Evaluating the Security Risks of Static vs. Dynamic Websites

Ballard Blair
Comp 116: Introduction to Computer Security
Professor Ming Chow
December 13, 2017

Abstract

This research paper aims to outline the security vulnerability differences in static and dynamic web applications. It will define how each type of web app works and in what ways a static website in particular is susceptible to being hacked. This paper aims to inform the reader about what defense measures should be taken to defend against attacks. More specifically, this paper will attempt to provide software developers with a clear understanding of what security vulnerabilities could affect their static website and how to provide their users with a secure web browsing experience.

Introduction

Web sites play a vital role in our day to day lives. We learn through them, interact with them, click them, block them, hate them and love them. It is important to understand the state of security behind these applications we trust with our some of our most sensitive data.

In order to fully understand the security risks of web applications in the modern day, it is imperative we understand the differences between static and dynamic web sites. A static site is one where the content can be delivered to the user without being modified, generated or processed: static content does not change. A user sends a request for a webpage the server, the server fetches the webpage and sends it back to the user. On a static web page, the content does not change in response to a user's actions or inputs.

Different from a static site, a dynamic web page contains information that changes depending on the user, user's input, the user's native language, the time of day, ect. A dynamic site can use either client-side scripting or server-side scripting or a combination of both to

manipulate elements on the webpage. A dynamic site provides an interactive user experience because of its ability to respond to the user in different contexts and conditions.

Both static and dynamic web pages are susceptible to security vulnerabilities. CVE, the industry standard for vulnerability and exposure identification, defines a vulnerability as "a weakness in the computational logic (i.e. code) found in software and some hardware components that, when exploited, results in a negative impact to confidentiality, integrity, or availability." [1]

To the Community

There is a dangerous misconception that static websites are not at risk to security vulnerabilities. At first glance, a static site has no obvious attack vectors where hackers could inject malicious code. It merely contains HTML, CSS files and maybe a bit of JavaScript. There is no need for a server, no need to store sensitive data and no need to deal with user input. Even some coding blogs advertise static sites as being totally secure. "As static files served to the browser, there are no actual vulnerabilities." [2] "With a static site, you don't have to worry about malicious code being injected into your site when users visit it." [3] While static sites may not be vulnerable to the number of attacks or the same type of attacks as a dynamic site, a static web site is not impervious to cyber-attacks.

The state of software security for all types of web applications is grim to say the least. According to Veracode's 2017 State of Software Security report, the OWASP Top Ten pass rate has consistently been below one-third of applications for the past five reports. [4] Software developers have been making a lot of the same mistakes for over the past ten years. Many of the critical security flaws outlined by the OWASP Top Ten in 2004 remain the same in 2017:

Cross Site Scripting and Injection Flaws. [5][6] Cross Site Scripting (XSS) allows an attacker to execute scripts in the victim's browser and hijack user sessions, deface web sites, or even redirect the victim to malicious sites. XSS can be mitigated by sanitizing user input and removing the ability for data to be interpreted as code. SQL injection, one of the most notable types of injection flaws, allows an attacker to bypass login credentials and access other sensitive data without proper authentication by using SQL queries via input data. Developers can defend against SQL injection attacks by filtering out all special characters in user input, using prepared statements, and limiting what privileges a database has access to. [7] Both XSS and injection flaws are two of the most common vulnerabilities found in dynamic sites because they stem from malicious user input being improperly validated.

A strictly static site is not vulnerable to the common attack vectors found with dynamic sites. The app does not handle user accounts or a database with sensitive information due to its lack of interactive elements. Nonetheless, developers should take measures to ensure that both data being transferred and site access is encrypted.

Action Item: Static Site Attack Vectors and Mitigation Techniques

Insecure Service Passwords

Whenever a change is rolled out for a web application, those files must be uploaded to the hosting server whether by manually uploading to the server through a hosting control panel or through FTP. Whatever method is used to upload files, access to the file structure of the web app is username and password protected. This is the first place where the site's security is at risk. People are terrible at picking good passwords. In a report put out by TeleSign, 47% of

people use passwords that are at least five years old. [8] It's shocking to see how the most common passwords used has not changed in the past ten years. In 2007, some of the most common passwords used were: password, 123456 and qwerty. [9] In 2016, we still see these passwords in the top ten list. [10] A 2015 information security report done by i-Sprint Innovations and eGov Innovation cited that weak authentication security accounts for 76% of data breaches and compromised records. [11] If an attacker were to gain access to the root file structure, they then could then deface the website, redirect traffic to malicious sites, or cause a whole host of head-ache inducing issues. To defend against this attack, the server username/password should never be communicated through email, text or messenger apps and the site owner should use a strong, complex password.

Insecure File Transfer Methods

Mentioned above, files can be uploaded to the server through the file transfer protocol or by remotely connecting to the server through RDP, SSH, or even Telnet. Upon reading the RFC for FTP, it is apparent that it was never designed with security in mind. It was originally designed to make sharing files easy for both users and programmers. [12] All of the data sent over FTP and Telnet is unencrypted ("in the clear"), meaning anyone sniffing the webmaster's network could steal any usernames, passwords, personal information or other sensitive information transferred over the network. RDP and SSH though are encrypted by default, making them better options for file upload. In addition, SCP should be used instead of FTP as well as SSH instead of Telnet. To mitigate the risk of having files or information leaked, encrypted network protocols should *always* be used when uploading files to a server.

HTTP vs. HTTPS vs. HSTS

Google Chrome now displays a warning for any webpage using the HTTP protocol instead of HTTPS, but is it really necessary for every web application? (Short and long answer: yes) HTTPS ensures the privacy and integrity of transmitted data by using secure protocols to encrypt communications. HTTPS used to use either Secure Socket Layer (SSL) or Transport Layer Security (TLS) to secure communication but recently TLS has become the standardized protocol. TLS/SSL use techniques known as asymmetric Public Key Infrastructure (PKI) and symmetric cryptography. PKI uses two 'keys' to encrypt communications, a 'public' key and a 'private' key. Anything encrypted with the public key can only be decrypted by the private key and vice-versa. [13] When SSL/TLS is used correctly, a network sniffing attacker or an ISP can only see which IP and port the user is connected to, a rough figure of how much data is being sent, and what encryption/compression method is being used in contrast to HTTP connections where all data is sent in 'plain text'.

SSL/TLS comes into the picture when an HTTP request is sent to a webpage and then the SSL handshake is started by the client (which could be a browser or another program such as a Mac OS update or Xshell). The client also sends a bit of information such as which version of SSL/TLS it is running, what ciphersuites it wants to use and what compression methods it wants to use. The server then checks what version of SSL/TLS will be supported by both parties, picks a ciphersuite from the client's options and optionally picks a compression algorithm. The server then sends back its identification via a digital certificate. This certificate must be trusted by the client itself or a 3rd party the client trusts. Once the client confirms the validity of the certificate, both the server and client can compute the session key for symmetric encryption. Once the key

is securely computed, all communication between the client and server will be encrypted. [14] Enabling HTTPS on a site is relatively simple process. Digital certificates require that your website has its own dedicated IP address, this ensures that all traffic going to that IP is only going to your website and no one else's. The site owner must then get a digital certificate from a Certificate Authority (CA) by purchase or through free services such as Lets Encrypt. The CA verifies that your web address belongs to your organization, thereby protecting site users from man-in-the-middle attacks. The certificate can then be activated and all traffic to your site will be encrypted. [15]

If HTTPS is being used, it makes a lot of sense to also enforce HTTP Strict Transport Security (HSTS). This security measures helps enforce that users only use HTTPS to connect to your web application, mitigating man-in-the-middle attacks and SSL stripping. HSTS works through the use of a special response header. Once a browser sees this header, the browser will enforce that all communication for the specified domain is sent over HTTPS. [16]

Content Security Policy

A Content Security Policy is an added layer of security that was originally created to defend against XSS and other injection-type attacks by controlling what resources are allowed to load for a certain page. Instead of blindly trusting everything that a server delivers, CSP defines the Content-Security-Policy HTTP header which allows the site owner to create a whitelist of trusted sources. A CSP is not necessarily applicable if no external sources are loaded for a purely static site. However, it is not uncommon for a static site to include external links to a framework CDNS, icon libraries, fonts, images, or ads. If there is ever linked external content

outside the control of the webmaster, it makes sense to restrict which resources the application can render. [17]

Conclusion

Although static sites are not vulnerable to nearly as many attacks as a dynamic site, it remains imperative that software developers provide users with a secure web experience. We live in an age where the web interacts with more and more sensitive data on a daily basis and where companies want to gather endless data on their users. I believe that along with making a user experience that is easy-to-use and nice looking, security *must* be prioritized. Developing software with security in mind provides key layers of protection for users: encryption, data integrity, and authentication. By showing your users that nobody can sniff their network or that the integrity of their data is important, the organization builds a trustworthy relationship with its clients. Software developers owe it to their users to provide encrypted access – no matter what content is being delivered.

References

[1] https://cve.mitre.org/about/terminology.html
[2] https://medium.freecodecamp.org/static-sites-are-back-24d01a01f11a
[3] https://www.netlify.com/blog/2016/05/18/9-reasons-your-site-should-be-static/
[4]https://info.veracode.com/state-of-software-security-developer-supplement-report-veracode-resource.html
[5]https://www.owasp.org/index.php/Top_10_2017-Top_10
[6]https://www.owasp.org/index.php/Top_10_2004
[7] https://www.veracode.com/blog/intro-appsec/sql-injection-attacks-and-how-prevent-them-infographic
[8]https://blog.hubspot.com/marketing/password-statistics
[9] https://www.pcmag.com/article2/0,2817,2113976,00.asp
[10] https://keepersecurity.com/public/Most-Common-Passwords-of-2016-Keeper-Security-Study.pdf
[11] http://resources.infosecinstitute.com/password-security-complexity-vs-length/#gref
[12]https://www.giac.org/paper/gsec/748/ftp-security-hole-about/101645
[13]https://tuftsdev.github.io/DefenseAgainstTheDarkArts/slides/week2-cryptography.pdf
[14]https://security.stackexchange.com/questions/20803/how-does-ssl-tls-work
[15] https://support.google.com/webmasters/answer/6073543?hl=en
[16]https://www.owasp.org/index.php/HTTP_Strict_Transport_Security_Cheat_Sheet
[17] https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP