Lessons Learned From the Yahoo! Hack

How SQL Injection Vulnerabilities in Third-Party Code Can Make for Security Cloudy

1. Executive Summary

On December 2012, an Egyptian hacker who calls himself ViruS_HimA, claimed to have breached Yahoo!'s security systems and acquired full access to certain Yahoo! databases, leading to full access on the server for that domain. Technically, we found that the hacker was able to determine the allegedly vulnerable Yahoo! application and the exact attack method – error message based SQL injection for the MSSQL (Microsoft SQL Server) database (DB).

From a business perspective, this attack underscores the security problem posed by hosting third-party code – as is often done with cloud-based services. In fact, according to a survey from PricewaterhouseCoopers, 23.6% of respondents say that cloud computing has increased vulnerabilities, and the largest perceived risk is the uncertain ability to enforce provider security policies.\(^1\) In the Yahoo! incident, the vulnerable application was probably not coded by the Yahoo! team, and not even hosted on Yahoo!'s server farm. This left Yahoo! with full responsibility for securing the application on one hand, and a very limited capability to actually control the code, on the other hand. This episode underscores technical and business urgencies:

Technically, security teams should:

- Protect third-party Web applications against SQL injection and other Web attacks: Incorporate security into the software development life cycle, perform penetration tests and vulnerability assessments on the application, and deploy the application behind a Web Application Firewall (WAF).
- Harden your system: When the application is promoted from development to production, the system configuration must be hardened to disable any irrelevant parts that may help the attacker. In the hardening process detailed error messages should be disabled, excessive file and directory permissions should be restricted, source code leftovers should be deleted, and so on.

From a business standpoint, executives should always assume third-party code – coming from partners, vendors, mergers and acquisitions – contains serious vulnerabilities. Although our technical recommendations take precedence, we recommend:

- Put in place legal requirements in a contract for what you will and will not accept from a security perspective.
- Incorporate security due diligence for any merger or acquisition activity.
- Require coding standards and security requirements in every specification between you and the third party.
- Demand metric reports for security of the vendor’s code that are repeatable and verifiable.
- Require that all security requirements are met prior to the first time the code is executed in your environment.
- Require a comprehensive review of possible vulnerabilities resulting from new external services operating in conjunction with your current services.
- Require a report specifying security issues and measures taken to address them for every task and deliverable from the vendor.

\(^1\) PwC 2012 Global State of Information Security Survey
2. Detailed Attack Analysis

The hacker has released the following screenshot as an evidence for the successful hacking

![Hacker Evidence Screenshot]

**Figure 1 The hacker's hack evidence screenshot**

In this section, the technical details of the attack that were revealed by this screenshot are analyzed.

Note: We have covered the topic of SQL injection (SQLi) in previous HII reports; however we include a brief primer to SQL injection in section 2.1 to make this report self-contained. If you are already familiar with the subject, you can start with section 2.2.

### 2.1 SQL injection 101

In a SQL Injection attack, attackers exploit Web application vulnerability in order to access the organization’s data in an unauthorized manner. For laypeople, this means typing computer code in the fields of a Website’s form. For example, instead of typing in a credit card number or a last name, a hacker types in something technical that looks like ‘x’=’x’. When clever code is used, this action tricks the Website into coughing up sensitive data.

In geek speak, SQL injection is a technique used to take advantage of non-validated input vulnerabilities to pass SQL commands through a Web application for execution by a backend database. Attackers take advantage of the fact that programmers often chain together SQL commands with user-provided parameters, and can therefore embed SQL commands inside these parameters. The result is that the attacker can execute arbitrary SQL queries and/or commands on the backend database server through the Web application.

The potential results can be disastrous. For example, attackers may be able to retrieve the organization's intellectual property, customer account information, and other sensitive data. A successful SQLi attack may also allow the attacker to steal the site’s administrator password, giving the attacker full control over the Web application.

Other times, a compromised site can host an attacker’s code which may lead site visitors to download malware (aka “Drive-by Downloads”). SQLi attacks also allow the manipulation of data, enabling – for example – the defacement of the Website.

---

A SQLi attack usually starts with identifying weaknesses in the applications where unchecked users’ input is transformed into database queries. Goal-oriented attacks continue with abusing these weaknesses in a repetitious trial and error process in order to discover the exact structure of the application’s database. The aim is to discover what sensitive and valuable information is stored in the database and how to extract it. In practice, this tedious process is usually automated and often based on widely-known tools that let an attacker quickly and effortlessly identify and exploit applications’ vulnerabilities.

2.2 Analyzing the Attack Method: MSSQL Injection with Conversion Errors

Hackers often abuse a SQL injection (SQLi) vulnerability in a Web application resource to steal data from a database. A popular target within the database would be the tables that contain personally identifiable information (PII) (users, customers, patients, transactions) such as names, addresses, e-mail and passwords or even credit card details. The attack is facilitated by the injection of a SQL SELECT statement, which allows hackers to query the database for its content. However, even when the application is vulnerable to SQLi and executes an arbitrary SQL statement, it does not necessarily display the results back.

To overcome this obstacle, hackers are using the “MSSQL with conversion errors” SQLi variant. As a matter of fact, the “MSSQL with conversion errors” method is an old trick in the hackers’ book. In order to use that method, the following preconditions must exist:

› The application is vulnerable to SQL injection.
› The application is using an MSSQL database.
› The application server is misconfigured to send a verbose error message.

The attackers abuse the application server misconfiguration to invoke an error page that contains the desired data retrieved using SQLi.

In the Yahoo! case, the hacker used a conversion error to generate the error page. According to the screenshot, the hacker’s attack vector was "' \( \text{and } 1 = \text{convert (int, (select top 1 table name from x)} \) ".

The hacker tells the database to retrieve data about table names and to convert it to an integer. Since the returned data is a character string (nvarchar in DB terminology) and not an integer, a detailed error message is generated that contains the value of the character string that could not be converted – in this case, the table name (“product_section_Master_dir”). Displaying this error message to the user might be very helpful when the application is being developed and tested, but it is a bad idea to have it on production systems, as the result of hackers’ injected query is now sent back to them.

---

Figure 2 MSSQL with conversion error attack results

It’s important to note that the attacker does not have to be an experienced hacker in order to use this specific variant of SQLi with MSSQL errors. There are some free “point and click” automatic SQLi abuse tools that enable anyone who knows how to operate a Windows application to do just that.

One such tool is the very popular Iranian tool, Havij, which has an explicit support for extracting data from MSSQL Server using the error-based methods.°

---

° The text is url decoded for readability. Text in red is a replacement for text blackened in the original hacker screenshot.
° http://blog.imperva.com/2012/03/havij-101.html
2.3 Leveraging MSSQL SQL Injection Vulnerability to Command Execution

The attacker claimed the SQL injection led to having full access on the server. This was probably done by using MSSQL's "XP_CMDSHELL" system-stored procedure. Many administrative activities in MSSQL DB can be performed through system stored procedures. The XP_CMDSHELL executes a given command string as an operating-system command shell and returns any output as rows of text. Therefore, a SQL injection vulnerability in an application using MSSQL DB enables the hacker to execute shell commands and take over the server.

In order to exploit it, the hacker only needs to modify the aforementioned injected SQL code from the SELECT statement used to extract data to the EXEC statement used to execute stored procedure that will execute the "XP_CMDSHELL" system stored procedure.

So instead of "' and 1 = convert (int,(select top 1 table name from x))" the attack vector will be something like "; EXEC xp_cmdshell 'some command'."

Once more, exploiting MSSQL SQLi vulnerability for command execution is supported in automatic SQLi tools such as Havij, which means a vulnerability can be exploited relatively easily.

3. Protecting Third-Party Code

3.1 Identifying the Vulnerable Application

Analyzing the screenshot above, appearing in the previous section, we can find certain clues to help us reveal the nature of a vulnerable site:

- Host name from address bar: Although blackened by the attacker, some of the host's domain name is visible and we can determine two of its features:
  - It ends in "yle.yahoo.net". Although Yahoo! hosts many applications, almost all of them are hosted under the yahoo.com domain name.
  - It has a relatively long host name.
- The application is powered by ASP.NET as can be determined by the distinct error message and not by PHP as do most of Yahoo! applications, which further shortens the list of the possibly vulnerable Yahoo! applications.
- The error message reveals that the application source file resides on "C:\webcorp\[blackened by hacker]p\YahooV2\app_code."

---

Using all these hints with some Google search has led to a single candidate for that exploited application: “in.horoscopes.lifestyle.yahoo.net” an Indian astrology Web application.

- Host name is relatively long and ends with “yle.yahoo.net”
- Examining the HTTP headers reveals that the astrology application is powered by the ASP.NET technology
- Trying to directly access the “app_code” directory on that server (“forceful browsing”) yields the following error message:
This error message tells us:

- The “app_code” directory exists on the server, although we are not allowed to view its content as it resides on a “hidden segment.”
- The physical path of this directory (C:\webcorp\astroyogi.com_new\astroyogi_revamp\YahooV2\app_code) conforms with our hint on the source file location.

Although we cannot be absolutely sure that this is indeed the application reported as hacked by the hackers, in the face of such evidence, we feel confident to assume with a great deal of certainty that it is.

3.2 Understanding the Relationship Between Yahoo! and AstroYogi.com

As the clues suggested, the vulnerable application was not developed by Yahoo! programmers, but by AstroYogi.com developers. AstroYogi.com is, as stated on its Website, “the leading astrology portal in India...formed co-branded channel alliances with internationally recognized brands such as MSN, Yahoo! and Google amongst others.”

![Figure 4 AstroYogi.com about page](http://www.astroyogi.com/aboutus.aspx)

In fact, not only that the code was not developed by Yahoo! programmers, the application itself is not even hosted on Yahoo! servers, but on the Indian Website servers.

![Figure 5 DNS Query results for in.horoscopes.lifestyle.yahoo.net](http://www.astroyogi.com/aboutus.aspx)

The routing of users from Yahoo! to AstroYogi.com is achieved by using a DNS alias. When the user wants to browse “in.horoscopes.lifestyle.yahoo.net” a DNS query is sent. When a DNS server looks up the application name on yahoo.net records and finds it is actually an alias, it replaces the name with the canonical name (in this case “yahoo.astroyogi.com”) and looks up the new name.

---

3.3 Protecting Third-Party Code

This is not the first time Yahoo! has been struggling with security issues on third-party code. Last July, a decommissioned part of Yahoo! Voices was breached, and approximately 450,000 users’ credentials were exposed.\(^8\) According to the hackers, the breach was enabled by a SQL injection vulnerability (union-based SQLi). Yahoo! Voices is an online publishing application that was developed by Associated Content and later acquired by Yahoo!.\(^9\)

The problem of third-party code is not limited to Yahoo! of course. Almost every Web application includes some components that were not developed by the application programmers. Even when the application is completely home brewed, surely its Web server and operating system are coded elsewhere.

The Payment Card Industry Data Security Standard (PCI DSS) Requirement 6.6 provides two options for Web applications protection.\(^10\) The first is to conduct a vulnerability assessment and incorporate the assessments into the software development life cycle (SDLC). The other is to deploy a Web Application Firewall (WAF) in front of the Web application.

Naturally, where all the options are available, the best protection is achieved by combining all of them together. However, with third party code, the ability to incorporate the assessments into the software development life cycle (SDLC), or simply put fixing the code, is virtually nonexistent. Therefore, the only viable way to protect third-party code is by putting it behind a WAF.

In this case of the third party astrology application, Yahoo! could have directed user traffic to AstroYogi.com not via DNS alias, but with a WAF, deployed on Yahoo! environment or on the cloud as a reverse proxy and shield the application. That way, the application would have been protected from the hacking, and Yahoo! would have spared the bad PR and the possible abuse of its users' privacy.

\(^8\) [http://www.bbc.co.uk/news/technology-18811300](http://www.bbc.co.uk/news/technology-18811300)
\(^10\) [https://www.pcisecuritystandards.org/documents/information_supplement_6.6.pdf](https://www.pcisecuritystandards.org/documents/information_supplement_6.6.pdf)
4. Summary and Conclusions

SQL injection is still very relevant – even after a decade of Web application development and Web application security awareness, SQL injection vulnerability exploitation is still very relevant and continues to take a major role in Web application hacking. A successful SQLi attack may lead to sensitive data disclosure, sensitive data manipulation, and full server takeover. Whether you are outsourcing development, services, or maintenance, the bottom line is that if you are allowing others to create code and run services that your customers will perceive as coming from you – meaning that you are responsible for any functional problems or security breaches.

To mitigate SQL injection and other Web attacks from third party code, application owners and security officers should read our guide on stopping SQL injection: [http://blog.imperva.com/2012/01/sql-injection.html](http://blog.imperva.com/2012/01/sql-injection.html).

When it comes to third-party code, protecting applications with a Web Application Firewall is essential. With third-party code, you are not able to fix the code and WAF is the only relevant protection option.

From a business standpoint, executives should always assume third-party code – coming from partners, vendors, mergers and acquisitions – is vulnerable. Appropriate legal and technical precautions should be applied when engaging in activities such as partnerships or acquisitions.

Hacker Intelligence Initiative Overview

The Imperva Hacker Intelligence Initiative goes inside the cyber-underground and provides analysis of the trending hacking techniques and interesting attack campaigns from the past month. A part of Imperva’s Application Defense Center research arm, the Hacker Intelligence Initiative (HII), is focused on tracking the latest trends in attacks, Web application security and cyber-crime business models with the goal of improving security controls and risk management processes.