Investigação do Impacto de Frameworks de Desenvolvimento de Software na Segurança de Sistemas Web

Isadora Garcia Ferrão[1], Douglas D. J. de Macedo[2], Diego Kreutz[1]

[1]Laboratório de Estudos Avançados (LEA)
Universidade Federal do Pampa (UNIPAMPA)

[2]Departamento de Ciência da Informação (CIN)
Universidade Federal de Santa Catarina (UFSC)

isadora-gf@hotmail.com, douglas.macedo@ufsc.br, kreutz@acm.org

Abstract. *Vulnerabilities are a critical and recurring issue on Web applications. Recent studies indicate that old vulnerabilities, such as SQL injection, are still a major concern. In this paper, we evaluate the impact of software development frameworks on securing Web applications. To detect security issues, we use vulnerability scanners on the top seven PHP frameworks used by developers. In each framework, we implemented the ten most frequent vulnerabilities found in Web applications. Our findings show the impact of each PHP framework on securing a Web application. In addition, the results also allow us to identify strengths and weaknesses of the vulnerability scanners.*

Resumo. *Aplicações Web vulneráveis são um problema crítico e recorrente. Estatísticas recentes indicam que mesmo vulnerabilidades antigas, como SQL injection, ainda representam um problema frequente. Este trabalho tem como objetivo avaliar, utilizando scanners de vulnerabilidades, o impacto de frameworks de desenvolvimento de software na segurança de sistemas Web. Para a análise, foram selecionados os sete principais frameworks utilizados por desenvolvedores PHP e implementadas as dez vulnerabilidades mais recorrentes em sistemas Web. Os resultados permitem identificar o impacto de cada framework na segurança do ambiente controlado. Além disso, pode-se observar também algumas qualidades e deficiências dos scanners.*

1. Introdução

Com a popularização e a constante evolução da Internet, onde a maioria dos sistemas passa a fazer parte e depender da rede, as boas práticas de segurança em sistemas *Web* são cada vez mais imprescindíveis. Relatórios recentes reportam que o número de ataques e incidentes de segurança, muitos deles envolvendo vulnerabilidades antigas e recorrentes em sistemas *Web*, não para de crescer [Symantec 2017]. Ao mesmo tempo surpreendente e muito preocupante, alguns estudos mostram que, em alguns países, a porcentagem de aplicações *Web* vulneráveis pode chegar na casa dos 70% [Alam et al. 2015].

Um dos tipos de recursos mais frequentemente utilizados na prática para detectar e diagnosticar diferentes tipos e níveis de falhas de segurança em sistemas *Web* são os *scanners* de vulnerabilidades [Bau et al. 2010, Fonseca et al. 2014, Alam et al. 2015, Ferrao and Kreutz 2017, Rocha et al. 2012]. Estudos recentes investigam a eficácia de *scanners* de vulnerabilidades em ambientes abertos (e.g. máquina virtual OWASP BWA) [Ferrao and Kreutz 2017] e ambientes controlados (e.g. implementação específica das dez vulnerabilidades mais recorrentes em sistemas *Web*) [Ferrao et al. 2018b,

Melchior et al. 2018]. Além de investigar a eficácia de ferramentas de varredura gratuitas, os estudos investigam *scanners* oferecidos como serviço (SaaS) por empresas especializadas em segurança [Melchior et al. 2018]. Os resultados apontam para um cenário muito preocupante. Primeiro, os dados apresentados reforçam as conclusões de estudos mais antigos, como o fato de ser difícil de escolher as ferramentas de varreduras mais adequadas para um determinado cenário. Segundo, apesar da evolução das tecnologias de desenvolvimento de sistemas *Web* e dos *scanners* de vulnerabilidades, fica evidente a necessidade de múltiplas ferramentas para atingir um bom nível de cobertura das vulnerabilidades existentes. Terceiro, os *scanners* SaaS alcançaram um desempenho melhor em relação a *scanners* tradicionais gratuitos. Entretanto, apesar disto, esses serviços online também deixam a desejar, atingindo índices globais (de um grupo de *scanners*) de cobertura de apenas 70% das vulnerabilidades implementadas num cenário controlado.

Dando continuidade a pesquisas recentes, como [Ferrao and Kreutz 2017], [Ferrao et al. 2018b], e [Melchior et al. 2018], este trabalho tem dois objetivos principais. O primeiro é analisar a eficácia de *scanners* de vulnerabilidade em ambientes controlados e implementados com o suporte de *frameworks* de desenvolvimento *Web*. Já o segundo objetivo é investigar o impacto dos *frameworks* na segurança dos sistemas. A exemplo, no trabalho [Ferrao et al. 2018b] os autores investigam a eficácia dos *scanners* utilizando um ambiente controlado contendo as dez vulnerabilidades mais recorrentes em sistemas *Web*. Pergunta: *Implementando as mesmas dez vulnerabilidades, utilizando diferentes frameworks de desenvolvimento Web, vai alterar os resultados de detecção das ferramentas de varredura?*

Os *frameworks* de desenvolvimento de software se popularizaram junto à comunidade por causa de atrativos benefícios, como agilidade, produtividade, padrões de projeto de forma simples e descomplicada e maior qualidade ao produto final. No que diz respeito à segurança, os *frameworks* utilizam arquivos de configuração com escopo limitado e bem definido, bibliotecas específicas, e mecanismos adicionais como filtros e validação automática de campos de formulários. Apesar destes recursos, cujo objetivo é simplificar o trabalho dos desenvolvedores e promover o desenvolvimento de sistemas mais seguros, através de pesquisas e observações da literatura existente, pode-se constatar que há uma falta de evidências científicas e/ou empíricas sobre o impacto de *frameworks* de desenvolvimento na segurança de sistemas *Web*. Os resultados de investigação e as discussões apresentadas neste trabalho mostram o impacto de *frameworks* no processo de garantir a segurança de aplicações vulneráveis, ou seja, aplicações contendo falhas explícitas de segurança.

As contribuições deste trabalho podem ser resumidas em: (i) identificação dos *frameworks* mais utilizados na prática para o desenvolvimento de aplicações *Web* baseada em PHP; (ii) implementação das dez vulnerabilidades mais recorrentes em sistemas *Web* em cada um dos *frameworks* selecionados; (iii) execução e avaliação de dez *scanners* de vulnerabilidades para analisar o impacto dos *frameworks* na segurança de sistemas *Web*; e (iv) discussão e comparação dos resultados com trabalhos anteriores, como [Ferrao et al. 2018b].

O restante do artigo está organizado como segue. A Seção 2 detalha as etapas de desenvolvimento do trabalho. Os resultados são discutidos na Seção 3. A Seção 4 é dedicada aos trabalhos relacionados. Por fim, são apresentadas as considerações finais e trabalhos futuros na Seção 5.

2. Desenvolvimento

A primeira etapa do trabalho consiste na escolha, instalação e estudo dos *scanners*. De forma similar a estudos recentes [Ferrao and Kreutz 2017, Ferrao et al. 2018b], utilizando os mesmos critérios de seleção, foram selecionados *scanners* gratuitos disponíveis na Internet, incluindo Uniscan, Paros Proxy, Zed Attack proxy, Nessus, Andiparos, Grabber, Wapiti, Ratproxy, Skipfish e Vega.

A segunda etapa inclui a escolha, instalação e configuração dos *frameworks* de desenvolvimento de software. Os critérios de seleção foram o fato de serem ferramentas gratuitas e de código aberto, suporte a PHP e sites de classificação [INDIATESTBOOK 2017]. Com base nestes critérios foram selecionados os *frameworks* Laravel, Phalcon, Codeigniter, Yii, Zend, Symfony e Cakephp.

A terceira etapa consiste na implementação das dez vulnerabilidades mais recorrentes em sistemas *Web* em cada um dos *frameworks*. Esta é a etapa mais trabalhosa uma vez que inclui a compreensão teórica e prática de cada uma dos *frameworks* para a implementação das dez vulnerabilidades.

Por fim, a última etapa consiste nos testes dos dez *scanners* em cada um dos sete *frameworks*. O tipo mais comum de testes para sistemas *Web*, empregado na prática pela maioria dos atacantes, é o *black-box*, também utilizado neste trabalho.

Mais detalhes sobre o desenvolvimento e o cenário controlado, incluindo detalhes de exploração de vulnerabilidades e implementação, podem ser encontrados em [Ferrao et al. 2018a]. As dez vulnerabilidades mais recorrentes em sistemas *Web* são apresentadas em [OWASP 2017].

3. Resultados

A Tabela 1 sumariza os resultados obtidos com os testes *black-box* nos cenários controlados utilizando sete frameworks PHP. Os dez *scanners* tradicionais são identificados da seguinte maneira: [T1] Andiparos, [T2] Nessus, [T3] Raproxy, [T4] Uniscan, [T5] Wapiti, [T6] Grabber, [T7] Paros Proxy, Skipfish, [T9] Vega e [T10] Zed Attack proxy. Na tabela constam os *frameworks*, as vulnerabilidades detectadas e quais *scanners* as detectaram.

Os *scanners* encontraram 70% das vulnerabilidades implementadas nos *frameworks* Phalcon e Codeigniter, garantindo as duas últimas posições em termos de mecanismos e recursos padrão para o desenvolvimento de aplicações *Web* seguras, como pode ser observado na Tabela 1. Vale ressaltar que as mesmas dez vulnerabilidades foram implementadas em todos os sete *frameworks*. O Phalcon foi o o único *framework* suscetível às vulnerabilidades de quebra de autenticação e gerenciamento de sessão e quebra de controle de acesso, ambas identificadas pelo *scanner* Zed Attack Proxy [T10]. Assim como a maioria dos demais *frameworks*, o Phalcon mitiga ataques de injeção de código de forma automática. Entretanto, a maioria dos riscos de segurança precisam ser tratados manualmente pelos desenvolvedores e/ou administradores dos sistemas.

Os *scanners* detectaram a vulnerabilidade *cross site scripting* no Codeigniter, bem como em outros *frameworks*. No caso do Codeigniter, realizando uma análise técnica manual, o *framework* possui um método chamado `xss clean` dentro da classe `Security`. O `xss clean` é utilizado para tratamento dos dados contra ataques XSS. Porém, a aplicação do método vem desativado por padrão no Codeigniter. É sabido que a maioria dos programadores utiliza *frameworks* de desenvolvimento de software em modo padrão. Portanto, há duas alternativas. A primeira é conscientizar os desenvolvedores e a segunda é forçar a habilitação do `xss clean` no modo padrão do *framework*. Já

Tabela 1. Vulnerabilidades detectadas no cenário controlado

Framework	Vulnerabilidades detectadas	Web scanners	% Vuln.
Laravel	*Cross site scripting*	[T2,T6,T9,T10]	40%
	Má configuração de segurança	[T10]	
	Exposição de dados sensíveis	[T7,T9,T10]	
	Componentes com vulnerabilidades conhecidas	[T2]	
Symfony	Injeção de código	[T1,T2]	40%
	Má configuração de segurança	[T1,T7,T8]	
	Exposição de dados sensíveis	[T8]	
	Cross site requesty forgery (CSRF)	[T2]	
Yii	Injeção de código	[T1,T2,T7]	50%
	Cross site scripting	[T9,T10]	
	Má configuração de segurança	[T1,T7,T9]	
	Cross site requesty forgery (CSRF)	[T2]	
	Componentes com vulnerabilidades conhecidas	[T2]	
Zend	Injeção de código	[T1,T2]	50%
	Cross site scripting	[T1,T2,T6,T9,T10]	
	Exposição de dados sensíveis	[T1,T7]	
	Cross site requesty forgery (CSRF)	[T2]	
	Componentes com vulnerabilidades conhecidas	[T2]	
Cakephp	Injeção de código	[T2]	60%
	Cross site scripting	[T9,T10]	
	Má configuração de segurança	[T7]	
	Exposição de dados sensíveis	[T9]	
	Cross site requesty forgery (CSRF)	[T2]	
	Componentes com vulnerabilidades conhecidas	[T2]	
Codeigniter	Injeção de código	[T1,T2,T7]	70%
	Cross site scripting	[T2,T7,T9,T10]	
	Má configuração de segurança	[T1,T6,T7,T8]	
	Exposição de dados sensíveis	[T4,T8,T9]	
	Falta de proteção contra ataques	[T8]	
	Cross site requesty forgery (CSRF)	[T2]	
	Componentes com vulnerabilidades conhecidas	[T2]	
Phalcon	Quebra de autenticação e gerenciamento de sessão	[T10]	70%
	Cross site scripting	[T9,T10]	
	Quebra de controle de acesso	[T10]	
	Má configuração de segurança	[T7]	
	Exposição de dados sensíveis	[T9]	
	Cross site requesty forgery (CSRF)	[T2]	
	Componentes com vulnerabilidades conhecidas	[T2]	

o *framework* Symfony possui suporte a diferentes recursos de segurança (e.g. firewall). Diferentemente do Codeigniter, o Symfony mitigou a vulnerabilidade de *cross site scripting* porque ele possui o mecanismo Twig pré-instalado. Por padrão, o Twig realiza uma "limpeza" dos dados de entrada fornecidos pelo usuário.

O Codeigniter foi o único *framework* suscetível a falta de proteção contra ataques. Os mecanismos de proteção contra ataques, como monitoramento e controle do número de sessões abertas simultaneamente por um mesmo usuário, tem por objetivo detectar, responder e bloquear ataques, tornando o trabalho do atacante mais difícil. Por exemplo, é recomendado que aplicativos *Web* não permitam ao usuário manter múltiplas sessões ativas ao mesmo tempo. Na prática, isto significa que um usuário não deveria ter permissão para abrir múltiplas vezes o mesmo aplicativo *Web* (e.g. em navegadores ou computado-

res distintos). Na melhor das hipóteses, o usuário deveria ser alertado sobre as tentativas de abrir novas sessões no mesmo sistema. Com isso, ele teria condições de decidir se a nova sessão é sua ou resultado de um potencial ataque.

Por padrão, o Codeigniter permite que os usuários tenham múltiplas sessões simultaneamente ativas. Este foi o motivo pelo qual os *scanners* detectaram a falta de proteção contra ataques. Para resolver este problema, pode-se acoplar ao código da classe `session` do Codeigniter. Esta classe permite verificar se o usuário está com a sessão ativa, além de rastrear as atividades enquanto ele navega no site.

Os *frameworks* que obtiveram o melhor resultado em termos de mecanismos de segurança para aplicações *Web* foram o Laravel e o Symfony. Nestes dois casos, os *scanners* detectaram apenas 40% das vulnerabilidades implementadas, conforme é apresentado na Tabela 1. Isto significa que, em sua configuração padrão, esses dois *frameworks* são significativamente superiores em termos de segurança. Por exemplo, as ferramentas de varredura detectaram 70% das falhas no ambiente controlado do Phalcon e do Codeigniter, o que representa praticamente o dobro de vulnerabilidades.

O Laravel foi o único *framework* que mitigou a vulnerabilidade *cross site request forgery*. O *framework* utiliza tokens CSRF para garantir que atacantes não possam gerar solicitações falsas. A cada solicitação, o Laravel cria e integra um token, através de uma chamada de AJAX. Quando a solicitação é invocada, o *framework* compara o token de solicitação com o salvo na sessão do usuário. Caso o token não corresponder, a solicitação é classificada com inválida e nenhuma outra ação é executada.

Os resultados mostram que, apesar de um bom *framework* poder ajudar a aumentar a segurança de aplicações *Web* de forma automática e transparente, ainda há espaço para pesquisa e desenvolvimento. Novos mecanismos de segurança podem ser adicionados aos *frameworks* para impedir que a maioria das vulnerabilidades mais recorrentes em sistemas *Web*, como as dez implementadas, possa ser facilmente explorada pelos atacantes.

Por fim, respondendo a pergunta: *Implementando as mesmas dez vulnerabilidades, utilizando diferentes frameworks de desenvolvimento Web, vai alterar os resultados de detecção das ferramentas de varredura apresentados no trabalho* (e.g. [Ferrao et al. 2018b])? A resposta curta é sim. A deteção dos *scanners* de vulnerabilidades depende de diferentes aspectos. Diferentemente do trabalho [Ferrao et al. 2018b], neste trabalho, os *frameworks* representaram uma camada extra de segurança.

4. Trabalho Relacionados

Estudos demonstram que há uma significativa disparidade entre os *scanners* de vulnerabilidades existentes em termos de abrangência e níveis de exploração das vulnerabilidades [Antunes and Vieira 2015, Fonseca et al. 2014, Doupé et al. 2012, Rocha et al. 2012, Bau et al. 2010, Holík and Neradova 2017, Ferrao et al. 2018b, Melchior et al. 2018].

Os resultados apresentados na literatura indicam a necessidade de não apenas uma única ferramenta, mas sim um conjunto de *scanners* para garantir uma boa cobertura na detecção de vulnerabilidades. Além disso, de forma recorrente, os estudos apontam a necessidade de evolução e avaliação empírica constante desse tipo de ferramenta.

Trabalhos recentes realizaram estudos empíricos de *scanners* tradicionais, gratuitos, e *scanners* SaaS [Holík and Neradova 2017, Ferrao and Kreutz 2017, Melchior et al. 2018]. Num primeiro momento, identificaram as ferramentas de melhor desempenho em sistemas abertos, contendo milhares de vulnerabilidades (e.g. MV BWA/OWASP). Num segundo momento, os pesquisadores investigaram a eficácia

dos *scanners* de vulnerabilidades tradicionais e SaaS em um ambiente controlado [Ferrao et al. 2018b, Melchior et al. 2018], o que é uma prática comum e permite uma análise e comparação mais justa entre os *scanners* [Doupé et al. 2012, Bau et al. 2010, Ferrao and Kreutz 2017].

Diferentemente dos trabalhos relacionados, este trabalho avalia o impacto de *frameworks* de desenvolvimento de software na segurança de sistemas *Web*.

5. Considerações Finais

Este trabalho realizou uma análise do impacto dos *frameworks* PHP na segurança de sistemas *Web*. Os resultados indicam que há uma diferença significativa entre os *frameworks* no que diz respeito à recursos e mecanismos de segurança. Enquanto que os *frameworks* Laravel e Symfony protegeram o sistema *Web* contra 60% das vulnerabilidades propositalmente implementadas, *frameworks* como o Phalcon mitigaram apenas 30% das vulnerabilidades. Isto significa que a simples escolha do *framework* de desenvolvimento de software pode ter um impacto significativo na segurança do sistema *Web*.

Os resultados indicam que, apesar dos benefícios trazidos por *frameworks*, ainda há espaço para pesquisa. Novos mecanismos de segurança podem ser adicionados aos *frameworks* para impedir que a maioria das vulnerabilidades mais recorrentes em sistemas *Web* possa ser explorada pelos atacantes. Uma versão estendida deste trabalho, incluindo uma relação de trabalhos futuros, pode ser encontrada em [Ferrao et al. 2018a].

Referências

Alam, D., Kabir, M. A., Bhuiyan, T., and Farah, T. (2015). A case study of sql injection vulnerabilities assessment of .bd domain web applications. In *CyberSec*, pages 73–77.

Antunes, N. and Vieira, M. (2015). Assessing and comparing vulnerability detection tools for web services: Benchmarking approach and examples. *IEEE TSC*, 8(2):269–283.

Bau, J., Bursztein, E., Gupta, D., and Mitchell, J. (2010). State of the art: Automated black-box web application vulnerability testing. In *IEEE SSP*, pages 332–345. IEEE.

Doupé, A., Cavedon, L., Kruegel, C., and Vigna, G. (2012). Enemy of the state: A state-aware black-box web vulnerability scanner. In *USENIX Security Symposium*.

Ferrao, I. G., de Macedo, D. D. J., and Kreutz, D. (2018a). Investigação do impacto de frameworks de desenvolvimento de software na segurança de sistemas web (versão estendida). `https://goo.gl/8pFutx`.

Ferrao, I. G. and Kreutz, D. (2017). Segurança na web: análise black-box de scanners de vulnerabilidades. In *1a ERES*, pages 135–142. `https://goo.gl/buo8hQ`.

Ferrao, I. G., Sa, G. N., and Kreutz, D. (2018b). Detecção de vulnerabilidades em ambientes web controlados. SIEPE. `https://goo.gl/GxkPyq`.

Fonseca, J., Seixas, N., Vieira, M., and Madeira, H. (2014). Analysis of field data on web security vulnerabilities. *IEEE Transactions on DSC*, 11(2):89–100.

Holík, F. and Neradova, S. (2017). Vulnerabilities of modern web applications. In *40th MIPRO*, pages 1256–1261. IEEE.

INDIATESTBOOK (2017). 9 Best PHP Frameworks. `https://goo.gl/jQK7D7`.

Melchior, F., Fernandes, R., Ferrao, I., Sa, G. N., and Kreutz, D. (2018). Avaliação de scanners web oferecidos como serviço. SIEPE. `https://goo.gl/bknUaR`.

OWASP (2017). Top ten 2017 project. `https://goo.gl/snkFmd`.

Rocha, D., Kreutz, D., and Turchetti, R. (2012). A free and extensible tool to detect vulnerabilities in web systems. In *7th Iberian Conference on CISTI*, pages 1–6. IEEE.

Symantec (2017). Internet security threat report. `https://goo.gl/iuhLPX`.