be required to mitigate these instances.

How to: Create a Stored Procedure SQL Server 2008
http://msdn.microsoft.com/en-us/library/ms345415.aspx
How to: Create a Stored Procedure SQL Server 2005
http://msdn.microsoft.com/en-us/library/ms345415(SQL.90).aspx

Parameterized queries are another option for developers. Parameterized queries have all the SQL code defined first before any variables are passed to the code. This differs from dynamic queries where SQL queries are constructed with both code and variables in the same statement. "Prepared statements ensure that an attacker is not able to change the intent of a query, even if SQL commands are inserted by an attacker. In the safe example below, if an attacker were to enter the userID of tom' or '1'='1, the parameterized query would not be vulnerable and would instead look for a username which literally matched the entire string tom' or '1'='1."[6]

Escaping user input is another SQL injection best practice which can be effective for certain attacks. Database systems each have a unique method for escaping special characters and thus indicating that they are part of data input, not a database command. From a SQL injection standpoint this option is effective because any characters supplied as input with the intention of being SQL commands will simply be represented as text input. The SQL server built-in functions QUOTENAME and REPLACE are useful for escaping input in Microsoft environments. [7]

Input Validation

Performing input validation is single most effective method for preventing SQL injection. Unfortunately, it is also one of the more challenging. The concept of input validation is simply to limit a client's ability to submit data which could become malicious. For instance, a zip code field should only be allowed to enter five numbers. Coding for input validation is much more effective when a white list approach is taken (permitting only known good) vs. blacklist (blocking known bad) since there are so many possible malicious characters/strings. Clearly some input is much more difficult to properly sanitize than others. A quotation mark in a person's name, for instance. Situations like this lend themselves to performing multiple

[6] http://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
[7] http://blogs.msdn.com/raulga/archive/2007/01/04/dynamic-sql-sql-injection.aspx

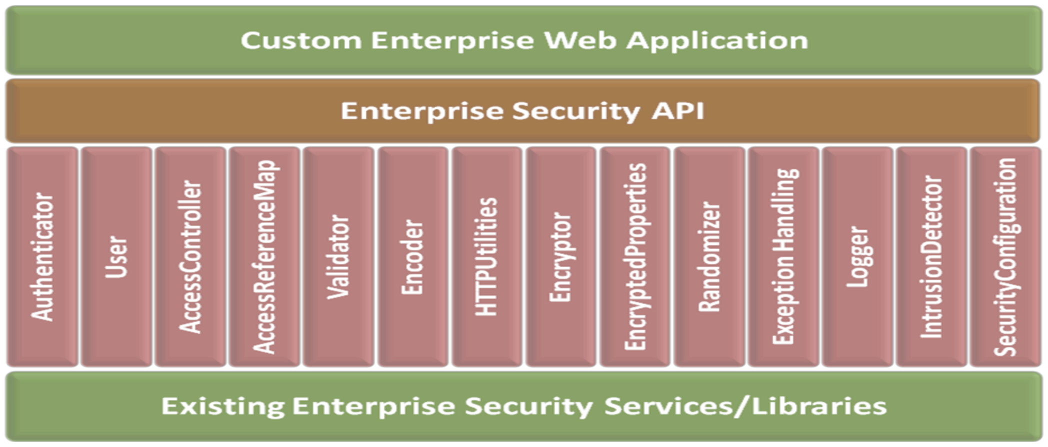
mitigation functions. In this example the data could be escaped after being properly validated. The point being that, just like many other areas of information security, no single method is completely and a defense in depth approach should be used.

OWASP ESAPI - http://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API

The Open Web Application Security Project (OWASP) is a non-profit organization which provides free and open solutions for web application security. OWASP has many free tools, best practice guides, videos, conferences and local chapters. One of the most interesting projects is the OWASP Enterprise Security API (ESAPI). ESAPI Toolkits have been created for Java, .Net and Classic ASP. ColdFusion, PHP, Python and Haskell are under development.

ESAPI was designed around the idea that creating security is a very challenging process both in time and knowledge. Most developers should not be creating security, but rather should have a set of easy to use controls to handle security related functions. ESAPI is designed to take care of the most commonly needed security functions so developers can focus their efforts in other areas. Additionally, ESAPI has been designed so that organizations can use other security libraries for easy integration into existing systems.

ESAPI Toolkits are a divided into a collection of 14 categories shown in below. Each category has a number of methods. ESAPI addresses many security related issues including authentication, logging and access control. From a SQL injection standpoint, the Validator category is the most relevant category providing methods for the canonicalization and validation of user input. Organizations, especially those without secure development standards, should consider using ESAPI. All web developers should monitor the progress of this project as it has the potential to become an industry standard base security API.



IV. Conclusion

Regardless of the technology used in a web infrastructure, SQL injection has proven itself to be a major concern for any organization. While there are many options for detecting vulnerabilities, limiting your exposure, and blocking known malicious traffic, there is no substitute for properly written code. Providing training and strong development standards can go a long way to ensuring your organization's web infrastructure and associated data are protected.