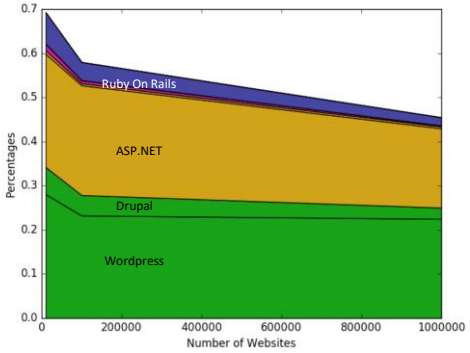
Insecure by Default?

Authentication Services in Popular Web Frameworks

Hannah Li and David Evans, *University of Virginia*

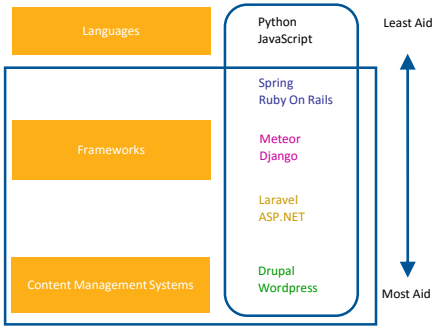
Motivation

According to BuiltWith.com, 70% of Quantcast's top 10K websites are built using one of the 8 most popular frameworks. Hence, we hypothesize that the aid provided by these frameworks has a major impact on the security of many websites. This work studies how different design choices made by web frameworks impact the security of web applications built by typical developers using those frameworks.



Scope

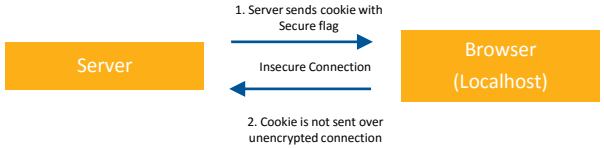
We focus on server-side frameworks or content management systems that provide default authentication services to help developers get started. Each of the three levels of aid – template, tutorial, libraries– from the framework expect increasing knowledge and expertise from the developer. For now, we do not consider third-party tutorials or modules which may be helpful additions to many of these frameworks.



Objectives and Preliminary Findings

Long Term Goals.
We seek to understand the usability and performance trade-offs that lead frameworks to adopt insecure defaults, and develop alternatives that lead to better security without sacrificing the needs of easy initial development and deployment. We seek to identify factors that make a framework less secure than is possible following best known practices, and to understand why framework designers choose less security options.

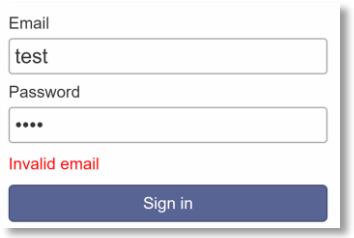
Security vs. Ease of Development.
While most frameworks turn the HTTP-Only cookie flag on by default, the Secure flag has to be set manually. This is undesirable for security, but convenient from a development perspective because most websites are developed and tested using localhost. With the Secure flag turned on, cookies would not work because localhost does not have encrypted connection.



Security/Usability Tradeoffs.
Frameworks also make decisions that may not be perceived as insecure, such as the choice of login error message and whether to include brute-force protection and login failure logging. For example, the template and code example respectively for Wordpress and Meteor give "Incorrect username/password" and "inactive account" errors.

While such specific error messages help the developer and user distinguish between non-existent user accounts and invalid passwords, accepted security practices encourage hiding this information for potential attackers.

Other examples include brute-force protection and login failure logging, which are not widely adopted in the examined frameworks, although they have significant security benefits with minimal downsides. On the other hand, CRSF protections appear to be provided by all frameworks when applicable.



Framework	Wordpress	Drupal	ASP.NET	Laravel	Django	Meteor	Ruby On Rails	Spring
Framework Aid	CMS	CMS	Template	Template	Tutorial	Tutorial	Library	Library
Password Encryption	MD5	SHA512	PBKDF2	Bcrypt	PBKDF2	Bcrypt	Bcrypt	Bcrypt
Password Reset	On by default	On by default	Off by default	Code examples	Code examples	Code examples	None	None
Credential Error Msg	Leaks by default	Hidden by default	Hidden by default	Hidden by default	Hidden in code examples	Leaks in by default	Hidden in description	None
Bruteforce protection	Described	On by default	Off by default	On by default	Code examples	None	Described	Described
Login failure logging	None	On by default	Off by default	None	Code examples	None	Described	None
CSRF Token	URL Nounce	On by default	On by default	On by default	Code examples	Uses Localstorage, not cookies	Code examples	Code examples
Percent of Sites (10K)	28	6	26	0.8	1.5	0.03	7.1	0.03

On by Default: **On by default** and **off by default** describe features provided by frameworks that are ready-to-use out of the box.
Off by Default: The features that are **off by default** are turned off in the default version unless the developer manually turns them on.
Code examples: **Code examples** signifies that a framework provides snippets of code for the developer to reference, but it's ultimately up to the developer to correctly implement the feature.
Described: Frameworks remind the developer of security features by **describing** them in prose.
None: Even the frameworks with features labeled **none** offers at least a low-level authentication module for the developer.