Understanding Cybersecurity Threats

Ryan Rinaldi

Info 312 Risk Management

Individual Project

This project considers the current biggest cybersecurity threats paired with common controls and their implementations, with the goal of further understanding each threat at a detailed level. The group work done with Nordstrom provides a very good overview of risks in general; this project will compliment that with a closer focus on a few specific threats. Much of the information found here comes from the Open Web Application Security Project (referred to as OWASP), as they are a worldwide industry standard and one of the most knowledgeable resources on the frequency and severity of specific threats. Specifically, OWASP regularly releases an updated list of the top 10 security threats. This list was used to determine some of the most pressing threats that businesses currently face. The threats chosen are loosely based on the group work done with Nordstrom, as only threats that could affect Nordstrom were used. One major asset that Nordstrom uses is it’s customer relationship management tool: the Personal Book contains personal and private information about virtually all Nordstrom’s shoppers, including purchase history, address, and profile. The threats covered in this project are two of the top three threats in the OWASP Top 10 list: code injection and cross site scripting.

Injection will look at methods in which attackers are able to inject or run SQL code on Nordstrom’s servers, as well as controls used to prevent such attacks and how these controls are implemented. This is OWASP’s first threat on it’s Top 10 list, due to it’s prevalence and extremely high potential loss. As a company that stores huge amounts of personal customer information, as well as manages payment information itself, Nordstrom stands to lose massively if it’s databases are compromised. Cross site scripting is a form of code injection, but focuses on injecting scripts into the website itself to attack other users rather than Nordstrom itself. Nordstrom’s website is extensive and services a massive amount of customers, so any kind of cross site scripting vulnerability could affect a large portion of Nordstrom’s overall customers, almost all of which have payment or personal information associated with their accounts. Nordstrom’s e-commerce is their fastest-growing channel, according to president Blade Nordstrom (Briggs). With e-commerce being such a major part of Nordstrom’s overall revenue (11.1% of Nordstrom’s sales were online in 2013), customer accounts have to be consistently protected and secured. These threats are some of the biggest faced by Nordstrom, and adequate controls are absolutely necessary to the continued success of the company.

The controls discussed in this project will occasionally reference certain aspects of the federal Cybersecurity Framework. In particular, it references the Framework Cores, which represent “key cybersecurity outcomes identified by industry as helpful in managing cybersecurity risk” (Cybersecurity Framework). The functions that compose the Core are Identify, Protect, Detect, Respond, and Recover. “Identify” means to understand the threat, in context of the business assets, resources, likelihood and potential losses. “Protect” means to design and implement safeguards that limit or prevent the impacts of threats and attacks. “Detect” means to design and implement tools that allow the organization to identify attacks as they happen, as well as determine when and how they are attacked. “Respond” is to develop the appropriate tools to take action when a detected attack occurs. “Recover” means to develop and implement tools that restores any capabilities or services that are impaired or limited during an attack. Controls for cybersecurity threats fall primarily into a single category, which will reference the ultimate purpose of the control.

**Injection**

This section looks specifically at SQL injection. Injection is the use of a text-based attack to manipulate the database. Successful attacks can “allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server” (OWASP).

To understand a threat and how to control it, we must have a thorough understanding of the threat vector and what the threat looks like. A SQL injection attack can occur when database code uses a value retrieved from a website or user. The SQL code could look like the following:

sql = “SELECT \* FROM UserData WHERE UserId = “ + userInputId;

In this case, the variable “userInputId” is retrieved from the user, perhaps in a web form where the user is asked to log in with his credentials. However, the user may input the following code as his user id: 12938273 or 1=1. In this case, the SQL string from before becomes the following, when the variable is replaced by the value:

SELECT \* FROM UserData WHERE UserId = 12938271 or 1=1

This means that the WHERE statement always evaluates as true; the attacker doesn’t need to have the correct User ID value, because by injecting the “or 1=1” code, he can always pass the WHERE statement. This obviously defeats the purpose of the WHERE statement in the SQL code, and could potentially lead to major breaches or data loss. In this case, the attacker would have access to all columns and rows from UserData, as if he had the correct UserId for every entry.

Many other specific types of injection exist: some allow attackers to spoof users by forcing username and password matching code to evaluate to true. If a value such as 12938271; DROP TABLE UserData was passed, it could potentially delete the User Data table, resulting in potential loss of data or service while the database is restored. With the ability to tamper with or disclose information on private databases, attackers would be able to do significant damage to Nordstrom’s reputation, Personal Book, and overall sales over a given time. Strong controls against SQL injection are a requirement for Nordstrom’s success.

SQL injection gives the attacker extensive control over the database, as he or she is able to execute any SQL command. The integrity of the database could be at risk, due to manipulated information. The privacy of the information stored on the database is also at risk. Ultimately, the amount of access the attacker has is determined by the database in use and its implementation. In certain cases attackers are able to read or write to files, or execute shell commands on the operating system (Acunetix). SQL injection is very difficult to detect; it can only really be detected once data theft of modification has been discovered.

The business impacts of SQL injection vary, but become extremely significant for massive companies like Nordstrom. Nordstrom also creates extensive personal profiles in its Personal Book, meaning they have to store a lot of important and private data. Nordstrom also manages its Payment Card Industry Data Security Standard (PCI DSS) compliance itself, without the use of a dedicated PCI DSS service. This means that not only personal information but also payment information has to be stored by Nordstrom, opening up the company to a new level of loss. Theft or unauthorized access could result in fraudulent purchases or manipulated records, resulting in overall loss of inventory or overall income. Such events could also find Nordstrom in violation of PCI DSS compliance, especially if proper SQL injection controls are not used. Fines for this could vary greatly depending on the size of the breach. These fines and loss of inventory/income could be very large, yet still not as significant as the reputation of the company.

Reputation is extremely important for a successful company, and recent articles showcasing cybersecurity threat in certain businesses show just how much damage such threats can do. Once certain companies are in the media for security breaches, reputation plummets and customers begin shopping elsewhere at businesses they believe are trustworthy. Loss of any kind of information, including payment information or Personal Book data, could have significant impact on Nordstrom’s reputation. With the amount of personal information stored on the Personal Book, customers could be scared away from Nordstrom at the thought of losing such information.

Controls for SQL Injection focus on keeping untrusted data separate from commands and queries. With this separation, any kind of injection attack (if accurately defined as untrusted data) is separated from SQL code, and has no opportunity to run. Controls for SQL injection focus on the “Protect” function of the Cybersecurity Framework Core. Identification is done to design controls that adequately meet the “Protect” function. As previously discussed, detection of SQL injection is very difficult and cannot be reliably done to facilitate response to attacks as they occur. “Recovery” is indeed a control that is used in the context of database backups, but does not help at all in cases of data theft.

Certain weaknesses that create SQL injection vulnerability include incorrectly or unfiltered escape characters or using weakly-typed user inputs. Filtering input for escape characters prevents injection of SQL symbols that could confuse the interpreter. Weakly-typed inputs are more likely to run when user input doesn’t closely match the expected input. For example, if the application accepts a number, but the code is so weakly-typed that SQL accepts a string as input as well, attackers could inject SQL code that would be unexpectedly executed. One “band-aid” solution could be to filter input to not allow any terms or words that could be confused for SQL commands, but this is a poor solution as many applications require these terms to be submitted. Better solutions allow these terms to be submitted, but will not allow them to be executed.

Strong controls for SQL injection primarily center around sanitising user input or the use of a parameterized API. Sanitising inputs include escaping special characters (replacing them with an equivalent term that won’t run in the database), and strongly-typing user input variables. This approach may include using black or white lists that define what valid data may be. Query parameterization refers to the process of building queries in a way that first defines all static SQL code, and each parameter is passed to the query separately (OWASP). With this approach, the database is able to distinguish between the static SQL code and user input, as all variables are placeholders and can only store values, not SQL fragments. Like many controls, it is safest to use both. Inputs should be strongly-typed and escaped for special characters when applicable, and the SQL code used to run the database should be parameterized, to prevent any injection that makes it through the escaping process from running.

SQL injection is a prominent threat that definitely demands attention and prevention, but with proper controls Nordstrom is absolutely able to create a safe and secure system.

**Cross Site Scripting**

Cross site scripting, sometimes referred to as XSS, is the most prevalent web application security flaw (OWASP). Cross site scripting is technically a form of code injection, but is substantially different from the SQL Injection discussed in the previous section. The attack occurs when a user is able to inject code into a webpage that uses user input to modify the page in some way. Unlike some other kinds of injection (SQL injection included), cross site scripting specifically targets other users, rather than the website hosts themselves. Using this vulnerability, attackers are able to deliver malicious scripts to other users through trusted sites. While Nordstrom is not primarily at risk, plenty of secondary risk from cross site scripting exists, to an extent that Nordstrom has to ensure that any cross site scripting vulnerabilities are accounted for.

Cross site scripting can occur when a webpage incorporates user input into output, in almost any fashion. Example of this include forum posts/comments (websites accept user comments, then modify the webpage to display them), search results (when websites include a “search results for *user’s search query*” term) or user forms with entries shown in a confirmation window. Attacks occur when untrusted content is submitted into dynamic content that is sent to another site user without validation for malicious content (OWASP). This threat vector can be exploited by anybody with access to input data, which can range from end users to administrators. The malicious content may be anything the browser can run; it is often Javascript, HTML, Flash, or other code (OWASP). An example of cross site scripting could be when instead of typing a forum comment, a malicious user enters the following code into a comment box:

<script>alert(“Cross site scripting vulnerability”);</script>

This code is not itself malicious, it simply opens a dialog box with the text “Cross site scripting vulnerability.” However, the code can easily be substituted between the <script> tags that help redirect the user, create fake dialog boxes, and prompt invisible downloads. <Script> tags are not the only vectors for cross site scripting; many other HTML tags can be used, with malicious content placed in the attribute fields of the elements. Due to the capability to prompt invisible downloads that the victim doesn’t realize are occurring, cross site scripting has the potential for massive amounts of harm and compromise. Typically cross site scripting is almost always seen in one of its two most common forms, persistent (also called stored) and non-persistent (also called reflected).

Persistent cross site scripting occurs when a user is able to submit content to the website that is stored or saved on the webpage for a period of time. In the previous example, if the malicious code was submitted as a forum comment and was not validated, every user that visited a page that displayed the comment would have the dialog box open in their browser. If the dialog box code was replaced with malicious code that began a download in the browser’s background, all users to view the page would receive the download.

Non-persistent, or reflected, cross site scripting attacks are more complex. This threat vector targets sites where user input can generate temporary or non-persistent changes in webpages. One example of this is search results: sometimes web pages include a clarifier such as “showing results for ‘cats’.” If the attacker were to attack with the dialog box script example seen earlier, the “Cross site scripting vulnerability” dialog box would show up on the attacker’s screen. Since this search page/injection is not persistent, however, the only way the attacker could get the same script to show up on a user’s computer is by somehow getting the victim to search “<script>alert(“Cross site scripting vulnerability”);</script>.” This can sometimes be done, depending on how the URL is designed. Some websites include the search query as a variable, which may look like the following:

www.vulnerable.com?search=cats

When the search term is included in the url, this example site becomes vulnerable to cross site scripting attacks. Attackers are able to create a URL that directs victims to the query with the malicious script, which looks like the following:

www.vulnerable.com?search=<script>alert(“Cross site scripting vulnerability”);</script>

Now, the URL directs users to the search page, and uses the “search” variable contained in the URL to run a search identical to the script injection code created by the attacker. Reflected cross site scripting attacks rely on sharing the infected links, so once the attacker creates the above URL he would then have to distribute it in an attempt to trick potential victims into following it. These attacks have the same damage potential as persistent cross site scripting, but can only infect those who click the specific link rather than all users that may navigate to a given web page.

Cross site scripting has the potential to cause severe problems, whether persistent or non-persistent. Severe attacks allow the attacker to disclose the victim’s session cookie, allowing the attacker to take over the user’s session and control the account. Attacks can also include the disclosure or tampering of the victim’s files, installation of malicious programs, redirection to a malicious or phishing site, or the modification of page content (OWASP). These problems, once again, don’t affect Nordstrom directly, but repeated attacks on its users reflect badly on Nordstrom and could potentially cause long term damage to a variety of Nordstrom assets.

Business impacts of cross site scripting can impact a variety of assets; one in particular may be company reputation. As discussed earlier, a company’s reputation can be extremely important and manifest in profit and overall success of the company. Being featured on the media for script attacks on its customers due to it’s unsecured site would be massively harmful to the future success of Nordstrom. Because cross site scripting cannot target Nordstrom directly, essentially all risk assumed by Nordstrom is secondary risk and is manifested in impacts to reputation. Some other risks include increased likelihood of fraud (due to hijacking user sessions) and loss of sales due to disrupting the victim’s shopping experience.

Although cross site scripting is different than SQL injection in many ways, their controls are extremely similar. The focus of injection controls (cross site scripting controls included) is focused on the separation of untrusted user-submitted data and active browser content (OWASP). As discussed before, a whitelist or blacklist input sanitation effort could work; however it could be prone to error, and some web forms would require the use of terms that could be confused with html tags. Most controls seek to force the web page code to behave like parameterized database query; however there is no such thing as a parameterized website, so sanitation efforts alone must maintain safety. Security encoding libraries for this purpose exist, with Microsoft .NET’s framework offering the Microsoft Anti-Cross Site Scripting Library and ASP.NET offering the ValidateRequest library (OWASP). The most important control is to religiously use good cross site scripting preventative measures when writing a web site. These measures include rules such as regularly HTML escaping all forms of user-submitted content (in both element content and element attributes), as well as Javascript escaping when possible (OWASP notes that there are some Javascript functions that can *never* safely use untrusted data, such as setInterval). Untrusted data also needs to be escaped when placed in CSS and the HTML URL parameter value (OWASP). Escaping is generally the most flexible and most used control, and works by converting certain HTML characters into other terms: for example, “<” becomes “&#60”, “>” becomes “&#62”, etc (Acunetix).

Cross site scripting controls are much less secure than SQL injection controls, because HTML and Javascript do not have the luxury of query parameterization. For this reason the escaping process must be much more carefully executed, as it is the primary and often one of the only lines of defense. Escaping and filtering (blacklisting/whitelisting) are both focused on the “Protect” function of the Cybersecurity Framework. Two more controls exist, that focus on the “Recovery” function; these controls are not adequate alone as they serve to mitigate the damage done by a successful cross site scripting attack. One is to use an HTTPOnly cookie flag on the session cookie, along with other cookies not accessed by Javascript. This is recommended by OWASP because it helps mitigate the risk of client side script accessing the protected cookie. The second control is the implementation of a content security policy (CSP): a browser-side mechanism that allows developers to “create source whitelists for client side resources of your web application, e.g. Javascript, CSS, images, etc” (OWASP). CSP instructs the browser to only execute/render information from these sources. Thus, injected code would not appear on the CSP whitelist and would not be rendered.

Cross side scripting is a very real threat to Nordstrom, with difficult controls to implement. To ensure adequate XSS protection, extensive testing and code reviews should be done to ensure that the website in its entirety is safe.

Nordstrom faces many threats to it’s success and profitability, but SQL injection and cross site scripting are definitely some of the biggest. These two threats have similar mechanics, but have different threat vectors, outcomes, and severities. Controls for both strive for the same goal, the separation of untrusted data and executable code, but have to be achieved in different ways. Nordstrom’s ability to control these threats is extremely important for future success.

This project has been a success because I have been able to explore the deeper workings of these threats, as well as incorporate them into a comprehensive model of risks surrounding Nordstrom in general with the team.

Works Consulted

Acunetix. (n.d.). Cross-site Scripting (XSS) Attack. Retrieved November 24, 2014.

Acunetix. (2011, March 22). Preventing XSS Attacks. Retrieved November 24, 2014.

Acunetix. (n.d.). SQL Injection: What is it? Retrieved November 23, 2014.

Briggs, B. (2013, February 28). The Web Represents Nearly 28% of 2012 Sales Growth for Nordstrom. Retrieved November 22, 2014.

Cook, J. (2014, February 22). With Sales Falling at Department Stores, Nordstrom Plans Bigger Online Push. Retrieved November 22, 2014.

Cross-site Scripting (XSS). (2014, April 22). Retrieved November 23, 2014, from https://www.owasp.org/index.php/Top\_10\_2013-Top\_10

Cybersecurity Framework. National Institute of Standards and Technology (2014). 1-39.

PCI FAQs. (2014, January 1). Retrieved November 21, 2014, from https://www.pcicomplianceguide.org/pci-faqs-2/

PCI SSC Data Security Standards Overview. (2013, January 1). Retrieved November 23, 2014, from https://www.pcisecuritystandards.org/security\_standards/

SQL Injection. (2014, August 14). Retrieved November 22, 2014, from https://www.owasp.org/index.php/Top\_10\_2013-Top\_10

Top 10 2013. (2014, August 26). Retrieved November 22, 2014, from https://www.owasp.org/index.php/Top\_10\_2013-Top\_10