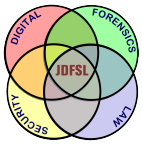

THE JOURNAL OF
DIGITAL FORENSICS, SECURITY AND LAW

Journal of Digital Forensics, Security and Law

Volume 12 | Number 2 Article 10

6-30-2017

SQL Injection: The Longest Running Sequel in Programming History

Matthew Horner
Norwich University, mhorner@norwich.edu

Thomas Hyslip
Norwich University, thyslip@norwich.edu

SQL INJECTION: THE LONGEST RUNNING SEQUEL IN PROGRAMMING HISTORY

Matthew Horner
Norwich University
Northfield, VT
mhorner@norwich.edu

Thomas Hyslip
Norwich University
Northfield, VT
thyslip@norwich.edu

ABSTRACT

One of the risks to a company operating a public-facing website with a Structure Query Language (SQL) database is an attacker exploiting the SQL injection vulnerability. An attacker can cause an SQL database to perform actions that the developer did not intend like revealing, modifying, or deleting sensitive data. This can cause a loss of confidentiality, integrity, and availability of information in a company's database, and it can lead to severe costs of up to $196,000 per successful injection attack (NTT Group, 2014). This paper discusses the history of the SQL injection vulnerability, focusing on:

- How an attacker can exploit the SQL injection vulnerability
- When the SQL injection attack first appeared
- How the attack has changed over the years
- Current techniques to defend adequately against the attack

The SQL injection vulnerability has been known for over seventeen (17) years, and the countermeasures are relatively simple compared to countermeasures for other threats like malware and viruses. The focus on security-minded programming can help prevent a successful SQL injection attack and avoid loss of competitive edge, regulatory fines and loss of reputation among an organization's customers.

Keywords: SQL, SQL Injection, Cybercrime, Intrusion, Database

1. INTRODUCTION

The Internet brings humans closer together than ever before, and in order to take advantage of the increased connectivity to customers, many organizations maintain a link to the Internet. However, with that link, organizations take on many risks because of the increased attack surface, but there are ways to mitigate those risks to an acceptable level with administrative, physical, and technical controls. Ultimately, it is the business

leader's or authorizing officials' responsibility to decide whether the benefits outweigh the potential negative effects of implementing a technology, but information security professionals can add more confidence behind that decision by having a thorough understanding of the threats and vulnerabilities to information systems (NIST, 2010).

One of the risks from a web server connected to the Internet is an attacker exploiting an SQL injection vulnerability on an organization's website. In fact, the Open Web Application Security Project (OWASP) consistently lists injection as the top website vulnerability while stating that it is "EXTREMELY simple" to prevent (OWASP, 2013, 2016). A vulnerability that is simple to fix yet continues to plague website designers begs the question, "Why haven't website programmers eliminated this vulnerability entirely?" A discussion of the history of the SQL injection vulnerability may shed light on how the vulnerability reached its current state and may offer clues as to why it refuses to go away. This paper discusses the history of SQL injection vulnerability, focusing on:

- How an attacker can exploit the SQL injection vulnerability
- When the SQL injection attack first appeared
- How the attack has changed over the years
- Current techniques to defend adequately against the attack

2. HOW AN ATTACKER CAN EXPLOIT THE SQL INJECTION VULNERABILITY

In order to understand the history of the SQL injection attack, it may help to understand how the attack works. In general, the principle behind the SQL injection attack is to take advantage of a poorly-coded website to transmit commands directly to a database, gain access to that database, and then perform the desired operation like copying, modifying, or deleting data (McDonald, 2002). To conduct the attack, a malicious user types SQL coding language into data entry fields on websites.

A sample injection attack from the OWASP is shown below to briefly describe one case of how an attacker can manipulate SQL coding. Colors are used to highlight where discussed concepts appear in the coding language. In this example, a website application uses typed data from an untrusted external user to construct the following vulnerable SQL request for information:

```
String query= "SELECT*FROM accounts
    WHERE custID='" +
    request.getParameter("id") + "'";
```

The attacker can modify the 'id' parameter value in the browser field to send ' or '1'='1; this could be done by typing a website address like:

```
http://example.com/app/accountView?id=' or
    '1'='1
```

In this case, the entry changes the meaning of the query to return all the records from the accounts table which can lead to unauthorized disclosure of confidential or private information (OWASP, 2013).

Additionally, instead of performing the injection process manually, attackers have designed computer programs to complete the process automatically. Examples of these programs include *BSQL Hacker, SQLmap, SQLninja*, and others (Shankdhar, 2015). To use these programs, the attacker inputs a website address; the program then searches for variations in the website address and returns the data associated with those different addresses (Cox, 2015). If an organization stores

a file of social security numbers in the same database as news articles, for example, these tools could return the social security number file even though that file was never meant to be available to the public.

These programs enable someone with a very low skill level to conduct these attacks, so the number of possible threats is very high. Essentially, anyone with a computer, an Internet connection, and intent could conduct an SQL injection attack and retrieve private or sensitive information (OWASP, 2016). Attackers could also insert, modify, and delete data in an organization's database using this vulnerability (OWASP, 2016). In some cases, attackers can perform actions normally restricted to administrators like shutting down the database management system (OWASP, 2016).

In addition to the relative simplicity of the attack, widespread SQL injection vulnerabilities and the perceived value of data in SQL databases help make SQL injection attacks common. According the 2016 NTT Group Global Threat Intelligence Report, injection attacks composed about a quarter of all attacks on website applications in 2015 (NTT Group, 2016). Additionally, per the NTT Group's 2014 report, each successful SQL injection attack can cost organizations up to $196,000 (NTT Group, 2014).

3. WHEN THE SQL INJECTION ATTACK FIRST APPEARED

The SQL injection vulnerability has been known for seventeen (17) years, but it continues to plague security professionals to this day. The vulnerability was first documented by "rain.forest.puppy" in the December 1998 issue of Phrack magazine which described a Microsoft SQL server yielding possibly sensitive data through the use of commands in normal user inputs like "name" or "phone number" (rain.forest.puppy, 1998). The author of the issue with the pseudonym "rain.forest.puppy" is Jeff Forristal, a well-respected security expert (Forristal, 2016).

Even though it was first documented in 1998, SQL injection did not appear to garner much attention in the information security community until 2002. The reason for the sudden interest in a four-year-old vulnerability may have been due to the timing of national events and the appearance of devastating viruses and worms. For example, the US had recently been attacked by terrorists using hijacked planes on September 11, 2001. Shortly thereafter, the nation's academics, military personnel, and politicians focused on increasing physical security and cybersecurity (Poeter, 2011). The new attention to cybersecurity may have prompted a critical examination of vulnerabilities that could weaken the US government or infrastructure, and SQL injection may have been highlighted as a particularly risky weakness to websites connected to database systems.

In addition to the terrorist attack on the U.S. homeland, several viruses and worms had spread across the Internet causing various amounts of damage prior to 2002.

- In March 1999, the *Melissa* macro virus infected PCs with Microsoft Word and Outlook, and it was estimated to cost millions of dollars in lost productivity (Lewis, 1999).
- The *ILoveYou* virus followed in May 2000, showcasing the power of social engineering by infecting about 45 million Windows PCs using an enticing attachment (Ward, 2010).
- The *Anna Kournikova* virus began spreading across computers in February of 2001 (Wood, 2011).

- The *Code Red* worm appeared in July 2001 exploiting a vulnerability in Microsoft's Internet Information Server (IIS) (Tham, 2001).
- The *Nimda* worm followed in September 2001 exploiting multiple vulnerabilities in Microsoft systems (Poore, 2001).

In fact, in 2009, *The Telegraph* ranked three of these viruses and worms in their top 10 list of worst computer viruses of all time (The Telegraph, 2009). The alarming appearance of rapidly-propagating viruses and worms in a short period of time may have shifted the attention of computer experts to information security.

The increased attention on SQL injection in 2002 appears to reflect the rising interest in cybersecurity at the time. Noting that the viruses and worms discussed above targeted Windows systems, Bill Gates released a memo earlier that year describing the new direction of Microsoft toward Trustworthy Computing, an initiative to improve the security of computer technology and software produced by the company (Gates, 2002). Since then, numerous academic papers have been written to address not only SQL injection attacks, but other well-known attacks as well.

In 2003, Hunag, Huang, Lin, and Tsai developed a platform for assessing web application security. The authors realized rapid development of web applications was resulting in poor coding, which introduced vulnerabilities in the applications. Their project named Web Application Vulnerability and Error Scanner (WAVES) was one of the first attempts to assess web applications for vulnerabilities, including SQL injection.

Halfond and Orso released their tool named Amensia in 2005, and it is often cited today as one of the first tools focused exclusively on SQL injection detection and prevention. AMNESIA stands for Analysis and Monitoring for Neutralizing SQL Injection Attacks, and it combines static analysis and runtime monitoring (Halfond & Orso, 2005). Amensia builds a table of expected SQL queries during the static analysis and then during the runtime monitoring, all dynamic queries are checked against the table to identify any unauthorized attempts (Halfond & Orso, 2005).

In 2008, Kemalis and Tzouramanis developed an SQL injection detection system (SQL-IDS) to monitor Java based web applications and detect SQL injection attacks in real time. SQL-IDS is a methodology based detection system that does query specific detection (Kemalis & Tzouramanis, 2008). This approach is very efficient and did not produce any false positives or false negatives in testing (Kemalis & Tzouramanis, 2008).

Shar and Tan (2013) presented a three-pronged approach to defeating SQL injection. They believed a combination of defensive coding and vulnerability detection, as well as runtime attack prevention was necessary to defeat SQL injection (Shar & Tan, 2013). The authors considered defensive coding the first line of defense. By replacing dynamic queries with stored procedures attackers would be unable to inject additional code (Shar & Tan, 2013). If dynamic queries must be included, then the authors recommend data validation and escaping. In addition to defensive coding, Shar and Tan (2013) recommended vulnerability detection and runtime attack prevention. They provide a review of numerous methods to detect SQL injections and tools to prevent SQL injections through runtime analysis (Shar & Tan, 2013).

In 2015, Alghamdi, Ahmad, and Imran offered a new technique to prevent SQL injection attacks. By using application layer detection at both the client and server, they are able to prevent SQL injection attacks. On the client side, SQL inputs are checked for

special characters used in typical SQL injections, and only filtered input strings are passed to the server (Alghamdi, et al., 2015). The server ensures all passed requests have the proper rights and permissions to access the web applications. Used together, these techniques significantly reduce SQL injection attacks (Alghamdi, et al., 2015).

4. HOW THE ATTACK HAS CHANGED

Like much of computer technology, SQL injection attacks rarely stay constant. The December 1998 issue of Phrack magazine discussed how an attacker could piggyback SQL commands in user inputs (rain.forest.puppy, 1998). While it is certainly an effective method to allow unauthorized disclosure of information stored in an SQL database, there are other types of attacks that can be used to gain unauthorized access to valuable data.

- By 2006, Halfond, Viegas, and Orso had described seven types of SQL injection attacks, distinguishing between the mechanisms of injection and the intent of the attacker (Halfond, Viegas, & Orso, 2006). While not attacks themselves, the mechanisms of injection focused more on how the attacker could inject SQL commands; for example, an attacker could manipulate user fields, cookies, server variables like HTTP headers, or use second-order injection to deliver commands (Halfond, Viegas, & Orso, 2006). The following types of attacks differ in the underlying vulnerability rather than the mechanism of injection.
- Tautology-based attacks involve inserting code into conditional statements so that the conditional statements always return a true value (Halfond, Viegas, & Orso, 2006). The aforementioned SQL injection example in this paper is a tautology-based attack because inserting the conditional statement ' or '1'='1 causes the query to become a conditional statement. Since one will always equal one, the database will return all rows from the accounts table as it executes the call for information.
- Instead of causing unauthorized disclosure of data stored in the database, illegal/logically incorrect queries function as more of a reconnaissance role (Halfond, Viegas, & Orso, 2006). In this attack, overly descriptive error codes can yield unintended information about the database such as table names. The attacker can then target specific parts of the database that may contain valuable or sensitive data.
- A union-query attack can cause a database to return an unintended table to the attacker with a command structured as "UNION SELECT <rest of injected query>" (Halfond, Viegas, & Orso, 2006).
- Piggybacking is the same attack discussed in Phrack magazine in 1998 where SQL commands are placed in user input fields (Halfond, Viegas, & Orso, 2006).
- Stored procedures, despite being advertised as the definitive shield against injection attacks, can become a method for injection attacks if the stored procedures contain vulnerabilities themselves (Halfond, Viegas, & Orso, 2006). Stored procedures are discussed in the countermeasures section of this paper, but as with many tools used by humans, if not implemented properly, they may become useless.

- Inference attacks are similar to illegal or logically incorrect queries in that the attacker is not directly accessing data in the SQL databases. Instead of relying on overly descriptive error messages, however; inference relies on the behavior of the database after receiving commands such as whether an error message appears at all (Halfond, Viegas, & Orso, 2006). Inference is further divided into blind injection and timing attacks (Halfond, Viegas, & Orso, 2006).
- Finally, alternate encoding involves masking commands by using hexadecimal, ASCII, or Unicode encoding. By itself this will not return valuable information, so it is combined with other types of attacks to assist in evading detection (Halfond, Viegas, & Orso, 2006). The existence of alternate encoding is a significant argument against the use of blacklisting as an effective countermeasure (Cisco, 2016).

In 2013, Kindy and Pathan composed a condensed list of only the two most common SQL injection attacks, tautology-based and inference attacks (Kindy & Pathan, 2013). However, the list goes into greater depth about the subtypes of each attack. Tautology subtypes are string injection, numerical injection,—and comments attack (Kindy & Pathan, 2013). Inference subtypes are blind injection, timing attacks, database backdoors, and command injection (Kindy & Pathan, 2013). Blind injection and timing attacks were discussed by Halfond, Viegas, and Orso in 2006 (Halfond, Viegas, & Orso, 2006). While adding a few new attack methods, the types of SQL injection attacks remain approximately the same as those discussed in 2006.

Attackers have used some form of SQL injection to deface public websites, install malware, or obtain sensitive information like social security numbers or credit card details; any of these consequences are expensive and embarrassing to the victim. In particular, one string of high-profile attacks took place over a seven-year period affecting popular organizations like 7-Eleven, JCPenney Inc., NASDAQ, Dow Jones Inc., and JetBlue Airways (Kitten, 2013). The attackers most often used SQL injection to gain initial access to a targeted system eventually collecting over 160 million credit card numbers and related personal information from 2005 to 2012 (Department of Justice, 2013). The cost to the 16 affected organizations totaled in the hundreds of millions of dollars, and three corporate victims alone had combined losses of over $300 million (Department of Justice, 2013). This does not include the indirect costs of identity theft of the individual consumers whose personal information was stolen. This streak of attacks suggests that some attackers are not content with gaining unauthorized access to personal information from a single event; if successful, attackers may continue to use the same exploits to attack multiple organizations.

In 2011, Sony was attacked multiple times through the Sony PlayStation Network, Sony Music Japan, and Sony Pictures. While it is difficult to confirm that the attackers of the PlayStation Network used an SQL injection vulnerability, the attackers of Sony Music Japan and Sony Pictures certainly did (Anthony, 2011; Wisniewski, 2011; Henderson, 2011). The attack on the PlayStation Network compromised the personal information of 100 million users and began a month-long outage estimated to cost $171 million (Henderson, 2011). While the attack on Sony Music did not result in a breach of sensitive information, the Sony Pictures attack resulted in the disclosure

of personal information of one million users (Wisniewski, 2011; Henderson, 2011). Sony's eventful year highlights the need to remove known vulnerabilities and apply lessons learned to all areas of an organization aggressively. Otherwise, organizations can expect to continue to lose money from known and preventable security issues.

5. COUNTERMEASURES AVAILABLE TO PROTECT AGAINST THE THREAT

Luckily, the defense against an SQL injection attack can be relatively simple, and most of the solutions involve treating data entered by users as untrusted or hostile (McDonald, 2002). Some countermeasures against an SQL injection attack include:

- Whitelisting
- Prepared statements
- Stored procedures

Whitelisting refers to accepting data in an input field only if it contains predefined, permitted characters (Cisco, 2016). For example, if a user input field was associated with entering numbers like whole dollar amounts, whitelisting would ensure that entries with numeric characters only would be accepted. Any entries with letters or special characters would be rejected. The inverse of whitelisting is blacklisting, which rejects entries with characters known to facilitate SQL injection attacks. Whitelisting is an "implicit deny" concept where all entries are rejected except those explicitly allowed by the programmer, and blacklisting is an "explicit deny" concept where all entries are accepted except those explicitly denied. For SQL injection attacks, whitelisting is more effective than blacklisting because attackers can use sophisticated techniques like alternate encoding to hide blacklisted characters in search fields nullifying the effect of the blacklisting feature (Cisco, 2016).

Prepared statements, also known as parameterized queries, focus on how programmers structure data requests based on what the user types in the input fields (OWASP, 2016). If done properly, a database will interpret the user input as it is exactly written and not as a command. For example, if a programmer used prepared statements in the SQL coding in the above example and a malicious attacker entered ' or '1'='1, the SQL server would search for an account with the exact name ' or '1'='1 instead of returning all data from the accounts table, preventing unauthorized disclosure of information (OWASP, 2016).

Stored procedures are very similar to prepared statements, and they have the same security benefits as prepared statements (OWASP, 2016). With stored procedures, however, the SQL code is kept within the SQL database (OWASP, 2016).

Other defenses exist to help make SQL injection attacks less successful. For example, keeping table names less predictable, like changing the table name "accounts" to "acc_pay_mon," helps make it more difficult for attackers to guess successful commands (McDonald, 2002). Additionally, an intrusion prevention system (IPS) can detect the signatures of an SQL injection attack and take automatic action to thwart the attack (Cisco, 2016).

To help reduce the risk of a successful SQL injection attack, security professionals can train an organization's website application developers to write code with security techniques like whitelisting, prepared statements, and stored procedures. While developers are focused on what their programs

should do, it is equally important to determine what their programs *could* do from a security standpoint, so code reviews and dynamic testing of website applications should be performed to help identify SQL injection vulnerabilities. Until developers uniformly begin coding with security in mind, injection will likely remain the number one website vulnerability on the OWASP's Top 10. The additional time and effort to incorporate secure coding, static and dynamic testing, and an IPS can be far less than the cost of a successful SQL injection attack costing $196,000 or a string of attacks costing hundreds of millions of dollars. One hundred million credit and debit cards were stolen in the 2009 Heartland Payment Systems breach which was the result of SQL injection and it is estimated they paid out approximately $140 million in fines and other penalties (Lewis, 2015).

6. CONCLUSION

From its discovery in 1998 to the current forms of attack, SQL injection has been a thorn in the sides of website developers using associated SQL databases. However, the attack itself did not receive much attention until 2002 which may have been a function of the terrorist attacks of September 11, 2001 and the concentration of viruses and worms appearing in the years prior to 2002. Between 1998 and 2006, the types of documented SQL injection attacks expanded from one to seven, but today's most common attacks can be grouped into two major groups: tautology-based and inference attacks. Countermeasures against an SQL injection attack include whitelisting, prepared statements, stored procedures, and an IPS. Many of those countermeasures rely on a properly trained programmer with security in mind, and the cost of countermeasures can be far less than the cost of a successful injection attack and the subsequent cleanup. When defending against SQL injection attacks, "an ounce of prevention is worth a pound of cure."

REFERENCES

Alghamdi, A., Ahmad, B., & Imran, M. (November, 2015). SQL injection attack, still an unaddressed issue with dynamic web applications. *International Journal of Computer Science Engineering, 4*(6).

Anthony, S. (2011, April 27). *How the Playstation Network was hacked.* Retrieved October 16, 2016, from Extreme Tech website: http://www.extremetech.com/gaming/84218-how-the-playstation-network-was-hacked

Cisco. (2016, February 15). *Understanding SQL injection.* Retrieved July 19, 2016, from Cisco website: http://www.cisco.com/c/en/us/about/security-center/sql-injection.html#6

Cox, J. (2015, November 20). *The history of SQL injection, the hack that will never go away.* Retrieved July 17, 2016, from Motherboard website: http://motherboard.vice.com/read/the-history-of-sql-injection-the-hack-that-will-never-go-away

Department of Justice. (2013, July 25). *Five indicted in New Jersey for largest known data breach conspiracy.* Retrieved October 18, 2016, from Department of Justice website: https://www.justice.gov/usao-nj/pr/five-indicted-new-jersey-largest-known-data-breach-conspiracy

Forristal, J. (2016). *Jeff Forristal LinkedIn profile.* Retrieved August 29, 2016, from LinkedIn website: https://www.linkedin.com/in/jeffforristal

Gates, B. (2002, January 15). *Bill Gates: Trustworthy computing.* Retrieved August 30, 2016, from Wired.com website: http://www.wired.com/2002/01/bill-gates-trustworthy-computing/

Halfond, W & Orso, A. (2005). AMNESIA: Analysis and monitoring for NEutralizing SQL-Injection attacks. *Proceedings of the Automated Software Engineering Conference 2005*, Long Beach, CA. Retrieved from http://www-bcf.usc.edu/~halfond/papers/halfond05ase.pdf

Halfond, W., Viegas, J., & Orso, A. (2006). *A classification of SQL injection attacks and countermeasures.* Retrieved September 1, 2016, from Georgia Institute of Technology website: http://www.cc.gatech.edu/fac/Alex.Orso/papers/halfond.viegas.orso.ISSSE06.pdf

Henderson, N. (2011, June 3). *Hackers attack Sony Pictures with single SQL injection.* Retrieved October 18, 2016, from The Whir website: http://www.thewhir.com/web-hosting-news/hackers-attack-sony-pictures-with-single-sql-injection

Henderson, N. (2011, May 24). *Sony estimates $171M in losses from Playstation Network outage, more from earthquake.* Retrieved October 18, 2016, from The Whir website: http://www.thewhir.com/web-hosting-news/sony-estimates-171m-in-losses-from-playstation-network-outage-more-from-earthquake

Hunag, Y., Huang, S., Lin, T., & Tsai, C. (2003, May). Web application security assessment by faultinjection and behavior monitoring. *Proceedings of the 12th International Conference on World Wide Web*, Budapest, Hungary, 148-159.

Retrieved from http://dl.acm.org/citation.cfm?doid=775152.775174

Kemalis, K., & Tzouramanis, T. (2008, March). SQL-IDS: a specification-based approach for SQL-injection detection. Proceedings of the 2008 ACM Symposium on Applied Computing, MArch16-20, 2008. Fortaleza, Brazil. Retrieved from http://dl.acm.org/citation.cfm?doid=1363686.1364201

Kindy, D., & Pathan, A. (2013). A Detailed Survey on various aspects of SQL Injection in Web Applications; Vulnerabilities, Innovative Attacks and Remedies. *Internation Journal of Communication Networks and Information Security* , 80-92.

Kitten, T. (2013, July 26). *Card fraud scheme: The breached victims.* Retrieved October 18, 2016, from Bank Info Security website: http://www.bankinfosecurity.com/card-fraud-scheme-breached-victims-a-5941

Lewis, D. (2015, May). *Heartland payment systems suffers data breach.* Forbes.com. Retrieved from https://www.forbes.com/sites/davelewis/2015/05/31/heartland-payment-systems-suffers-data-breach/#7f5798a2744a

Lewis, P. (1999, April 1). *State of the art; Melissa and her cousins.* Retrieved August 30, 2016, from The New York Times website: http://www.nytimes.com/1999/04/01/technology/state-of-the-art-melissa-and-her-cousins.html

McDonald, S. (2002, April 8). *SQL injection: Modes of attack, defence, and why it matters.* Retrieved July 17, 2016, from SANS Institute: https://www.sans.org/reading-room/whitepapers/securecode/sql-injection-modes-attack-defence-matters-23

NIST. (2010, February). *NIST Special Publication 800-37 Guide for Applying the Risk Management Framework to Federal Information Systems Revision 1.* Retrieved August 29, 2016, from NIST website: http://csrc.nist.gov/publications/nistpubs/800-37-rev1/sp800-37-rev1-final.pdf

NTT Group. (2016). *2016 NTT Group Global Threat Intelligence Report.* NTT Group Security.

NTT Group. (2014). *NTT Group 2014 Global Threat Intelligence Report.* NTT Innovation Institute.

OWASP. (2013). *OWASP Top 10 - 2013: The ten most critical web application security risks.* OWASP.

OWASP. (2016, April 10). *SQL injection.* Retrieved July 17, 2016, from OWASP website: https://www.owasp.org/index.php/SQL_injection

OWASP. (2016, May 25). *SQL injection prevention cheat sheet.* Retrieved July 19, 2016, from OWASP website: https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

Poeter, D. (2011, September 8). *How cybersecurity has changed since 9/11.* Retrieved August 30, 2016, from PCMag website: http://www.pcmag.com/article2/0,2817,2392642,00.asp

Poore, K. (2001, November 11). *Nimda worm - Why is it different?* Retrieved August 31, 2016, from SANS website: https://www.sans.org/reading-room/whitepapers/malicious/nimda-worm-different-98 rain.forest.puppy. (1998, December 25). NT web technology vulnerabilities. *Phrack Magazine , 8* (54).

Shankdhar, P. (2015, April 28). *Best free and open source SQL injection tools*. Retrieved August 29, 2016, from Infosec Institute website: http://resources.infosecinstitute.com/best-free-and-open-source-sql-injection-tools/

Shar, L., & Tan, H. (2013, March). Defeating SQL injection. *Computer. 46*(3). Retrieved from https://www.computer.org/csdl/mags/co/2013/03/mco2013030069.pdf

Tham, A. (2001, August 4). *What is Code Red worm?* Retrieved August 31, 2016, from SANSwebsite: https://www.sans.org/reading-room/whitepapers/malicious/code-red-worm-45

The Telegraph. (2009, March 18). *Top 10 worst computer viruses.* Retrieved August 31, 2016, from The Telegraph website: http://www.telegraph.co.uk/technology/5012057/Top-10-worst-computer-viruses-of-all-time.html

Ward, M. (2010, May 4). *A decade on from the ILoveYou bug.* Retrieved August 30, 2016, from BBC website: http://www.bbc.com/news/10095957

Wisniewski, C. (2011, May 24). *Sony Music Japan hacked through SQL injection flaw.* Retrieved October 18, 2016, from Sophos website: https://nakedsecurity.sophos.com/2011/05/24/sony-music-japan-hacked-through-sql-injection-flaw/

Wood, P. (2011, February 10). *10th anniversary of the Anna Kournikova virus.* Retrieved August 31, 2016, from Symantec website: http://www.symantec.com/connect/blogs/10th-anniversary-anna-kournikova-virus

