[[1]](#footnote-1)

The Benefits of a Vulnerable ASP.NET Web Application (December 2016)

Nicholas G. Gilbert, Student, Covenant College

*Abstract* — Web based cyber-attacks are a real threat to any person or business that uses the internet. Applications like the Damn Vulnerable Web App demonstrate common web attacks including SQL injection, XSS attacks, and brute force attacks. While the DVWA is useful for demonstrating attacks on PHP applications, there is no comparable application for ASP.NET. The goal of this project is to create a DVWA-like application built with ASP.NET.

*Index Terms* — ASP.NET, MVC, Data Security, Information Security, Relational Databases, Web Design, SQL Server

# INTRODUCTION

A

uthor and web developer Peter Welch keeps a blog about his experiences building web applications. Not one to be politically correct, he refers to the internet as “its own special hellscape” from a developer’s perspective [1]. Writing a traditional program has challenges, but those challenges pale in comparison to writing web applications. Such challenges include optimizing code so web pages load quickly, maintaining compatibility for multiple, potentially outdated browsers, and scaling applications to handle thousands of users simultaneously. The challenge this project will focus on is writing secure web applications.

When building a new project, modern web developers do not start from scratch. Modern websites are built on preexisting frameworks provided by others. With choices ranging from full-featured development frameworks like ASP.NET and Ruby on Rails to Content Management Systems like Wix and SquareSpace, web developers rarely have to build a website from scratch today.

Good frameworks must handle common threats to the security of applications built with them. Those vulnerabilities can be universal, like XSS attacks and SQL injection, or vulnerabilities specific to a framework like the ASP.NET POET vulnerability which jeopardizes the confidentiality of information stored in an ASP.NET application [2].

To demonstrate the threats that web applications face, Dewhurst Security has developed the Damn Vulnerable Web App (DVWA). The DVWA is a deliberately vulnerable PHP/MySQL web application that allows penetration testers to hone their skills and developers see the kind of attacks that their applications can face. The attacks demonstrated include SQL injection, XSS attacks, command injection, file upload vulnerabilities, and brute forcing.



**Figure 1: The Damn Vulnerable Web App. This application, developed by Dewhurst Security, demonstrates vulnerabilities in web applications that developers need to be aware of when building their applications. It also allows penetration testers to hone their skills without illegally hacking applications.**

# Project Objective

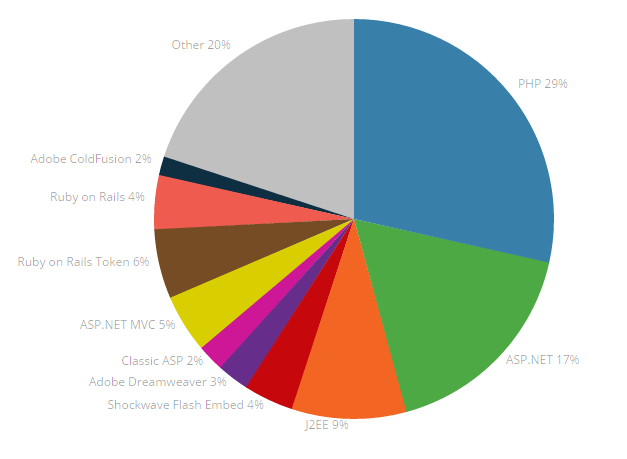
The DVWA is useful for demonstrating specific attacks that can occur against any web application. However, it does not demonstrate the role that frameworks play in the security of web applications. Only one version of the application exists; written in PHP and using a MySQL database. PHP and MySQL are commonly used together but that is not the only combination of tools used to build web applications.

The goal of this project is to provide an application like the DVWA but built using a non-PHP framework. A proof-of-concept will be given in the form of an application that demonstrates three of the most dangerous attacks and how to secure against them. Test cases will also be provided to demonstrate the improved security of an application when it is coded to security standards given by the framework provider. The project is called the Vulnerable .NET App (VDNA).

# Approach

## Choosing a Framework and Database

Ideally, a version of the DVWA would exist for every possible framework and database combination. However, writing one for every combination would go well beyond the time allotted for this project. The VDNA is built on Microsoft’s ASP.NET framework ([www.asp.net](http://www.asp.net)), the second most popular framework for web applications after PHP [3].



**Chart 1: The most popular web frameworks in 2016. The top four frameworks are based on four different languages. PHP, the most popular framework language, holds 29% of the market. ASP.NET, based on ASP/C#, holds 17%. Java’s enterprise edition is 3rd at 9% and Ruby on Rails is 4th at 10% [3].**

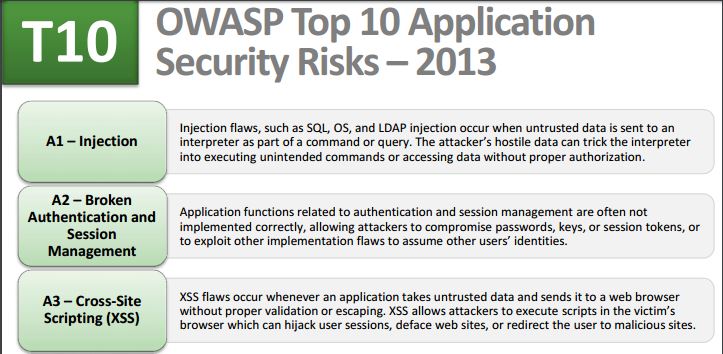
ASP.NET is compatible with most relational databases, but many developers choose to work with Microsoft’s SQL Server since they are provided by the same company and have native support for each other. Therefore, the database VDNA uses is SQL Server 2014.

## Which Attacks to Demonstrate

Web developers face numerous threats from malicious users; all of which could have been included in this project. To narrow the focus of this project, three common attacks will be demonstrated.

The Open Web Application Security Project (OWASP) monitors cyber-attacks and collects related statistics. With those statistics, OWASP compiles a list of the most common types of attacks which occur. Currently, the most common attacks are injection, cross-site scripting, and authentication [4]. Therefore, those are the three attacks demonstrated by the VDNA.

OWASP’s top ranked threat is injection. Injection is tricking an application into running malicious code or commands provided by an attacker. A common form of injection, and the form demonstrated in this project, is SQL injection. SQL injection is used by attackers to see and manipulate an application’s database in ways that were not intended by the developers. A poorly coded web application will take any input given by the user and place it in the middle of a hard-coded SQL statement which is then issued to the database. VDNA demonstrates a form weak to SQL injection, and one that guards against it with Microsoft’s security measures.



**Figure 2: The top three entries from OWASP’s list of the top ten threats to application security. Factors which go into ranking the attacks include frequency and the extent of damage that can be done by the attack. The top three attacks, and therefore the three demonstrated in this application, are Injection, Broken Authentication, and Cross-Site Scripting attacks.**

The second threat ranked by OWASP is broken authentication. Many web applications, especially ones that store financial information, implement user accounts to grant everyone access to their own information (and only their own information). Unfortunately, that information is often accessible to users who know how to trick an application into revealing information they shouldn’t be able to. A form of broken authentication, and the form demonstrated by VDNA, is the ability to brute-force one’s way into a user’s account. If a login screen has no preventative measures and a user chooses a weak password, an attacker can brute force their way into the user’s account.

Finally, the third threat to web applications ranked by OWASP is cross-site scripting (XSS) attacks. XSS attacks are similar to SQL injection as they trick an application into executing outside code. Unlike SQL or command injection, where code is processed on the backend of an application, XSS attacks occur on the frontend of an application and can be used to alter its appearance and content.

Typically, an attacker will write a script and feed it into the application somewhere that it accepts user input. If the application renders HTML based on that input, and the input is not filtered, a user can inject <script> tags to run JavaScript in the application.

There are two types of XSS attacks. Stored XSS is where the results of an attack are not immediately visible to the user but the application is altered internally by the injected script. The other type, and the type demonstrated by the VDNA, is reflected XSS attacks. Reflected attacks allow the malicious user to see the results of a script injection. The type of XSS an application is vulnerable to depends on how it processes user input

Injection, broken authentication, and XSS attacks are three dangerous threats to the security of web applications. Since they are ranked so high by OWASP, they are the three attacks demonstrated in this project.

# Features

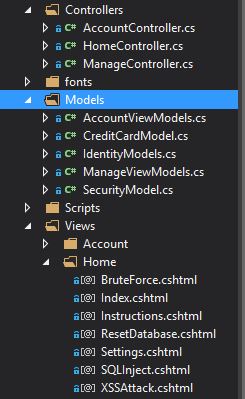
## MVC Model

Given all the challenges web developers face, the code base for a web project can become complex; hindering development as a result. In the face of that problem, the philosophy of Model-View-Controller (MVC) has risen [5]. ASP.NET provides a template geared toward that philosophy which was used to build the VDNA.

The model of the application represents the records stored within the database. The model is made up of C# classes representing the tables of the database. Each instance of those classes is a record stored in the corresponding table. Working with models bypasses the need for user-declared SQL connections which makes applications more secure.

The views of the VDNA are the web pages rendered to the user. Views display information from the model and provide the interface through which the user can provide input to the controller.

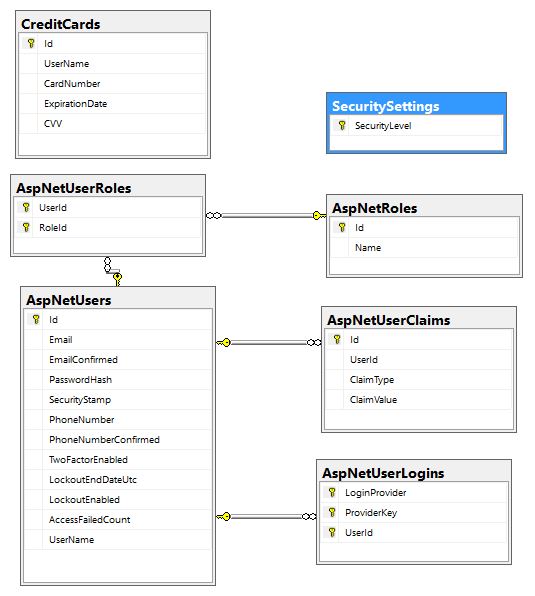
The VDNA’s controllers take user input and operate on the models based on that input. The logic for the attack pages, as well as generating and manipulating the database, is contained within the controllers.



**Figure 3: The file structure for the VDNA with the MVC files shown. The model is made up of C# classes correlating to the SQL Server database tables. The views display the models to the user and provide the interfaces through which users feed input to the controllers.**

## Database Structure

The SQL Server 2014 database is modeled for Microsoft’s ASP.NET Identity library. ASP.NET Identity provides the required infrastructure for user accounts in ASP.NET applications. Each user account is uniquely identified by its ID which acts as the primary key in most of the other tables.



**Figure 4: The database diagram for the SQL Server 2014 database used by the VDNA. The database is normalized to BCNF and provides the tables necessary for the application to have user accounts. It also provides the two security level options for the application. Finally, it lets users store credit card data which acts as sample data for the attacks.**

The database is normalized to BCNF. Each user account has a username (which is an email address) and password with which they can log in. The SecuritySettings table stores two strings, “low” and “high”, to tell the attack pages how difficult to make each attack. An admin user is created by default and every other registered user has user-level permissions.

## Attack Pages

The views provided by the VDNA consist of instruction pages, a home page that provides a starting point, and the attack pages. Each attack page provides an explanation of what attack the page demonstrates and a realistic scenario through which to demonstrate it.

In accordance with the philosophy of MVC, the views show the appropriate models to the user depending on the page they are on. The views also provide an interface through which the user can conduct the attacks..

## Security Level

Finally, the VDNA provides two different security levels. The low setting tells the application to execute insecure code to process the input given to each view. The low setting represents an ASP.NET application which is poorly developed from a security perspective. Test cases are provided that successfully exploit each attack when the security is set to low.

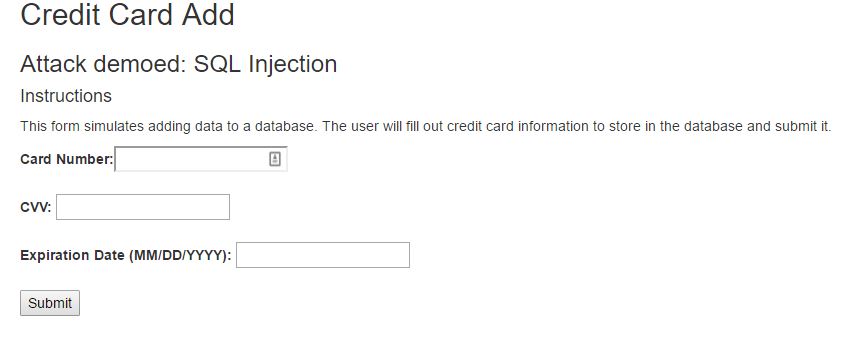
On the other hand, the high security setting tells the application to use code written to Microsoft’s standards of security [6]. Theoretically, only zero-day attacks should succeed against the attack pages when the security level is set to high.

# Collecting Data

## SQL Injection

The greatest threat to web applications as ranked by OWASP, injection, is demonstrated by the VDNA in the form of SQL injection. The SQL injection view displays a form that lets the signed-in user add a credit card to associate with their account.

If the security is set to low, a SQL connection object is used in the controller to process the user’s input. The input is not checked beforehand, leaving the application vulnerable to SQL Injection. If the security is set to high, a DBContext object is used to update the database through the credit card model.



**Figure 5: The SQL Injection demo view. This form allows a user to associate a credit card with their account.**

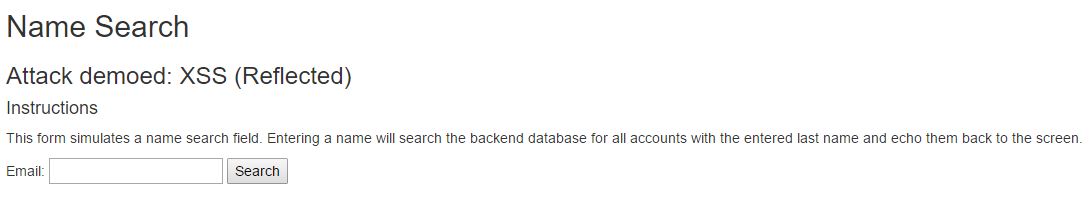
The test cases in table one are used to demonstrate the benefit of using DBContext over a SQL connection.

|  |  |  |
| --- | --- | --- |
| **Input** | **Result (Low)** | **Result (High)** |
| 1111222233334444 | No Attack | No Attack |
| 1111222233334444' -- | Invalid SQL Error | Dangerous Input Error |
| 1111222233334444', '111', '20160801'); -- | Record inserted  Attack successful | Dangerous Input Error |
| 1111222233334444', '111', '20160801'); DROP TABLE CreditCards; -- | Record inserted, table dropped  Attack successful | Dangerous Input Error |

**Table 1: The test cases given to the SQL Injection view and the result for each case.**

## Cross-site Scripting (XSS)

OWASP’s second-highest threat, XSS attacks, are demonstrated by the VDNA in the form of reflected XSS attacks. The XSS attack view displays a form which allows the signed-in user to search for other registered users. If the security level is set to low, the form is vulnerable to XSS attacks. If it is set to high, the controller behind the view filters input for HTML tags.



**Figure 6: The Reflected XSS demo view. This form allows a user to search for other registered users in the application.**

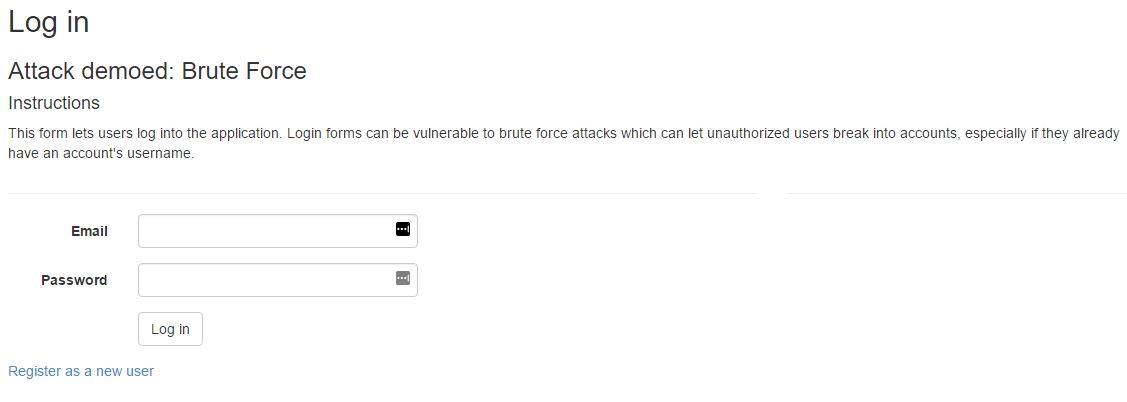
The test cases in table two are used to demonstrate the benefit of filtering user input for dangerous input before it is processed.

|  |  |  |
| --- | --- | --- |
| **Input** | **Result (Low)** | **Result (High)** |
| nick | No Attack | No Attack |
| <script>alert(“XSS”) </script> | Alert box shown  Attack successful | < > stripped  Attack failed |
| nick <script> alert(“XSS”) </script> | Search for nick  Alert box shown  Attack successful | < > stripped  Attack failed |

**Table 2: The test cases given to the XSS Attack view and the result for each case.**

## Brute Force Authentication

Finally, the third threat ranked by OWASP is poor implementation of authentication. To demonstrate this vulnerability, the VDNA has a login form which is vulnerable to brute-forcing login attempts. When the security level is set to low, the administrator password is “mockingjay”; a weak password. When the security level is high, the admin password is “q1S2@fXZETG9433!2”; a strong password rendering brute-forcing impossible.



**Figure 7: The broken authentication demo view. This login form is vulnerable to brute-force attacks.**

It is typically a last resort, but when hackers decide to take a brute-force approach, a strong password is the best defense. Malicious users have multiple tools at their disposal to brute-force login forms. The tool used to demonstrate the brute-force attack is THC-Hydra. The included bash script Hydra.sh can be run on any Linux machine with THC-Hydra installed. The results of the script running against the VDNA are shown in table three.

|  |  |
| --- | --- |
| **Security Level** | **Time to Brute-Force** |
| Low |  |
| High | Attack failed |

**Table 3: The included script Hydra.sh executes an attack against the VDNA using THC-Hydra, a brute-force attack tool. This table shows what happens when security is set to low compared to high.**

# Faith Integration

The VDNA demonstrates three of the biggest threats to web applications and how to guard against them. However, an optimistic developer might assume that users of the applications they write will not try to attack them. From a Reformed perspective, it is a simple point to argue against. The Reformed tradition and most evangelical Christians recognize the fallen nature of man. Genesis 2-3 narrates the story of Adam and Eve’s decision to take of the Tree that God commanded they not take from. As the first humans, their original sin corrupted all of humanity, meaning all people are sinners by birth, as well as by deed for “all have sinned and fallen short of the glory of God” (Romans 3:23, ESV).

Even though the world is a fallen one, the optimistic developer is correct in a limited way. The majority of users of an application are not going to try and attack it. Some of that is due to God’s common grace manifested in the world. Even though the Reformed tradition reserves salvation for God’s elect, King David wrote that “the Lord is good to all, and his mercy is over all that is made” (Psalm 145:9, ESV). A user is likely not going to hack their banking website for the same reason that non-Christians can still do good deeds; God’s common grace.

Where the optimistic developer’s logic falls short is failing to take into account the limits of God’s common grace, self-evident by the fact that sin is still in the world. Man is still sinful “for out of the heart come evil thoughts, murder, adultery, sexual immorality, theft, false witness, [and] slander” (Matthew 15:19). The need for computer security comes from the reality that man is sinful despite God’s common grace. The virtual world is not a distinct world from the physical. Cyberspace exists as an extension of the physical world. Cyberspace exists from all the servers, wires,

Finally, it is worth noting what a Christian response to technology in general should look like. The answer is not to shun all technology as sinful.

Virtual world is extension of the physical world. (God created out of nothing but everything else is created out of what God created) (Colossians 1:16)

Christian answer is not to run from technology (Proverbs 30:19)

# Conclusion

References

1. P. Welch, "Programming Sucks," in Still Drinking, 2014. [Online]. Available: https://www.stilldrinking.org/programming-sucks. Accessed: Nov. 28, 2016.
2. Microsoft, "Microsoft security bulletin MS10-070 - important," 2010. [Online]. Available: https://technet.microsoft.com/library/security/ms10-070. Accessed: Nov. 28, 2016.
3. "Framework technologies Web Usage Statistics", *Trends.builtwith.com*, 2016. [Online]. Available: http://trends.builtwith.com/framework. [Accessed: 06- Apr- 2016].
4. J. Williams and D. Wichers. OWASP top 10 2013: The Ten Most Critical Web Application Security Risks. *OWASP Foundation*, April 2013
5. Microsoft, "Model-view-controller". [Online]. Available: https://msdn.microsoft.com/en-us/library/ff649643.aspx. Accessed: Nov. 30, 2016.
6. Microsoft, "Basic security practices for web applications". [Online]. Available: https://msdn.microsoft.com/en-us/library/zdh19h94.aspx. Accessed: Dec. 1, 2016.

1. This paper was submitted for review on December 5, 2016 to Dr. Jeffrey Humphries, Associate Professor of Computer Science at Covenant College in Lookout Mountain, GA. This paper serves as the Bachelor’s Thesis of Nicholas G. Gilbert, fulfilling the requirement of COS492: SIP in Computer Science

   Nicholas G. Gilbert is an undergraduate student studying Computer Science at Covenant College, 14049 Scenic Highway Box 222, Lookout Mountain, GA 30750 USA (email: nickgilbert1994@gmail.com). [↑](#footnote-ref-1)