Threat Brief: LizaMoon Mass SQL Injection Attack

LizaMoon, a new mass SQL injection attack, was discovered last week. It's the latest version in a series of mass SQL injection hacks monitored over the last several years. This type of attack has become extremely common as a tool used by cybercriminals, as it is a relatively easy way to enable rapid growth of botnets, which are currently the most popular and widespread method for monetizing a breach.

Mass SQL injection attack techniques differ slightly from targeted SQL injection attacks since they normally have a different intent. In most cases, mass SQL injection attacks are used to insert JavaScript and iFrames into the pages of legitimate websites, redirecting victims to malicious pages with client-side exploits and/or scareware and not necessarily to gain direct access to the database itself. In this specific case, the goal of the LizaMoon attack is the insertion of a script tag pointing to a malicious website promoting fake anti-virus software.

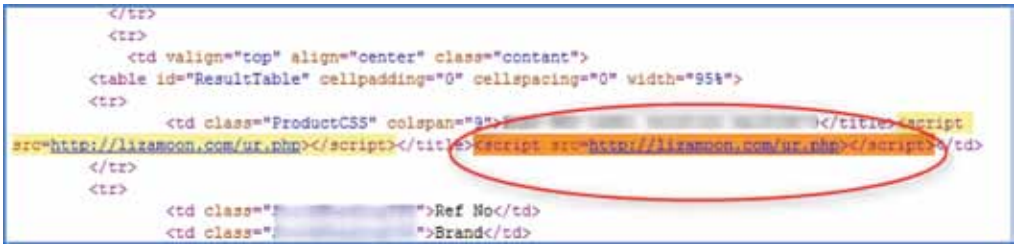

Figure 1. LizaMoon script tag pointing to a malicious site.

The second difference stems from the fact that the attack is highly "automated" in order to achieve its broad site penetration. Attacks require the discovery of a common attack vector for all targeted websites to robotize the infection. There are several ways to achieve this—for example, by exploiting a vulnerability in the web server application or using a generic SQL injection tool that exploits vulnerabilities in widespread web applications or platforms. This appears to be the case with LizaMoon, as the exploit is apparently finding success in injecting a broad range of applications utilizing SQL server back-end databases.

The attack's next step is to obfuscate the injection to bypass classical signature-based security engines. Below is an example of the actual encoded attack query:

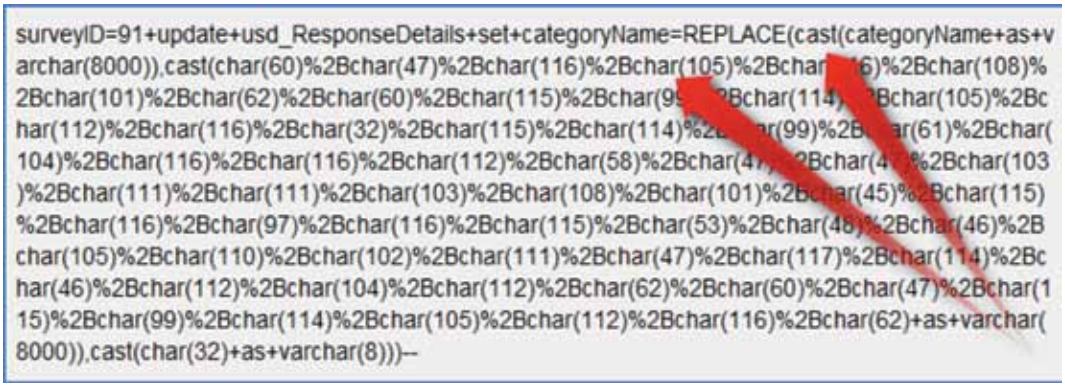

Figure 2. LizaMoon encoded attack query.

The LizaMoon injection doesn't have a particularly advanced obfuscation technique—it simply implements a well-known CAST trick combined with the character (CHAR) function. However, even this primitive method obscured the threat enough to successfully bypass common security approaches and infect more than one million pages on the Internet, highlighting the poor level of web application and database security prevalent in most organizations. While the full scope of the applications affected is not yet known, it is clear that it has impacted a large number of sites and early feedback from those users is that the remediation process is running into dozens and hundreds of hours.

The encouraging fact is that these mass, automated attacks have their downside as well: a non-customized approach. The result is that in parallel with the successful attacks, there will also be a large number of failed attempts.

Figure 3. A failed database attack attempt.

In the above case, although the code was injected into the database, it does not execute correctly because it was sanitized during an internal transformation. Had this been a more targeted attack, with more active hacker involvement, the perpetrator would have been able to easily modify the attack to properly embed the malicious script. In fact, if the above example were common, we would expect a variant of LizaMoon that specifically targets application code that does such transformations. This is still a possibility in the coming weeks.

Recommendations for Preventing Breaches

Clearly, the damages from such an attack are significant, ranging from lost time to remove the inserted scripts or restore from backups to the potential reputation damage from exposing website visitors and customers to a fraudulent site or a site that will infect their systems with malware. What can be done to prevent these attacks?

First, the vulnerabilities these techniques exploit (generally and in this specific case as well) are not normally in the database itself, but in poorly written application code. Whenever possible, bind variables should be used, as the resulting code will not be susceptible to this type of injection.

The reality is that for many organizations, the use of third-party applications or legacy applications with limited developer resources available means that code changes to mitigate these weaknesses are not always possible. In addition, the threat of a specific, targeted SQL injection attack, or a new zero-day vulnerability in the database itself, is still a concern.

In these cases, a real-time database monitoring component is an important element in providing security-in-depth for your applications and databases. McAfee® Database Activity Monitoring detects this type of attack out of the box, as can be seen in the snapshot from a standard installation of the product subjected to the LizaMoon attack.

Figure 4. Detection of the LizaMoon SQL injection attack by McAfee Database Activity Monitoring.

The SQL query, despite being obfuscated, is caught by one of the system's generic virtual patching rules, in this case a rule that detects attempts to inject code into SQL server. Nearly 400 rules come prepackaged with McAfee software to prevent exploit of database-specific vulnerabilities addressed by a patch that may not yet have been installed, or, as happens here, for common threat vectors to databases and the applications built on top of them.

It is important to note that not all database monitoring solutions provide protection for these general attacks, and, moreover, those that rely on monitoring SQL traffic will typically miss encrypted, encoded, or otherwise obfuscated attacks. Our unique approach of monitoring process memory ensures that as the command is translated by the database management system (DBMS) into an execution plan, it is evaluated against the security policy.

McAfee
2821 Mission College Boulevard
Santa Clara, CA 95054
888 847 8766
www.mcafee.com

26600alrt_lizamoon_0411_fnl_ASD