Thesis Guidelines: An Analysis of Successful SQLIA for Future Evolutionary Prediction

Andrew Pechin

A Senior Thesis submitted in partial fulfillment
of the requirements for graduation
in the Honors Program
Liberty University
Spring 2023

Acceptance of Senior Honors Thesis

This Senior Honors Thesis is accepted in partial
fulfillment of the requirements for graduation from the
Honors Program of Liberty University.

_____

James McNicholas, Ph.D.
Thesis Chair

_____

Richard Bansley, M.S.
Committee Member

_____

Emily C. Knowles, D.B.A.
Assistant Honors Director

_____

Date

Abstract

Web applications are a fundamental component of the internet, many interact with backend databases. Securing web applications and their databases from hackers should be a top priority for cybersecurity researchers. Structured Query Language (SQL) injection attacks (SQLIA) constitute a significant threat to web applications. They can hijack the backend databases to steal personally identifiable information (PII), initiate scams, or launch more sophisticated cyberattacks. SQLIA has evolved since its conception in the early 2000s and will continue to do so in the coming years. This paper analyzes past literature and successful SQLIA from specific time periods to identify themes and methods used by security researchers and hackers. By extrapolating and interpreting the themes of both literature and effective SQLIA, trends can be identified, and a clearer understanding of the future of SQL injection can be defined to improve cybersecurity best practices.

An Analysis of Successful SQLIA for Future Evolutionary Prediction

Structured Query Language (SQL) is a standardized programming language designed to store or retrieve information from a database. Many web applications that house and reference substantial amounts of data leverage databases that use SQL. Some of the information stored can be sensitive and the revealing of that data to the wrong people could lead to a compromise of the web application's integrity. A SQL injection attack (SLQIA) is a cyber-attack type that leverages the input fields of user interfaces to launch malicious SQL code. This code can be tailored to retrieve sensitive information that can compromise the application. SQL injection attacks are mainly deployed against user input fields that are not sanitized or filtered. The resulting information gathered from the database is used by hackers to exploit a system further by unauthorized retrieval of data within the database, or to make the data within unusable or corrupted. SQL injection falls under the category of command injection attack and has been listed as one of the most prevalent attack types threatening cyberspace ("OWASP Top 10", 2021). With an increase in the reliance on web applications using database storage of relevant information for everyday tasks, it is important to further understand a major threat to them (Gupta et al., 2019). By reviewing the evolution of both literature regarding SQL injection and successful SQL injection attacks, a better understanding of this threat can be gained and attempts to predict the evolutionary course of this attack vector can be achieved.

Literature Review

Before successful SQL injection cyberattacks can be analyzed, relevant literature must be reviewed. By observing and analyzing relevant SQLIA literature from 2005 until 2021, key themes of SQL injection research and cybersecurity can be derived, and a better understanding of the context surrounding successful SQLIA can be obtained.

Basic date filtering was used to obtain an appropriate amount of literature for each five-year interval examined: 2005 to 2010, 2010 to 2015, and 2015 to 2021. While the literature review includes relatively old research (greater than half a decade old), the information they divulge is critical in understanding the development of SQLIA literature over the years. During each observed interval, distinct themes and methods can be extracted. From 2005 to 2010, security researchers focused primarily on defining SQL injection and providing basic but novel security countermeasures. The focus of SQLIA research shifted from 2010 to 2015, continuing to produce SQLIA countermeasures but also identifying the potential damage of SQL injection. Finally, 2015 to 2021 saw literature focus on the adaptability of SQL injection to new technologies and countermeasures.

Literature from 2005-2010

By 2005, SQL injection remained a novel cyber-attack type (Halfond et al., 2006). Literature regarding SQL injection was produced quickly. Halfond et al. analyzed contemporary SQL injection attacks to develop a classification system. By analyzing their mechanisms, attack intent, and the use of an example attack, the researchers categorized SQL injection attacks into seven types: tautologies, illegal queries, union queries, piggy-backed queries, stored procedures, inference, and alternate encodings. Halfond et al. demonstrated each proposed SQLIA type, solidifying the findings. This development of a SQL injection classification system helps researchers and professionals gain a greater understanding of this novel attack type.

Beyond classifying the types of SQL injection attacks, defining their relationship to other attacks is paramount. The first formal definition of command injection attacks was created in 2006 through the analysis of SQL injection (Su & Wasserman, 2006). Using a SQL injection attack, Su & Wasserman observed that an attack's success is directly related to how the syntax of

the query is changed. By changing the syntactical nature of it, they revealed that the query can function in an unintended way, such as providing confidential data. By developing research into defining the SQL injection family of cyberattacks, further information can be gathered to help mitigate this attack type.

By 2007, SQL injection remained one of the least understood types of web application cyberattacks (Moyle, 2007). During this time, SQL injection's definition and consequences are further defined by security professionals like Moyle. Furthermore, tighter security policies, such as zero-trust, are defined and suggested in place of current, overly trusting policies. Moyle described how existing web applications treat SQL queries as trusted commands, and hackers use this trust to piggy-back malicious queries on legitimate ones. Moyle presented an example of error-based SQL injection, demonstrating a hypothetical SQLIA. Moyle also elaborated on varying consequences of SQL injections, such as stolen PII, database shutdowns, online shoplifting, and scams. Aside from the financial impact, the long-term damage to an organization's reputation is emphasized as a key consequence of SQLIA. Moyle suggested various forms of static query analysis are suggested b as countermeasures for SQLIA.

Many websites are still developed in the Active Server Pages (ASP), PHP: Hypertext Preprocessor (PHP), and Java Server Pages (JSP) development frameworks (Liwu et al., 2009). These frameworks use programming language to dynamically generate HTML code, resulting in web pages that are tailored to the code designer. All of these web application designs use databases to dynamically display pages using the information in the backend database, according to Liwu et. al. Furthermore, these web application designs are also prime targets for SQLIA as it allows hackers to create their own pages with fake information, perpetuating malicious scams or malware. In response to this threat, Liwu et al. proposed a new database protection system. The

proposed database protection system lays between the web server and the database server, starkly contrasting with the more externally placed contemporary protection systems. Also, the system parsed network and database protocol packets in search of malicious packet properties. Furthermore, the parsed packets revealed the SQL statements being sent to the database server. Once extracted, the SQL statements were analyzed and filtered for malicious properties based on predefined rules. The work by Liwu et al. (2009) demonstrates a shift away from defining SQLIA and focusing more on preventing it.

The continued threat of SQLIA against web applications continued to prompt the development of research investigating mitigation and prevention techniques (Chen & Buford, 2009). Chen and Buford (2009) developed design considerations for a SQL injection honeypot for SQLIA prevention. By presenting the appearance of a vulnerable web application server, Chen and Buford proposed that honeypots can trick hackers into stealing false information that can be traced back to them. Their honeypot saw an increase in effectiveness since SQL injection typically is a multi-attempt cyberattack, giving ample opportunities to trick hackers. The proposed honeypot also monitored any changes in data, revealing if the SQLIA tried to produce malicious web pages on dynamically producing web architectures, like ASP, PHP, or JSP. Furthermore, their honeypot monitored and restricted stored procedures, a primary method of obtaining remote code execution in SQL injection cyberattacks. Basic defense mechanisms, like basic input validation and minimization of error messages, would also be included to make the honeypot less suspicious to attackers. The development of more creative defense solutions to SQL injection by Chen and Buford further demonstrates the shift in literature from SQLIA definition to SQLIA defense.

Literature from 2010-2015

Where literature shortly following SQL injection focused on defining the cyberattack, papers from 2010 to 2015, like Bono and Domangue's (2012) work, focused on countermeasures and the potential impacts of SQL injection. Their black box security evaluation of a web application server revealed the devastating effects of SQL injection. Bono and Domangue's database analysis through various SQL queries revealed that the server tested was vulnerable to SQLIA. Furthermore, they gained administrative access to the database, which allowed the testers to compromise the user data and retrieve countless pieces of Personally Identifiable Information (PII).

Moreover, other literature from this time demonstrated the extent of the threat landscape of SQL injection among web applications (Munadi et al., 2013). Munadi et al. tested a SQL injection security hole against multiple different web domains in the Aceh province of Indonesia. Their manual input SQL injection leveraged incorrect single quotation marks in the query to test for vulnerabilities. The change in syntax caused an error message to appear which is used to further the attack process. Of the 100 websites tested, Munadi et al. discovered that 16 were found to be vulnerable to SQL injection. The work of Munadi et al. demonstrates the potential for vast amounts of security breaches using SQL injection.

With regards to the domains in which SQL injection is effective, literature detailing the range of the attack has been developed (Alam et al., 2015). Alam et al. investigated SQL injection efficacy concerning differing domains. With regards to the national Bangladesh top level domain (TLD) .bd domain, black box penetration testing by Alam et al. involving user input-based SQL injection revealed that a large proportion of web applications in this domain are vulnerable. Of the 900 examined, 600 were vulnerable. Furthermore, Alam et al. retrieved

various PII from vulnerable websites using an outdated version of PHP. Their examination of the effectiveness of SQL injection against differing domains helps define the attack potential of SQL injection against web applications.

Literature from 2015-2021

SQL injection has remained a top threat to web applications (Sonakshi et al., 2016). However, Sonakshi et al.'s work noted that while technology has changed, SQL injection has evolved with it. Sonakshi et al. made a clear and novel distinction between SQL injection types and tested each type on a local server. Also, SQL injection of live websites is leveraged to demonstrate URL attacks and highlights the main attack types against web applications as well as their efficacy. After the attack demonstration, Sonakshi et al. presented guidelines to prevent these types of attacks. The reiteration and demonstration of SQL injection attack types by Sonakshi et al. (2016) helps aid in the understanding of the contemporary threat landscape.

NoSQL is a novel database system that is more scalable and user-friendly than traditional SQL databases, leading to their rising popularity in the early 2010s (Ron et al., 2016). Instead of storing data in relational databases, which stores data based on its relationship with other stored data in a tabular, column and row format, Ron et al. noted that NoSQL databases use data structures like document databases, key-value stores, wide-column databases, and graph databases. Ron et al. noted that another key difference between traditional SQL and NoSQL databases is the query language itself. NoSQL uses different query languages to operate, such as node.js in graph databases. With the change in language, traditional SQL injection attack methods are irrelevant, according to Ron et al.. However, the researchers noted that some of the concepts from SQL injection still apply to NoSQL, and new attack methods have arisen.

Ron et al. (2016) demonstrated that tautologies, which inject code into conditional statements that generate always true expressions, are shown to still be effective against NoSQL applications. Also, illegal data extraction is noted to still be viable in union queries; a SQL injection technique that exploits illegal Boolean logic expressions. Piggy-backed queries are also still viable, which involve hackers attaching a malicious query to a legitimate query. Hackers have also developed new injection attack methods specifically for NoSQL databases, which Ron et al. elaborated on. They noted that NoSQL database querying allows the use of JavaScript execution for advanced transactions. Also, JavaScript injections are a new form of NoSQL injection that leverages the JavaScript execution properties of NoSQL database querying to pass in malicious JavaScript code. Furthermore, many NoSQL databases expose a Hypertext Transport Protocol (HTTP) Representational State Transfer (REST) Application Programming Interface (API), which can be used by hackers to launch cross-origin violation, another form of NoSQL injection. Ron et al. demonstrated how SQLIA continues to evolve in response to the development of new database technologies. The emergence of NoSQL injection identified by Ron et al. reveals the ability of SQLIA to transcend technological barriers and achieve data compromise.

Although a new service, cloud computing's ability to host various technologies as a service is critical to most business infrastructures (Fu et al., 2019). Fu et al. investigated the viability of SQL injection against cloud environments. By attempting SQL injection in Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS), Fu et al. revealed that these cloud services could not provide extra security against SQL injection and that even newer technologies like the cloud are still vulnerable to SQL injection. Fu et al.'s evidence

of its success against newer technologies, such as the cloud environment, reinforced the efficacy and extent of SQL injection against web application vulnerabilities.

Though declining in frequency, SQLIA continued to present a threat to web applications that use databases, including NoSQL databases (Sakib, 2021). Technology regarding the detection and prevention of SQL injection has improved, resulting in Sakib's observed decline in attack frequency. The techniques created by Sakib can be classified into two categories: machine learning (ML) and non-machine learning approaches. The proposed ML approach models SQL injection detection as a data mining-based binary classification problem, the goal being able to classify a query as malicious or otherwise. Sakib reported that Non-ML approaches diverge into either static analysis (examining system source code), dynamic analysis (executing and testing the query at runtime), or a combination of the two.

NoSQL detection and prevention techniques both parallel and differ from that of SQL injection (Sakib, 2021). According to Sakib, various ML approaches can detect NoSQL injection, such as feature-based supervised learning. In this approach, Sakib compiled features of malicious NoSQL queries for binary classification and accurately identified possible NoSQL attacks. In contrast to SQLIA detection and prevention, Sakib reported that non-ML approaches for detecting and preventing NoSQL injection are mainly static in nature. NoSQL detection and prevention approaches included the use of automata, user input validation, parameterization, and malicious feature detection, all of which are used to enforce proper code hygiene. A proper understanding of the current detection and prevention methodologies for both SQL and NoSQL injection provided by Sakib not only identifies the characteristics of contemporary attack methods but also reveals how hackers might change their injection strategies to circumvent these measures.

Successful Instances of SQLIA

The literature collected from each five-year interval helps establish both the current state of SQLIA understanding and cybersecurity posture. With cybersecurity literature in mind, successful SQLIA can be examined in the context of the time period they occurred in. This context provides a deeper understanding of how SQL injection evolved alongside the understanding and security posture held by security researchers.

Attacks from 2005-2010

As early as 2006, SQLIA was plaguing many websites that rely on SQL databases ("Spike in SQL injection," 2006). SecureWorks Inc., an internet security company providing service to 1,300 corporations, reported that consumer databases faced an onslaught of SQL injection attacks ("Spike in SQL injection," 2006). SecureWorks Inc. reported upwards of 8,000 SQLIA per day ("Spike in SQL injection," 2006). The frequency of SQLIA continued to rise since the beginning of 2006, with Secure Works Inc. recording the average number of SQL injection exploits increasing from 100 to 200 per day in as little as three months ("Spike in SQL injection," 2006). Although the frequency was high, the SQLIA were not copies of each other, but were designed purposefully with the target organization in mind ("Spike in SQL injection," 2006).

SQLIA consistently gathers sensitive information from vulnerable databases (Nogod, 2016). Nogod (2016) reported that an attack in the late 2000s made it evident that SQLIA can be used to commit non-cyber-related crimes, such as credit and debit card fraud. In 2010, hacker Albert Gonzales was convicted and sentenced to 20 years in federal prison and a fine of $25,000. Nogod (2016) reported that the sentencing came after Gonzales and his Russian hacker counterparts'

two-year hacking spree. Nogod (2016) confirmed that the group used SQL injection techniques to steal over 90 million credit and debit card numbers from major retailers.

To gain access to more significant amounts of private information, SQLIA evolved to increase its impact area ("SQL Injection Attack," 2008). In 2008, a large-scale, JavaScript-based SQLIA infected an estimated 500,000 websites ("SQL Injection Attack," 2008). Though initially thought to have exploited vulnerabilities in Microsoft's SQL server and Internet Information Server software, the infected web applications were poorly coded ("SQL Injection Attack," 2008). User input fields failed to sanitize input data, allowing hackers to launch malicious queries ("SQL Injection Attack," 2008).

In early 2007, hackers hijacked a major sports event website using SQL injection to distribute malware (Ullrich & Lam, 2008). According to Ullrich and Lam (2008), the infected website leveraged a development process where new pages could be created from database content on the fly. Also, by exploiting a database variable that was not sufficiently validated before being used in a SQL statement, the hackers were able to bypass security and compromise the database. The hackers then generated fake iFrames linked to malware-based exploits. Unsuspecting visitors downloaded the resulting malware through a Vector Markup Language (VML) exploit in Internet Explorer browsers. Ullrich and Lam noted that the malware recorded the victims' keystrokes, allowing password extraction.

Hackers further pushed the impact zone of SQL injection by leveraging scalability tools ("Botnet installs SQL injection," 2008). In the same year as the JavaScript-based attack, a botnet supplied its infected machines with SQL injection tools ("Botnet installs SQL injection," 2008). The botnet, dubbed "Asprox," leverages a malicious executable that searches Google for websites vulnerable to SQL injection ("Botnet installs SQL injection," 2008). Upon successfully

injecting a malicious SQL query, the bot modified the web page with a malicious iFrame designed to trick visitors into downloading a malicious JavaScript file ("Botnet installs SQL injection," 2008). When executed, the downloaded JavaScript file redirects the victim's machine to a site with more malicious scripts designed to replicate the Asprox botnet ("Botnet installs SQL injection," 2008). The Asprox-based SQLIA infected over 1,000 websites, expanding the botnet past its initial 15,000 host count ("Botnet installs SQL injection," 2008).

In 2009, less than a year following the Asprox botnet, a similar attack compromised more than 210,000 websites ("SQL injection," 2009). SQL injection was the primary cyberattack vector ("SQL injection," 2009). A malicious query then deployed an iFrame onto the website, similar to the Asprox botnet attack ("SQL injection," 2009). Once compromised, the website propagated malware designed to steal user data and create backdoors in infected machines ("SQL injection," 2009).Victims that travel to the infected website would have the malware downloaded onto their devices via the malicious iFrame, leading to the extraction of user data and the potential creation of a botnet ("SQL injection," 2009).

Attacks during the late 2000's involved removing the confidentiality of personal information,(Vijayan & Gaudin, 2009). According to Vijayan and Gaudin (2009), Hackers using SQL injection techniques stole 130 million credit and debit card numbers from multiple retailers, most notably 7-Eleven Incorporated. The hackers used poorly coded web application software to infiltrate the corporate systems and install packet-sniffing tools. Vijayan and Gaudin (2009) state that the simplicity of these attacks indicates the poor security used by the retailers, primarily caused by the use of older versions of Microsoft SQL server.

Stealing usernames and passwords was a primary objective of SQLIA ("Hacker hits British Navy," 2010). In 2010, hackers compromised the website of Britain's Royal Navy by a

SQL injection attack (“Hacker hits British Navy,” 2010). The hacker, known by the pseudonym TinKode, stole the usernames and passwords stored on the site (“Hacker hits British Navy,” 2010). Though the attack was limited to a single website, the damage inflicted extended past the compromise of sensitive credentials (“Hacker hits British Navy,” 2010). The attack identified weaknesses in the cyber defense strategy of the UK, demonstrating the efficacy of non-state SQL injection-based cyberattacks against world powers and establishing SQLIA as a national cyber threat (“Hacker hits British Navy,” 2010).

Attacks from 2010-2015

SQLIA grew to become the most significant threat to web applications in 2011 (“TripAdvisor data stolen,” 2011). Expedia, one of the largest online travel websites, had its TripAdvisor site exploited by a SQL injection attack (“TripAdvisor data stolen,” 2011). The attack leaked an undisclosed number of email addresses from the list of 20 million subscribed customers (“TripAdvisor data stolen,” 2011). Though the attack resulted in potentially millions of customers losing the privacy of their email addresses, the purpose of the attack came after its success (“TripAdvisor data stolen,” 2011). With the email addresses in hand, hackers could send phishing, junk, and spam emails (“TripAdvisor data stolen,” 2011). Interacting with the malicious emails could lead to the extraction of sensitive information, passwords, and other Personally Identifiable Information (PII) from victims, to which TripAdvisor warned its subscribers (“TripAdvisor data stolen,” 2011).

The threat of SQL injection rose to new levels, with attacks showing an increase in virality (“LizaMoon injection attack,” 2011). One of the largest SQL injection attacks to date infected over half a million website domains in late march of 2011 (“LizaMoon injection attack,” 2011). The attack, dubbed LizaMoon, started with 28,000 infected websites in its arsenal

("LizaMoon injection attack," 2011). Since then, LizaMoon has infected millions of legitimate URLs on Google with malicious code ("LizaMoon injection attack," 2011). The injected code redirects users to a malicious site that launches a fake antivirus scam ("LizaMoon injection attack," 2011). During the scam, it prompted users to download antivirus in response to a false malware report ("LizaMoon injection attack," 2011). Over 20 sites were listed as malicious redirects used by the SQLIA, establishing LizaMoon as one of the greatest mass-injection attacks ever recorded ("LizaMoon injection attack," 2011). A vulnerability in outdated Web systems, such as Content Management Systems (CMS) and blog systems on Microsoft SQL Server 2003 and 2005 databases, were likely exploited to launch the attack ("LizaMoon injection attack," 2011).

The confidentiality of user PII is repeatedly a primary target of SQL injection attacks ("SQL injection exposes Comodo partner," 2011). Hackers posted exposed data files stolen from ComodoBR via SQL injection on a text-sharing site ("SQL injection exposes Comodo partner," 2011). The Brazil-based partner of the IT security company Comodo had customer information exposed through web-based SQL injection ("SQL injection exposes Comodo partner," 2011). Names, addresses, emails, phone numbers, and other PII were made accessible by the hackers on Pastebin, exposing the victims to possible phishing and similar scams ("SQL injection exposes Comodo partner," 2011). They also stole ComodoBR employee credentials, allowing the hackers to potentially access company data ("SQL injection exposes Comodo partner," 2011). Also, the attack attempted to access certificate-signing requests ("SQL injection exposes Comodo partner," 2011). By trying to access certificate-signing requests, the hackers indicated a desire to impersonate legitimate websites with counterfeit Secure Socket Layer (SSL) certificates, mirroring similar attacks against a Comodo reseller in Italy ("SQL injection exposes Comodo

partner," 2011). The ComodoBR attack and others like it pushed security professionals to rethink the certificate-verification process ("SQL injection exposes Comodo partner," 2011).

The restructuring of security principles continued following a security breach at Nokia in August 2011 ("Nokia SQL injection," 2011). Nokia users had email addresses and other PII compromised via SQL injection ("Nokia SQL injection," 2011). Initially, the attack infected Nokia's forums website with a redirect that did not directly harm users ("Nokia SQL injection," 2011). However, the attacker also gained access to customer PII stored on the Nokia database ("Nokia SQL injection," 2011). The published data opened victims to high volumes of unsolicited emails using their stolen PII ("Nokia SQL injection," 2011). "pr0tect0r AKA mrNRG", the perpetrator behind the breach, used it to expose and ridicule the poor security standards of Nokia ("Nokia SQL injection," 2011). Having a simple attack like SQL injection breach the security of a major corporation like Nokia revealed a need for drastic change in IT security policy ("Nokia SQL injection," 2011).

As seen in the Asprox botnet attack of 2009, the use of malicious iFrames in tandem with SQL injection is dangerously effective (Bort, 2011). Continuing this trend in 2011, Bort noted that hackers compromised websites based on Microsoft's Active Server Pages Network Enabled Technologies (ASP.NET) framework. ASP.NET is a standardized dynamic website and web application development framework used by web developers. The hackers used SQL injection to compromise 180,000 sites with malicious JavaScript code. They designed the code used in the attack to load an iframe, redirecting users to a malicious web page. Bort (2011) states that once the user connects to the page, it attempts to infect the victim with malware via drive-by exploits. Many popular antivirus software failed to detect the SQL injection attack. The incident pressured Microsoft to release statements urging ASP.NET web developers to review SQL-related

procedures in their code for injection vulnerabilities, promoting secure development for web applications.

Hackers leveraged advances in automation tools to launch a mass-SQL injection attack that infected over a million pages ("Mass SQL automated tools," 2012). The "Lilupophilupop" SQLIA campaign leveraged automated tools and search engine reconnaissance to grow from 80 to over 1 million infected URLs ("Mass SQL automated tools," 2012). The malicious pages redirected victims to sites that led to Lilupophilupop.com; a malicious site containing an antivirus scam attack ("Mass SQL automated tools," 2012). The victims would pay for fake antivirus and install malware onto their systems, allowing the possibility of further infection and damage ("Mass SQL automated tools," 2012). The Lilupophilupop attack was very similar in method to the LizaMoon attack, as they both led their victims to antivirus scam pages ("Mass SQL automated tools," 2012). Hackers used the Google search engine to discover websites vulnerable to SQL injection, and automated tools, such as bots, made searching more efficient ("Mass SQL automated tools," 2012). Post-mortem analysis led security researchers to emphasize the need for secure code reviews for vulnerabilities ("Mass SQL automated tools," 2012).

SQL injection has been a prominent threat to web applications (Perlroth, 2012). Perlroth's (2012) report demonstrates that unlawful access and distribution of sensitive information remain the goal of many attacks during this time. In Perlroth's (2012) report, thousands of personal records from 53 universities, including Harvard, Stanford, and Princeton, were leaked by a group of hackers known as Team GhostShell. The group leveraged SQL injection against university websites and retrieved PII such as email addresses, addresses, usernames, passwords, and other information.

The gadget and toy company VTech experienced a data breach allowing hackers access to customer data (By, 2015). In the breach, hackers exposed the PII of 200,000 children and 5 million parents. Among the data stolen was the home addresses of both children and parents as well as the password and security question of the parents. By (2015) notes that, this data could allow the attackers to connect children with their parents based on their address and become intimately knowledgeable about them. Hackers compromised the site via a simple SQLIA against the front-end web application to gain access to Vtech servers.

Attacks from 2015-2021

PII data theft continued through SQLIA into the late 2010s. In 2016, hackers successfully compromised the servers of the major video game company Epic Games ("Gamers' accounts compromised," 2016). The attack stole the PII data of over 800,000 users of the company's forums website ("Gamers' accounts compromised," 2016). Email addresses were among the stolen data ("Gamers' accounts compromised," 2016). The attack led security researchers to criticize the poor security of Epic Games, implying that the simplicity and frequency of SQL injection should make it a top priority when architecting security features ("Gamers' accounts compromised," 2016).

While SQLIA are typically used by hackers to gain usable PII for monetary or personal gain, state-sponsored hackers also use SQLIA to push political agendas (Coulombe, 2017). During the 2016 election, Russian hackers successfully infiltrated the voter database of a voting jurisdiction in Illinois, states Coulombe. Using SQL injection against the voter registration and election sites, the hackers gained access to 15 million files containing data on Illinois voters from 2006 to the present day. The hackers maintained persistence in the system for three weeks before discovery.

Coulombe (2017) reports that a security assessment revealed that the federal government had the highest prevalence of low-effort exploitable vulnerabilities, including SQL injection.

Credential stuffing is often begotten of a successful SQLIA, as it leverages compromised account credentials to bypass security measures ("Gaming Industry Targeted," 2019). From November 2017 to March 2019, hackers relentlessly attacked web applications with SQL injection to launch credential-stuffing attacks ("Gaming Industry Targeted," 2019). Hackers subjected the gaming industry to approximately 12 billion credential-stuffing attacks within the 17-month period ("Gaming Industry Targeted," 2019). The frequent attacks made the gaming industry the most rapidly rising target for credential stuffing. SQL injection, making up two-thirds of all web application attacks during 2019, was the leading vector for credential stuffing attacks ("Gaming Industry Targeted," 2019).

The gaming industry continued to be attacked by web application attacks throughout the Covid-19 pandemic at an even higher frequency than in 2019 ("Video Game Industry," 2021). During the COVID-19 pandemic, cyberattack traffic against the gaming community grew more than any other industry ("Video Game Industry," 2021). Hackers launched over 240 million web application attacks in 2020 alone, increasing 340% since 2019 ("Video Game Industry," 2021). SQL injection was the primary attack method against the gaming industry, making up 59% of all cyberattacks executed ("Video Game Industry," 2021). The gaming community also suffered 11 billion credential stuffing attacks as a result of the web application attacks, showing a 224% increase over 2019 ("Video Game Industry," 2021).

Though not an extremely damaging attack, SQL injection, when combined with other cyberattacks, has been shown to cause catastrophic damage to organizations (Bannister, 2021). Bannister described how a Russian hacker group known as REvil used an unpatched SQL

injection vulnerability to launch a massive ransomware attack against the IT management platform Kaseya VSA. The ransomware deployed by REvil affected more than 1,000 organizations in Kaseya VSA's supply chain. Bannister stated that the hackers demanded a ransom of 70 million dollars in exchange for Kaseya VSA systems.

The Cloud-based IT management and remote monitoring platform patched the issues that allowed for the large-scale ransomware attack by REvil (Afifi-Sabet, 2021). The initial response from Kaseya underestimated the scale of the attack immensely, according to Afifi-Sabet. In the end, the hackers compromised over 1,500 total clients. Of the resolved flaws that, Afifi-Sabet noted that one was related to the SQL injection vulnerability that allowed for the exploitation of Kaseya. Afifi-Sabet's report of this large-scale attack reinforced the threat of SQL injection to modern applications.

Analysis

SQL injection has evolved drastically from its earliest moments in 2005 until recent memory in 2021. By observing successful SQLIA, key attack themes and methods can be drawn out. The sections below list the observed themes and methods drawn from the successful attacks mentioned.

SQLIA is Primarily Targeting PII

While many aspects of SQL injection changed over time, one theme remained constant: SQLIA are consistently used to steal PII. In its infancy, SQLIA were used to steal credit card numbers, emails, and other PII (Nogod, 2016; Vijayan & Gaudin, 2009). Into the early 2010s, SQL injection terrorized web applications by stealing PII from their databases ("Hacker hits British Navy," 2010; "TripAdvisor data stolen," 2011; "SQL injection exposes Comodo partner," 2011; "Nokia SQL injection," 2011; Perlroth, 2012; By, 2015). Continuing the trend of targeting

PII into the late 2010s, SQL injection targeted new audiences ("Gamers' accounts compromised," 2016; Coulombe, 2017; "Gaming Industry Targeted," 2019; "Video Game Industry," 2021). SQL injection, at its core, is designed to steal PII from web application databases.

SQLIA as a Means to Launch Attacks

Another core aspect observed in successful SQLIA is that it is often used as a vehicle to launch another attack or deploy malware. In the late 2000s and into the early 2010s, the Asprox botnet and similar attacks used SQL injection to load malicious iFrames used to install malware ("Botnet installs SQL injection," 2008; Ullrich & Lam, 2008; "SQL injection," 2009; Bort, 2011). Hackers continued to hijack web applications and use them to trick victims into downloading malware ("Mass SQL automated tools," 2012).

Though malware was a popular follow-up to a successful SQLIA, hackers often used less sophisticated attacks, like credential stuffing, with stolen PII ("Gaming Industry Targeted," 2019; "Video Game Industry," 2021). The potential damage of SQLIA continued to rise as researchers identified it as the precursor to ransomware attacks, a critical threat to organizations (Bannister 2021). Through the deployment of increasingly sophisticated and damaging attacks post-successful injection, the danger of SQLIA has only increased over time.

SQLIA as a Means to Scam

Instead of attempting to execute sophisticated cyberattacks, many hackers will use the stolen PII in a successful SQLIA to launch scams. Instead of compromising computers, hackers will try to leverage human error and misinformation to their advantage. The Asprox botnet and other attacks attempted to trick victims into downloading malware by redirecting them to a malicious site ("Botnet installs SQL injection," 2008; Ullrich & Lam, 2008). Hackers continued

using malicious redirects post-SQL injection to trick victims in the LizaMoon and the Lilupophilupop injection attacks. Both of these attacks used a fake antivirus scam designed to install malware on the victim's system ("LizaMoon injection attack," 2011; "Mass SQL automated tools," 2012).

Hackers used stolen PII to initiate even simpler scams. Phishing campaigns, spam emails, and junk emails were popular follow-ups to successful SQLIA since hackers consistently acquired email addresses ("TripAdvisor data stolen," 2011; "Nokia SQL injection," 2011). The trend of scamming post-SQLIA was not reported as frequently in attacks during the late 2010s. However, PII continued to be a primary target throughout the life of SQLIA, meaning it is reasonable to assume that scams utilizing stolen PII will continue to afflict victims of SQLIA.

SQLIA Enhancements

Over the nearly 20 years of SQL injection observed, hackers made varying technology and methodology-based enhancements to enhance the cyberattack. Of the upgrades made to the cyberattack, three enhancements, in particular, were consistently observed in successful SQLIA. An increase in SQLIA area of effect, an increase in attack frequency, and an increase in SQLIA potency through technology-based improvements.

Increased Area of Effect

In order to affect more users and steal more PII, hackers continuously enhanced the effective area, or blast radius, of their SQL injection cyberattacks. As early as 2008, botnets were the catalyst for the increased blast radius of SQL injection ("Botnet installs SQL injection," 2008). The blast radius of SQL injection continued to rise throughout the early 2010s, with the Lizamoon and Lilupophilupop SQL injection cyberattacks. Lizamoon and Lilupophilupop emphasized virality as a means to infect millions of web pages, being the largest SQLIA to date

("LizaMoon injection attack," 2011; "Mass SQL automated tools," 2012). However, after 2012, the trend of increasing SQL injection's area of effect became less apparent, indicating a possible shift in attack methodology.

Increased Frequency of Attacks

While the trend of increasing the SQLIA blast radius stalled by the early 2010s, hackers continuously increased the frequency of their attacks throughout the observed 15-year period. SQL injection initially grew in frequency and popularity in 2006, with security experts reporting a doubling of its attack frequency in mere months ("Spike in SQL injection," 2006). In more recent memory, SQL injection was the most popular web application cyberattack against some of the largest industries in the United States, such as the video game industry. SQL injection was the most common form of web application attack in 2019. It was used extensively for credential stuffing attacks, indicating that its attack frequency was continuously rising ("Gaming industry targeted," 2019). SQLIA frequency continued to increase as hackers used it to launch more credential stuffing attacks in 2020, an attack form that has increased three-fold since 2019 ("Video game industry," 2021). SQL injection has shown no signs of slowing down and continues to threaten web applications.

Increased Attack Potency Via Technology

In the late 2000s and early 2010s, hackers proved that SQL injection cyberattacks could evolve alongside technology. By using technological enhancements, SQLIA can achieve tremendous success. The use of Botnets allowed hackers to accomplish a greater area of impact, indicating that technical concepts like botnets can improve the efficacy of SQLIA ("Botnet installs SQL injection," 2008). Furthermore, attackers used automation to look for vulnerable web pages in the Lilupophilupop SQL injection cyberattack ("Mass SQL automated tools,"

2012). Though the observed SQL injection attacks in the late 2010s failed to indicate a continuation of this trend, it is reasonable to assume that new technologies, like adversarial ML for example, will continue to be used by hackers to enhance the efficacy and success rate of SQL injection.

SQLIA Improves Cybersecurity Posture

The trends in successful SQL injection described so far have all pertained to the nature of the attacks. However, the evolution in SQL injection has also forced a shift in factors outside the attack itself. Observed trends in successful SQLIA can also include the attack's impact on overall security awareness and posture. As SQL injection has grown and affected more victims over the observed 15-year period, cybersecurity posture has also evolved in response to these successful attacks.

Based on the lack of security remediations listed in the observed successful SQLIA in the late 2000s, there was less pressure on security researchers and professionals to develop countermeasures to SQLIA. SQLIA was still a relatively novel attack type, and research was still defining and classifying SQLIA (Halfond et al., 2006; Su & Wasserman, 2006). However, SQLIA in the early 2010s led security researchers to re-evaluate cybersecurity best practices, such as rethinking the certificate verification process and emphasizing the need for code reviews to improve security ("SQL injection exposes Comodo partner," 2011; "Nokia SQL injection," 2011; Bort, 2011; "Mass SQL automated tools," 2012). Security researchers and professionals continued to gauge their cybersecurity readiness into the late 2010s, with SQLIA causing both governmental and organizational entities to rethink their cybersecurity strategies ("Gamers' accounts compromised," 2016; Coulombe, 2017). Though successful SQLIA are tragic events

and cause pain across cyberspace, their benefit in improving cybersecurity tactics, techniques, and procedures is undeniable.

The Future of SQL Injection

While researchers have proposed several theoretical frameworks to counter contemporary and future SQLIA, their implementations have yet to be validated (Albalawi & Mohamed, 2021). Furthermore, the developments of technologies like artificial intelligence and machine learning may both pose a continued risk to or improve the efficacy of SQLIA and should be investigated in further research (Gogoi et al., 2021). Nevertheless, analysis of SQLIA literature and successful attacks reveals key themes that will persist in SQLIA. A review of the literature from 2005 to 2010 saw researchers defining SQL injection and creating basic cybersecurity measures against it. From 2010 to 2015, cybersecurity literature provided more advanced mitigation and detection techniques as well as defined the threat of SQLIA. Finally, SQLIA literature from 2015 to the early 2020s exhibited themes of developing cutting-edge defensive measures for newer technologies as well as testing the efficacy of SQLIA against them. The themes of each five-year interval indicate that SQL injection literature in the future will continue to focus on developing more innovative approaches to combating SQLIA and consistently re-evaluating the threat of SQL injection against new technologies.

Successful instances of SQLIA provide the most significant insight into the future direction of web application hackers. The themes and methods identified give way to a clearer understanding of the future of SQLIA. PII and other exploitable data will continue to be the focus of SQLIA, along with deploying scams and more advanced cyberattacks post-injection. While the increase in blast radius has become a lesser focus in recent years, hackers are showing no signs of reducing the frequency of SQL injection. Just as security researchers continue to use

newer technologies to combat SQL injection, hackers will do the same to enhance the cyberattack. Finally, successful SQL injection will continue to prompt security professionals to re-evaluate security best practices and advance mitigation and detection techniques.

Conclusion

Web applications play a critical role in society today. Understanding the threats that they face helps security professionals better defend them. SQL injection has remained a common yet severe threat to web applications. Though SQL injection remains a threat, anticipating its future in cyberspace helps dramatically reduce the risk it presents. According to the review of the literature detailing SQL injection attacks and methodologies, it was found that researchers are shifting their focus from trying to have a comprehensive understanding of SQL injection and its variations to testing and analyzing its effectiveness among new technologies. Furthermore, SQL injection attacks themselves have been moving away from simply stealing sensitive data but rather using SQL injection to launch even greater attacks against web applications. This provides a foundation to produce this report which seeks to establish a correlation among SQL injection attack methods and themes in order to anticipate its evolution and potential future implications.

References

Afifi-Sabet, K. (2021). Kaseya patches VSA flaws exploited in REvil ransomware attack. *IT Pro*. http://ezproxy.liberty.edu/login?qurl=https%3A%2F%2Fwww.proquest.com%2Fmagazines%2Fkaseya-patches-vsa-flaws-exploited-revil%2Fdocview%2F2550573435%2Fse-2%3Faccountid%3D12085.

Alam, D., Kabir M., Bhuiyan, T., & Farah, T. (2015). A case study of SQL injection vulnerabilities assessment of .bd domain web applications. *2015 Fourth International Conference on Cyber Security, Cyber Warfare, and Digital Forensic (CyberSec)*, 73-77. https://doi.org/10.1109/CyberSec.2015.23.

Albalawi, S., & Mohamed, A. (2022). Authentication enhancement against SQL injection attacks (SQLIAs). *2022 2nd International Conference on Computing and Information Technology (ICCIT)*, 405-408. https://doi.org/10.1109/ICCIT52419.2022.9711626.

Bannister, A. (2021). Revil ransomware attackers demand $70M following Kaseya VSA Supply Chain attack. *The Daily Swig*. https://portswigger.net/daily-swig/revil-ransomware-attackers-demand-70m-following-kaseya-vsa-supply-chain-attack.

Bono, S. & Domangue, E. (2012). SQL injection: A case study. *Independent Security Evaluators.* http://docs.media.bitpipe.com/io_10x/io_106211/item_570783/ISE%20Whitepaper%20-%20SQL%20Injection%20-%2010-1-12.pdf.

Bort, J. (2011). Massive SQL injection attack has compromised nearly 200,000 ASP.Net sites; Attackers are successfully planting malware onto Websites and PCs with little help from antivirus scanners. *Network World*.

https://link.gale.com/apps/doc/A270612901/GBIB?u=vic_liberty&sid=bookmark-GBIB&xid=a3e1d17c.

Botnet installs SQL injection tool. (2008). *eWeek.* https://link.gale.com/apps/doc/A217853102/GBIB?u=vic_liberty&sid=bookmark-GBIB&xid=88e54e5a.

By, A. S. (2015). Leaked pentagon bomber data, scoop up toy company details on kids, and pinch credit cards from Wisconsin resort. *Nextgov.Com (Online).* https://go.openathens.net/redirector/liberty.edu?url=https://www.proquest.com/magazines/leaked-pentagon-bomber-data-scoop-up-toy-company/docview/1738103552/se-2.

Chen, T. & Buford, J., (2009). Design considerations for a honeypot for SQL injection attacks. *2009 IEEE 34th Conference on Local Computer Networks*, 915-921, https://doi.org/10.1109/LCN.2009.5355040.

Coulombe, R. (2017). Lethal injection. *Security Dealer & Integrator, 39*, 20-21. https://go.openathens.net/redirector/liberty.edu?url=https://www.proquest.com/magazines/lethal-injection/docview/1936421776/se-2.

Expedias TripAdvisor member data stolen in possible SQL injection attack 522785. (2011). *eWeek.* https://link.gale.com/apps/doc/A252362009/GBIB?u=vic_liberty&sid=bookmark-GBIB&xid=4f967353.

Fu, X., Wang, Z., Chen, Y., Chen, Y., & Wu, H. (2019). SQL injection in cloud: An actual case study. *Advances on P2P, Parallel, Grid, Cloud and Internet Computing, 24*. https://doi.org/10.1007/978-3-030-02607-3_13.

Gaming industry targeted with 12bn attacks and counting. (2019). *Business World,* https://go.openathens.net/redirector/liberty.edu?url=https://www.proquest.com/magazines/gaming-industry-targeted-with-12bn-attacks/docview/2239131881/se-2.

Gogoi, B., Ahmed, T., & Dutta, A. (2021). Defending against SQL Injection Attacks in Web Applications using Machine Learning and Natural Language Processing. *2021 IEEE 18th India Council International Conference (INDICON)*, 1-6. https://doi.org/10.1109/INDICON52576.2021.9691740.

Gupta, H., Mondal, S., Ray, S., Giri, B., Majumdar, R., & Mishra, V. P. (2019). Impact of SQL injection in database security. *International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*, 296-299. https://doi.org/10.1109/ICCIKE47802.2019.9004430.

Hacker hits British Navy website with SQL injection attack 108377. (2010). *eWeek*. https://link.gale.com/apps/doc/A241582371/GBIB?u=vic_liberty&sid=bookmark-GBIB&xid=19a56f0d.

Halfond, W. G., Viegas, J., & Orso, A. (2006). A classification of SQL-injection attacks and countermeasures. *In Proceedings of the IEEE international symposium on secure software engineering*, *1*, 13-15. https://www.cc.gatech.edu/fac/Alex.Orso/papers/halfond.viegas.orso.ISSSE06.pdf.

Keizer, G. (2008). Huge web hack attack infects 500,000 pages. *Computerworld*. https://web.archive.org/web/20151019183916/http://www.computerworld.com/article/2535473/security0/huge-web-hack-attack-infects-500-000-pages.html.

Kovacs, E. (2013). Hackers leak data allegedly stolen from Chinese Chamber of Commerce website. *Softpedia News*. https://news.softpedia.com/news/Hackers-Leak-Data-Allegedly-Stolen-from-Chinese-Chamber-of-Commerce-Website-396936.shtml.

Leppänen, A., Toiviainen, T., & Kankaanranta, T. (2020). From a vulnerability search to a criminal case: script analysis of an SQL injection attack. *International Journal of Cyber Criminology, 14(1),* 63-80. http://ezproxy.liberty.edu/login?qurl=https%3A%2F%2Fwww.proquest.com%2Fscholarly-journals%2Fvulnerability-search-criminal-case-script%2Fdocview%2F2404395593%2Fse-2.

Leyden, J. (2021). SQL injection flaw in billing software app tied to US ransomware infection. *The Daily Swig*. https://portswigger.net/daily-swig/sql-injection-flaw-in-billing-software-app-tied-to-us-ransomware-infection.

Liwu, D., Ruzhi, X., Lizheng, J., & Guangjuan, L., (2009). A database protection system aiming at SQL attack. *Fifth International Conference on Information Assurance and Security,* 655-657. https://doi.org/10.1109/IAS.2009.322.

LizaMoon mass SQL injection attack escalates out of control 378108. (2011). *eWeek*. https://link.gale.com/apps/doc/A253041103/GBIB?u=vic_liberty&sid=bookmark-GBIB&xid=05d015ca.

Mass SQL injection attacks uses automated tools search to infect new sites 671231. (2012). *eWeek*. https://link.gale.com/apps/doc/A279278331/GBIB?u=vic_liberty&sid=bookmark-GBIB&xid=d6114548.

Moyle. (2007). The blackhat's toolbox: SQL injections. *Network Security.*, *2007*(11), 12–14. https://doi.org/10.1016/S1353-4858(08)70040-0.

Munadi, R., Surya Fajri, T., Meutia, E, & Mustafa, E., (2013). Analysis of SQL injection attack in web service (a case study of website in Aceh province). *3rd International Conference on Instrumentation, Communications, Information Technology and Biomedical Engineering (ICICI-BME)*, 431-435. https://doi.org/10.1109/ICICI-BME.2013.6698541.

Nogod. (2016). The mind of a credit card hacker. *The Internal Auditor : Journal of the Institute of Internal Auditors.*, *73*(4). https://search.ebscohost.com/login.aspx?direct=true&db=heh&AN=117767652&site=ehost-live&scope=site.

Nokia shuts down forums after SQL injection exposes developer info 186516. (2011). *eWeek*. https://link.gale.com/apps/doc/A265556068/GBIB?u=vic_liberty&sid=bookmark-GBIB&xid=d3f4d36d.

OWASP top 10 (2021). *Open Web Application Security Project*. https://owasp.org/Top10/.

Perlroth, N. (2012). Hackers breach 53 universities and dump thousands of personal records online. *New York Times*. https://web.archive.org/web/20121005021105/http://bits.blogs.nytimes.com/2012/10/03/hackers-breach-53-universities-dump-thousands-of-personal-records-online/.

Poulsen, K. (2002). Guesswork plagues web hole reporting. *Security Focus*. https://web.archive.org/web/20120709141229/http://www.securityfocus.com/news/346.

Ron, A., Shulman-Peleg, A., & Puzanov, A. (2016). Analysis and mitigation of NoSQL injections. *IEEE Security & Privacy, 14*(2), 30-39. https://doi.org/10.1109/MSP.2016.36.

Sadeghian, A., Zamani, M., & Ibrahim, S. (2013). SQL injection is still alive: a study on SQL injection signature evasion techniques. *2013 International Conference on Informatics and Creative Multimedia*, 265-268. https://doi.org/10.1109/ICICM.2013.52.

Sakib. (2021). A survey on detection and prevention of SQL and NoSQL injection attack on server-side applications. *International Journal of Computer Applications, 183*(10), 1–7. https://doi.org/10.5120/ijca2021921396.

Singh, N., Dayal, M. Raw, R., & Kumar S. (2016). SQL injection: Types, methodology, attack queries and prevention. *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2872-2876. https://ieeexplore.ieee.org/document/7724789.

Spike seen in SQL injection attacks. (2006). *Computerworld*, *40*(31), 26. https://link.gale.com/apps/doc/A149649263/GBIB?u=vic_liberty&sid=bookmark-GBIB&xid=9d1e240a.

Sonakshi, Kumar, R., & Gopal, G. (2016). Case study of SQL injection attacks. *International Journal of Engineering Sciences and Research Technology, 5(7),* 176–189. https://doi.org/10.5281/zenodo.56935.

SQL injection. (2009). *SC Magazine*, *20*(10), 12. https://link.gale.com/apps/doc/A210119656/GBIB?u=vic_liberty&sid=bookmark-GBIB&xid=bd8f294a.

SQL injection attack. (2008). *SC Magazine*, *19*(6), 14. https://link.gale.com/apps/doc/A180638910/GBIB?u=vic_liberty&sid=summon&xid=df45380f.

SQL injection attack exposes Comodo partner customer data 530220. (2011). *eWeek*. https://link.gale.com/apps/doc/A257326528/GBIB?u=vic_liberty&sid=bookmark-GBIB&xid=42c01e6b.

Su, Z., & Wassermann, G. (2006). The essence of command injection attacks in web applications. *SIGPLAN Notices, 41(1),* 372-382. https://doi.org/10.1145/1111320.1111070.

Thousands of gamers' accounts compromised in hack attack. (2016, 11). *Computer Shopper,* , 18. https://go.openathens.net/redirector/liberty.edu?url=https://www.proquest.com/magazines/thousands-gamers-accounts-compromised-hack-attack/docview/1833137287/se-2.

Ullrich, & Lam, J. (2008). Defacing websites via SQL injection. *Network Security.*, *2008*(1), 9–10. https://doi.org/10.1016/S1353-4858(08)70007-2.

Video game industry faced highest growth in cyberattacks during pandemic. (2021, June 25). *Total Telecom Magazine,* NA. https://link.gale.com/apps/doc/A667025113/GBIB?u=vic_liberty&sid=bookmark-GBIB&xid=629ec046.

Vijayan, J., & Gaudin, S. (2009). U.S. says SQL injection caused major breaches. *Computerworld, 43(26),* 4. https://link.gale.com/apps/doc/A207419462/GBIB?u=vic_liberty&sid=bookmark-GBIB&xid=edb08ab4.