SQL Injection Evasion Detection

Executive Summary

The detection of SQL injection attacks has primarily been accomplished through pattern matching techniques against signatures and keywords known to be malicious. Until recently, this technique has been successful. Now attackers are hiding their malicious intent in a variety of ways to escape detection.

These attempts at evading detection require new technology and techniques in order to be discovered and stopped before reaching critical systems and causing the exposure or destruction of corporate data.

Evasion detection engines are a new form of protection against such attacks. These engines recognize attempts at cloaking the malicious code that can result in a successful SQL injection attack and act upon it, resulting in the successful detection and subsequent prevention of hidden SQL injection attacks.

The Trojan Horse

Historically, SQL injection attacks have been prevented through the use of pattern matching techniques against signatures and keyword-based stores to identify potentially malicious requests. Like the Trojan Horse of Greek lore, dangerous code is hidden inside a valid request and only becomes apparent after it has been accepted inside the walls of the data center.

Threat prevention systems such as IPS and application firewalls are learned in this invasion technique and are capable of sniffing out these hidden attacks before they are accepted into the data center and allowed to pass through the application infrastructure and wreak havoc. Over the years, lengthy lists of the possible combinations of keywords and characters that can result in a successful SQL injection attack have been compiled, leading to the creation of signature databases. Because databases each have their own fairly unique implementation of SQL, the industry standard database query language, these signatures often include database specific attacks to ensure the detection of the widest array of SQL injection attacks.

In addition to signatures, keyword matching is also often used to prevent edge cases from successfully penetrating the defenses of an application. Certain keywords such as “DROP” and “UNION” are often sought out as potential attacks and subsequently refused as a general rule when discovered within requests destined for any database implementation.

Attackers have learned that their “Trojan Horse” is easily recognized by application firewalls and thus denied entry into the application infrastructure completely. It was necessary for them to find a new way to carry their attacks into the data center, and thus was born the “Trojan Zebra.”

The Trojan Zebra

The Trojan Zebra looks a lot like the Trojan Horse used to perpetrate attacks in the past, and yet its coloring and pattern is different. Threat prevention systems, looking for specific patterns, are confused by the appearance of the Trojan Zebra and allow it to enter the data center where its hidden attack is easily carried out against corporate databases.

SQL injection attacks today are successful because the zebra looks a lot like a horse, but the stripes on its coat form a different pattern, one that is not contained in today’s signature databases. Worse, the striped pattern varies from zebra to zebra, rendering pattern matching techniques today virtually useless.

SQL injection attacks today are like the stripes on the Trojan Zebra—the danger is there, but it is hidden within the varied striped patterns and is therefore virtually undetectable. Attackers use the flexibility of string-based parameters and the diversity of language to hide traditional attacks from threat prevention systems. Just as the pattern of stripes on a zebra is unique, so is each SQL injection attack perpetrated using new techniques to hide their true intent.

Traditional SQL Injection Attack	Evasion Technique	Hidden SQL Injection Attack
…71985' OR '1' = '1'	White Space Manipulation	…71985'OR'1'='1'
'&id=111 UNION /**/ SELECT *…'	'C' Syntax Comment	&id=111/*This is my comment…*/UN/*Can You*/IO/*Find It*/N/**/S/**/E/* */LE/*Another comment to*/CT/*Find. Can you dig*//*it*/*
1 UNION SELECT ALL FROM WHERE	Encoding: HEX	1 UNION SELECT ALL FROM WHERE
	Encoding: BASE 64	MSBVTklPTiBTRUxFQ1QgQUxMIEZST00gV0hFUkU=
	Encoding: DECIMAL	&#49&#32&#85&#78&#73&#79&#78&#32&#83&#69&#76&#69&#67&#84&#32&#65&#76&#76&#32&#70&#82&#79&#77&#32&#87&#72&#69&#82&#69
…71985' OR '1' = '1'	Variations on a Theme	…71985' OR 'city' = 'seattle'

There are essentially four categories of evasion techniques in use today:

White Space Manipulation

Almost all modern signature-based SQL injection detection engines are capable of detecting attacks that vary the number and encoding of white spaces around the malicious SQL code. What these engines are not capable of handling is a *lack* of white spaces around that same code. Because the pattern matching and signatures method are generally looking for a pattern of text that includes *one or more* spaces, they fail to detect the same pattern of text when no spaces are included.

Also note that though white space manipulation usually involves the removal of white spaces from a potential query, the following characters may also be used to confuse signature and pattern matching based detection systems:

- Tab
- Carriage return
- Line feed

This attack evasion works because the SQL parsing engine of most databases does not care about white space and formatting characters and are in fact generally written to handle a variable amount all these characters around keywords.

Comment Exploitation

In the past attackers used the common double hyphen comment syntax, for example, --, to mask their intentions. Many engines today detect this attempt, and thus attackers have changed tactics, using the widely supported "C" style comment syntax, for example, /* */, to evade detection.

Attackers use “C” style comments in place of spaces, to separate commands that are commonly used together in attacks but are easily detected through signature matches, and in some cases to break up keywords.

For example, UNION is a common keyword used in the commission of SQL injection attacks. While signature-based detection engines may not always flag the UNION keyword as a potential attack because it is not followed by a space or another known keyword, a keyword match *will* flag UNION as a potential attack. To avoid this detection, attackers may attempt to use “C” style comment markers to break up the keyword, for example, UN/**/ION, thus defeating not only signature matching but keyword matching as well.

This attack works because most SQL parsing engines in databases today strip all comments from queries *before* parsing the SQL (everything between /* and */), thus leaving a perfectly valid SQL statement.

Encoding Techniques

Encoding techniques are perhaps the easiest method of defeating detection through signatures or pattern matching engines. This is because encoding has the effect of completely changing the text much in the same way cryptography changes the text it is meant to hide from unintended viewers.

The most common encodings used to evade detection are:

- URL Encoding
- Unicode/UTF-8
- Hex Encoding
- char() function

Encoding techniques work due to the heterogeneity of the web, the need to support multiple languages and character sets, and the failure of threat-prevention solutions to properly decode or support multiple code pages for incoming requests before applying threat detection methods.

Variations on a Theme

There are multiple variations of evasion attacks that rely upon the native capabilities of the SQL language as defined in the SQL99 standard.

Concatenation

Concatenation breaks up identifiable keywords and evades detection by taking advantage of the SQL engine’s native ability to build a single string from multiple pieces. Concatenation syntax varies based on the database, but in general uses either the plus sign (+) or the pipe (||) character to indicate concatenation at the SQL level.

For example:
EXEC(‘SEL’ + ‘ECT US’ + ‘ER’)
EXEC(‘SEL’ || ‘ECT US’ || ‘ER’)

Conversion

Conversion techniques make use of the SQL engine’s native ability to convert data types. This allows the attacker to evade detection by introducing valid SQL functions that change the signature of the statement.

For example:
OR username = char(37) /* 37 is equivalent to the SQL wildcard character, % */

Variables
Many engines allow the declaration of variables, which can then be used to circumvent not only application firewall detection, but additional, code-based input validation as well.

For example:
; declare @myvar nvarchar(80); set @myvar = N'UNI' + N'ON SEL' + N'ECT U' + N'SER'); EXEC(@myvar)

The Trojan Zebra is successful at delivering attacks because these evasion techniques result in an impossibly high number of permutations in signatures. In many cases they can be used together. For example, one can manipulate white spaces in the original request and then encode the entire request, requiring not one but two successful detections in order to prevent the attack. This sudden increase in the number of possible attack signatures cannot be accounted for using traditional pattern matching or signature database techniques.

A new technique is required to detect not only the attack, but the attempted evasion of that detection.

The Policy Evasion Detection Engine

In order to successfully detect both the SQL injection attack as well as the evasion of that detection it is necessary to incorporate evasion detection technology into existing threat prevention solutions, such as F5's BIG-IP® Application Security Manager (ASM).

ASM now includes sophisticated anti-evasion technology designed to detect and neutralize SQL injection evasion attacks. This technology, the Policy Evasion Detection Engine (imPEDE), is capable of recognizing a variety of evasion attempts and subsequently preventing them from reaching their intended target.

ASM's imPEDE accomplishes this task by normalizing data that would typically slip through traditional threat prevention systems that rely on signatures and pattern-matching systems. By normalizing the data regardless of its arriving format, imPEDE is able to remove the impact of evasion attempts on matching against signature databases and keywords.

ASM's imPEDE normalization techniques work because the SQL injection attacks themselves have not changed, just the manner in which they are embedded within requests. By detecting the attempts to evade the underlying system, imPEDE allows ASM's proven methods of preventing SQL injection to continue to be successful at protecting applications and data stored in corporate databases.

imPEDE further enhances security without degrading performance—a common concern regarding the deployment of threat prevention systems and web application firewalls in general—by employing policy-based detection. imPEDE allows policy to determine what URLs should be examined and which ones are assumed threat free. Most commonly, policies are applied to URLs which submit data but not necessarily those simply retrieving and displaying data, as those are least likely to contain potential threats from attackers.

imPEDE's policy based approach is flexible, allowing the administrator to determine what should and should not be protected, and can be further be enhanced by ASMs ability to monitor and report upon site changes that may include new URLs or changes to the behavior of existing URLs. This enables administrators to make decisions regarding the level of security necessary on a per URL basis as the site changes, making site based exploration a much simpler and easier task.

Conclusion

A purely signature or keyword matching based threat prevention system such as an IPS cannot properly deal with evasion attacks. While these techniques are a good basis for preventing known threats from reaching applications, such a static method of threat detection cannot continue to expect to be successful against the evolving dynamic nature of web application attacks, in particular SQL injection.

Advanced technology, such as ASM's imPEDE, is required in order to detect the evasions used today to penetrate through existing threat prevention solutions. These solutions, such as IPS or stand-alone web application firewalls, provide protection primarily at the web application layer and cannot address the broader issue of application delivery security. ASM, when coupled with the network and application transport layer security of an application delivery platform and integrated into an Application Delivery Network, offers a holistic solution for ensuring the secure, fast, and available delivery of applications.