Estratégias para Segurança em Aplicações Web: Auditoria, Infraestrutura e Filtragem de Ataques

Pedro Sampaio[1], **Fernando Pinheiro**[1], **Italo Brito**[1]

[1]Superintendencia de Tecnologia da Informação – Universidade Federal da Bahia
Av. Adhemar de Barros, s/n – Campus Ondina – Salvador – BA – Brazil

{pedro.sampaio, fernando.pinheiro, italovalcy}@ufba.br

Abstract. *Noting the steady increase in attacks on their web infrastructure, and verify that current protection measures were no longer sufficient, the UFBA security analysts team began a process of testing new solutions to improve security architecture. These solutions contemplated points such as validating of application source code, improvement on the infrastructure of the service with site isolation and resource limiting and the adoption of application firewall in order to filter malicious requests to the aplications. This article demonstrates the latest experiences with these strategies and some preliminary results.*

Resumo. *Ao constatar o crescente aumento de ataques contra sua infraestrutura web, e verificar que as medidas correntes de proteção não eram mais suficientes, a equipe de analistas da UFBA iniciou um processo de testes e implantação de novas soluções para melhorar a arquitetura de segurança. Essas soluções contemplaram aspectos como validação do código fonte das aplicações, melhoria na infraestrutura do serviço com isolamento e limitação de recursos de sites e adoção de firewall de aplicação para filtragem de acessos maliciosos às aplicações web. Este artigo demonstra as últimas experiências com estas estratégias e alguns resultados preliminares desse trabalho.*

1. Introdução

Nos últimos anos, a quantidade de ataques que tem como alvo aplicações web aumentou significativamente. Grande parte por ser um ambiente ubíquo que se insere hoje na maioria das aplicações que trabalham em rede. Também pelo fato de ser um dos serviços mais expostos a ataques remotos, por conta da sua função como interface entre usuários e recursos.

A UFBA possui diversas aplicações web disponibilizadas para seus usuários e para a comunidade em geral, hospedadas nas mais variadas plataformas, tais como: aplicações corporativas desenvolvidas, em sua maioria, em Java; aplicações de usuário, que possuem um espaço no servidor web e banco de dados, e podem fazer uso de qualquer framework PHP, ASP, HTML, etc, ou ainda utilizar plataformas de gerenciamento de conteúdo (CMS, do inglês *Content Management System*); e hospedagem em CMS's homologados pela equipe (e.g. *Drupal*, *Wordpress*, *Foswiki*), onde cria-se apenas um novo site dentro da plataforma compartilhada, com plugins e configurações gerenciadas pela equipe de TI da Universidade.

Esse cenário diversificado permite uma flexilidade muito grande na hospedagem de conteúdo web, porém pode levar à sérios riscos de segurança da informação: sistemas desatualizados e com vulnerabilidades conhecidas; vulnerabilidades em código de

aplicações desenvolvidas sem cuidado com a segurança; comprometimento em massa de aplicações devido ao uso de infraestrutura compartilhada; vulnerabilidades nos protocolos e serviços de rede; dentre outros.

O CERT.br (Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil) publicou relatório recente que aponta um aumento de 128% nas notificações de ataques a servidores Web em relação a 2014, totalizando 65.647 notificações de um total de 722.205 incidentes [CERT.br, 2016]. Os atacantes geralmente exploram vulnerabilidades em aplicações Web para, então, hospedar páginas falsas de instituições financeiras, código malicioso (e.g. vírus, cavalo de troia[1], *backdoors*[2] etc), ferramentas de ataques a terceiros, indisponibilizar o acesso ao conteúdo do site ou simplesmente para desfigurar a página da instituição e publicar mensagens de conteúdo ofensivo.

Em particular, a UFBA, e diversas outras universidades e órgãos governamentais, também têm sido alvo de ataques cibernéticos contra aplicações web nos últimos anos. Somente em 2015 a UFBA sofreu 93 ataques de desfiguração de páginas de um total de 273 incidentes de segurança. Nesses ataques, as principais falhas exploradas foram: vulnerabilidades de injeção de código SQL, inclusão local de arquivos e ataques de força-bruta contra mecanismos de autenticação.

Assim, como contramedida a esse cenário adverso, a equipe de analistas da Universidade Federal da Bahia passou a arquitetar medidas estratégicas para tratar os riscos, identificar e mitigar as vulnerabilidades e minimizar os impactos, baseado em pontos chave que, acredita-se, são vetores críticos para qualquer organização. Estas medidas visam tratar aspectos como código vulnerável das aplicações, infraestrutura susceptível a propagação de ataques, filtragem de tentativas de acesso malicioso às aplicações e aplicações com falhas de segurança conhecidas.

Esse artigo apresenta um trabalho técnico realizado na UFBA para tratar e mitigar falhas de segurança em aplicações web supracitadas. Serão apresentadas diversas estratégias e ferramentas adotadas, resultados preliminares e trabalhos futuros.

2. Técnicas para detecção e tratamento de falhas em aplicações web

2.1. Revisão de Código

Grande parte dos usuários de serviços de hospedagem de aplicações e websites não possui habilidades maduras de desenvolvimento web. Por esse motivo, é muito comum encontrar falhas de segurança críticas que poderiam ser mitigadas facilmente com a ajuda de princípios simples de desenvolvimento seguro. Mesmo após empreender mecanismos de segurança indireta como firewalls e sistemas de detecção de intrusão, este ponto de falha na infraestrutura não é facilmente contemplada.

Uma das soluções para educar os usuários é realizar workshops e treinamentos, entre outros encontros para disseminar informação. Porém a maioria dos usuários não dispõe de tempo para a curva de aprendizado exigida, e nem tem como principal ocupação o desenvolvimento de aplicações, preferindo focar na produção de conteúdo. Por esse

[1] Software usado para furtar informações e credenciais de maneira silenciosa

[2] *Backdoors* são códigos especialmente escritos para ganhar acesso privilegiado a um website, permitindo um atacante executar comandos remotamente, modificar conteúdo em massa do servidor e muito mais. Um backdoor pode ser uma pequena aplicação ou script PHP, ASP, JSP, etc.

motivo, também idealizou-se o serviço de auditoria de código, por meio de ferramentas de análise estática e dinâmica, visando disponibilizar um instrumento de fácil acesso aos usuários, afim de garantir um nível mínimo de segurança em suas aplicações.

2.1.1. Análise Estática

A busca por bugs em programas pode ser feita de diversas maneiras. A maneira mais comum e custosa, é através da leitura visual do código. Para isso utiliza-se da experiência do auditor com relação a bugs conhecidos, assim como a sua percepção e perspicácia em descobrir os bugs desconhecidos e inusitados [Chess e McGraw, 2004]. Para aplicações pequenas, esta abordagem pode ser viável e precisa porém, a medida que a quantidade de código aumenta, também aumenta a probabilidade de ocorrerem erros na análise [Teixeira, 2007].

Assim, surge a necessidade de se utilizar ferramentas para acelerar o processo, levantando dados que auxiliem a análise e direcionem o auditor para os pontos mais prováveis de falha. Desta forma diminui-se a necessidade de conhecer toda a arquitetura da aplicação e pode-se focar em pontos compartimentalizados e específicos de análise.

Essa automatização é chamada de análise estática, e se propõe a examinar um programa a procura de falhas, sem precisar executá-lo. O exame pode ser feito no código fonte do programa ou em uma versão compilada do mesmo [Chess e McGraw, 2004].

Em casos de clientes internos que possuem processos profissionais de produção, como fabricas de software, algumas ferramentas de análise estática automatizada podem ser introduzidas na cadeia de desenvolvimento. Desta forma, a auditoria passa a ser parte do processo, permitindo que o próprio programador possa visualizar as falhas ainda no inicio da produção, e os corrija a cada ciclo de compilação.

A Superintendência de Tecnologia da Informação da UFBA contém a Coordenação de Sistemas de Informação (CSI), que desenvolve aplicativos internos para a universidade. Um projeto piloto foi realizado em conjunto com a CSI, para testar a ferramenta aberta de análise estática SonarQube. Esta ferramenta foi escolhida, pelo fato de ser especializada em código JAVA, e por poder ser integrada a IDE utilizada pelos programadores.

O SonarQube funciona por meio de uma arquitetura cliente servidor, e agentes são usados para realizar a análise localmente. Os resultados são enviados no final para o servidor, que por sua vez categoriza e correlaciona as informações em relatórios normalizados.

Durante os testes, alguns templates de bugs de segurança foram importados do aplicativo FindBugs para o SonarQube, já que o mesmo não possui uma base própria com regras de análise específicos para falhas de segurança. O escâner utiliza essas regras para relacionar pontos de falha, ao analisar o código em tempo de compilação.

Outra ferramenta utilizada foi o escâner de vulnerabilidades RIPS, que atua sobre aplicações PHP. Durante diversos casos de incidentes, a ferramenta auxiliou na busca pelas causas raízes de ataques, reportando indícios importantes, e corroborando as justificativas nos relatórios de tratamento. Utilizando uma interface web, os resultados são

mostrados indicando exatamente o local no código onde há suspeita, como mostra a figura 1.

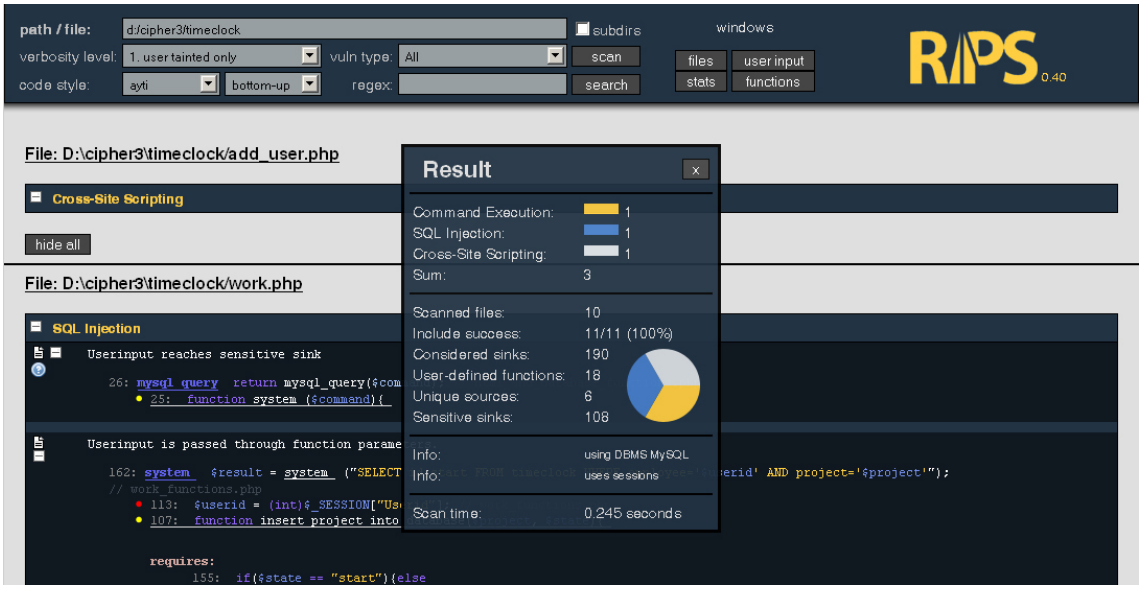

Figura 1. Resultado da ferramenta RIPS.

Uma desvantagem encontrada nas ferramentas de automação de análise estática é a quantidade elevada de falsos positivos, que podem aumentar o ruido na análise do código, e trazendo a necessidade de verificação manual das vulnerabilidades reportadas. Certa customização das configurações é eventualmente necessária para diminuir os erros e tornar a análise mais eficiente.

2.1.2. Análise Dinâmica

Ao contrário da análise estática, a análise dinâmica avalia um programa baseado nos dados da sua execução em tempo real. Para isso podem ser utilizadas diversas ferramentas que auxiliam nesse processo, automatizando tarefas repetitivas como testes de validação de entrada de dados, verificação de mensagens em protocolos, e homologação de configurações seguras.

O SQLmap é uma ferramenta de linha de comando que automatiza a busca por vulnerabilidades de injeção de comandos SQL. Ela irá procurar nas páginas do site por possíveis locais de entrada de dados, e tentará fazer a injeção, afim de colher os resultados e identificar a possibilidade de exploração. A análise não é totalmente eficiente, mas pode indicar pontos mais prováveis de risco, deixando análises mais complexas para serem feitas manualmente.

A ferramenta pode ainda extrair automaticamente os dados de bancos vulneráveis que forem eventualmente encontrados, como mostra a figura 2.

Outros aspectos como a gerência de sessões, geração de tokens, implementação de criptografia e outros tipos de mecanismos utilizados pela aplicação web podem ser analisados, utilizando ferramentas de interceptação como o ZAP, desenvolvido pelo grupo

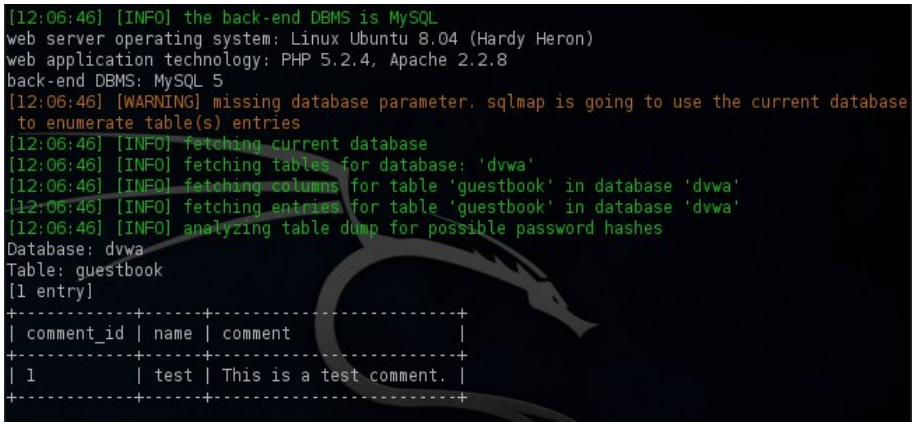

Figura 2. Exemplo de extração de banco de dados com SQLmap.

OWASP. Com esta ferramenta podemos interceptar mensagens e modificar parâmetros passados, afim de subverter funcionalidades implementadas para fins maliciosos não previstos pelo desenvolvedor.

A figura 3 mostra o uso do ZAP na sua funcionalidade crawler. Esse função permite o mapeamento do site, afim de realizar o reconhecimento da aplicação. Posteriormente, a funcionalidade de proxy pode ser usada para injetar dados nas mensagens.

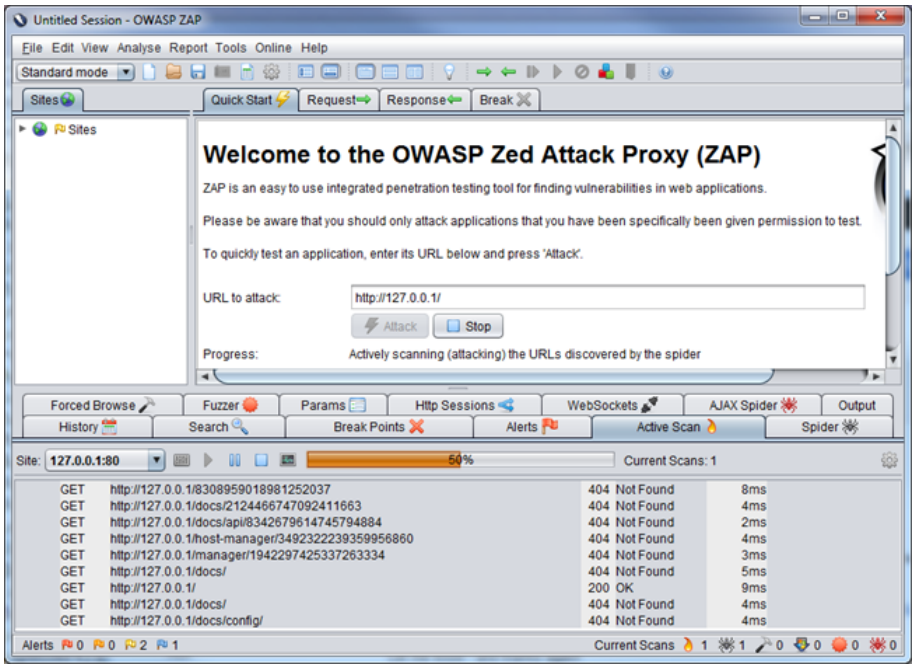

Figura 3. Reconhecimento do site com ZAP.

2.2. Infraestrutura Segura para Aplicações WEB

O software presente nos sites web não é único ponto de falha existente. Muitos ataques são facilitados também pela ausência de configurações seguras. Na maioria das vezes por conta de serviços desatualizados, e que não tem acompanhamento periódico do seu uso.

Para diminuir esses casos, a arquitetura de implementação da infraestrutura web deve ser pensada para que evitar ou conter tais problemas.

A pichação de sites é um dos ataques mais comuns hoje, e os efeitos deste tipo de incidente pode ser benigno, se uma arquitetura segura for implementada no serviço web, sobre tudo no formato de armazenamento dos dados pertencentes aos sites. Se esses locais não forem isolados entre si, o invasor de um site, pode acabar facilmente invadindo outros sites no mesmo servidor.

Outro problema recorrente é a falta visão situacional devido a escassez de informação a cerca dos incidentes. É sabido que as aplicações em geral podem gerar montanhas de registros de atividade ou as vezes quase nenhum, e em ambos os casos a dificuldade de análise e tratamento de incidentes aumenta exponencialmente. Desta forma um conjunto de configurações que vise formatar os logs para se tornarem precisos e o uso de ferramentas para facilitar a visualização e até automatizar o processo de alerta, são essenciais dentro de uma infraestrutura segura.

2.2.1. Virtual Hosts Independentes com Cloudlinux

Uma das soluções a ser adotada é o aplicativo Cloudlinux. Com ele é possível gerenciar recursos usados pro cada usuário, isolando cada site, de forma que as atividades de um usuário não afetem os outros. Assim, a cada usuário é dado uma cota de recursos (Memória RAM, CPU) máxima, da qual não poderá exceder.

Essa limitação de recursos com cota pode ser flexibilidade sobre demanda através do uso de técnicas de *burst*, que consiste em aumentar os recursos de um site em momentos de pico até limiares máximos configurados. Em particular, uma contribuição da nossa equipe para a ferramenta foi o desenvolvimento de plugins que integram o CloudLinux, mais especificamente os LVEs (do inglês, *Lightweight Virtual Environment*), à ferramenta de monitoramento Zabbix, viabilizando a checagem do uso de recursos e aumento automático para atender a momentos de sobrecarga.

Outro recurso interessante é o CageFS. Ele é um sistema de arquivos que permite criar uma jaula para cada site, semelhante a um chroot. Porém, com muito mais integração com outras partes da infraestrutura web. Além disso, quando ocorre uma invasão a um site, os outros sites hospedados no mesmos servidor físico não são afetados, por estarem também enjaulados.

Uma opção ao uso do CloudLinux para isolamento de sites virtuais em ambiente compartilhado é o Docker [Gomes e Souza, 2015], que permite criar contextos isolados de processos em execução, chamados *containers*, com fácil integração e implantação. Os benefícios do Docker vão além do isolamento de sites, tangenciando um processo de desenvolvimento ágil e integrado entre as equipes de infraestrutura e sistemas. Por outro lado, algumas funcionalidades de contenção de recursos como CPU e memória precisam ser melhor investigados para evitar que uma aplicação afete o desempenho de outras.

2.2.2. Tratamento/Visualização fácil de logs

Em qualquer incidente, a fonte de dados mais utilizada para o tratamento são os logs de atividade e erro. Porém, um ataque pode atingir mais de uma aplicação, site, ou dispositivo. A maior dificuldade na análise destas informações esta na multiplicidade de formatos nos logs. Cada dispositivo ou programa implementa seus registros de forma diferente.

Para consolidar essas informações de forma eficiente e útil, é preciso utilizar ferramentas que normalizem e correlacionem esses logs, originadas de fontes diversas. Uma destas ferramentas é o OSSIM. Ela permite sintetizar logs diferentes em um único formato e correlacionar essas informações de forma a apontar um mesmo evento correlato.

Desta forma, indício de ataques podem ser facilmente encontrados e substanciados, revelando pistas concretas, e agilizando o tratamento. As informações são ainda indexadas, e categorizadas, para posterior pesquisa e extração de estatísticas, como indica a figura 4.

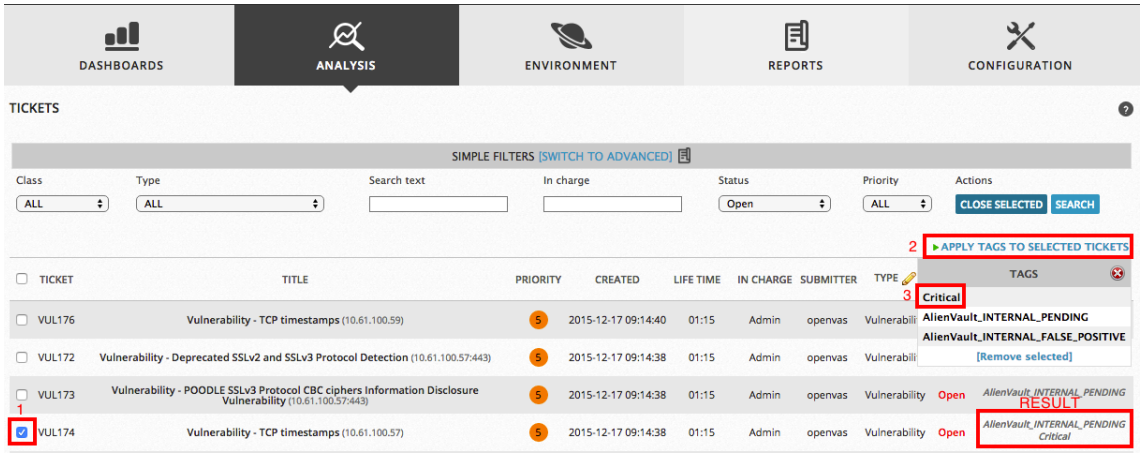

Figura 4. Eventos de ataques correlacionados pelo OSSIM.

2.3. Filtragem de Ataques

Na maioria dos casos, os ataques são realizados de forma remota, utilizando protocolos de comunicações normais, mas subvertidos para fins maliciosos. Por exemplo, em um ataque de injeção SQL a query maliciosa deve ser enviada de alguma maneira para o servidor, para que seja interpretada pela aplicação vulnerável. Geralmente o envio é feito utilizando uma variável, presente em um formulário html, que é transportado em uma mensagem HTTP POST.

Esse método constitui uma assinatura de ataque, que pode ser identificada com o auxilio de ferramentas de filtragem como sistemas de detecção e prevenção de intrusão, e firewalls. Uma das ferramentas testadas pela UFBA foi o modsecurity, como veremos a seguir.

2.3.1. Firewall de Aplicações Web com Modsecurity

O modsecurity é um módulo do servidor web Apache2 que permite filtrar ataques contra sites, baseado em regras pré estabelecidas. Uma análise da comunicação HTTP é feita,

desencapsulando os dados contidos no pacote. A regras então estabelecem se a mensagem contém um ataque ou não, utilizando expressões regulares para engatilhar um alerta.

Em testes feitos, o modsecurity pôde filtrar ataques SQL, XSS, Diretório transverso, entre outros, com relativa eficácia. Porém, um alto grau de falso positivos foram constatados, quando utilizando regras padrão fornecidas pela OWASP, e configurações não modificadas. Para diminuir os falsos positivos, foi necessário realizar uma customização para o ambiente da UFBA.

3. Resultados Preliminares

As técnicas descritas na Seção 2 foram de extrema importância para aumentar a segurança das aplicações web na UFBA. Cada uma das técnicas preencheu uma lacuna de segurança e trouxe maior robustez e disponibilidade para as aplicações. Alguns desses resultados podem ser mensurados de forma direta pela quantidades de ataques contidos, diminuição na quantidade de incidentes de segurança ou estatísticas de disponibilidade dos sites. Outros, no entanto, são medidos de forma indireta, pela satisfação dos usuários pela confiabilidade no serviço, pela detecção antecipada de vulnerabilidades e pelo reconhecimento do trabalho pela equipe. A seguir apresentaremos e discutiremos alguns resultados preliminares. Outros resultados foram omitidos devido a limitações de espaço e serão apresentados em momento oportuno.

Um dos mecanismos cujo custo-benéficio se mostrou sobremaneira vantajoso foi o uso do Mod-Security para conteção de ataques de força-bruta em aplicações web desenvolvidas pelos e CMS conhecidos, como Drupal e Wordpress. A Figura 5 apresenta estatísticas de contenção desses pelo mod-security. A configuração dessas regras de contenção leva em consideração o comportamento das requisições HTTP de uma aplicação web quando do acesso à funcionalidades administrativas. Por exemplo, no caso do Wordpress, tentaticas de acesso à página `wp-login.php` com código de retorno HTTP 200 (indicando falha de login) e que somem um número alto de requisições em poucos minutos, são bloquedas por um tempo razoável. Ao ser bloqueado, o usuário recebe como retorno uma página customizada com informações do motivo do bloqueio.

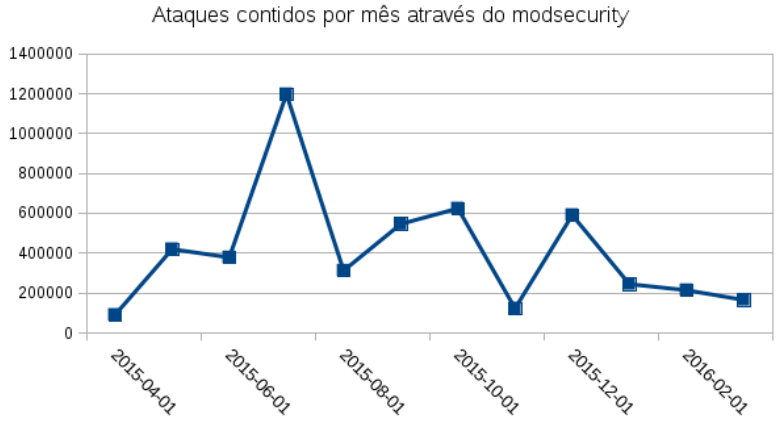

Figura 5. Estatísticas de contenção de ataques web via Mod-Security

Além do controle de ataques de força-bruta, a equipe realizou um longo trabalho de customização das regras da OWASP para bloquear ataques de injeção de código SQL

por reputação de requisições. Nesse modo de bloqueio, um índice de anomalia é adicionado à cada requisição e incrementado a medida que padrões de ataques conhecidos são encontrados naquela requisição. Diferente do modo padrão, cujo bloqueio ocorre assim que uma regra tem casamento com o padrão de trafego, as customizações efetuadas modificam os índices de cada regra para maior adequação ao cenário da Universidade, uma vez que o bloqueio de requisições legítimas tem prejuízo proporcional à liberação de requisições maliciosas por afetar a disponibilidade do website.

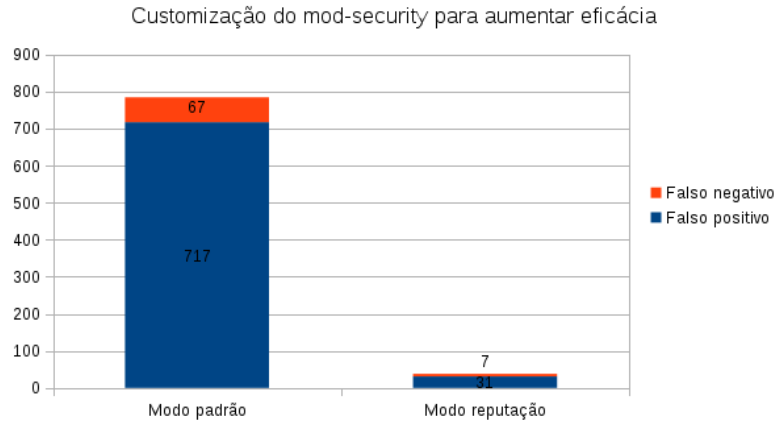

Figura 6. Resultados de testes de validação para customizações no mod-security

A Figura 6 apresenta os dados de bloqueio do modsecurity em sua configuração padrão e após as customizações implementadas. Nesse teste foram utilizadas 2187 amostras de requisições HTTP GET a mais de 300 sites hospedados em um ambiente de servidor compartilhado. Desse total de requisições, 650 requisições eram ataques de injeção de código SQL e as demais eram requisições válidas dos usuários, nos mais variados formatos. É possível notar que após as customizações implementas, foi possível elevar de forma significativa o índice de detecção de ataques, com taxa de falso negativo (quando o ataque não é identificado) de aproximadamente 2%, 7 ocorrências, ao passo que o número de falsos positivos (requisições incorretamente bloqueadas) manteve-se em um valor aceitável de 5%, com 31 ocorrências. Essas ocorrências de falso positivo podem ainda ser menores com a configuração de regras de exceção nas aplicações específicas. Futuramente pretendemos permitir que o próprio usuário que está navegando por um site possa nos reportar o falso positivo, através de uma página de auto-recuperação e com menor interação da equipe de TI possível.

Outro resultado importante alcançado foi na análise estática de código, onde, de maneira pró-ativa ou reativa, é possível fazer uma análise detalha do código-fonte de aplicações em busca de padrões de desenvolvimento com falhas conhecidas. A Figura 7 apresenta a análise de uma aplicação real de um grupo de pesquisa da Universidade que estava hospedada em nosso ambiente. Como pode ser visto a aplicação continha falhas graves como execução de código (1), inclusão de arquivos (70), injeção de código SQL (141), cross-site scripting (724), dentre outras. A apresentação desses dados possibilitou mobilizar os responsáveis pelo site para sua migração imediata para outra plataforma, o que diminuiu a incidência de ataques naquele servidor de forma notável.

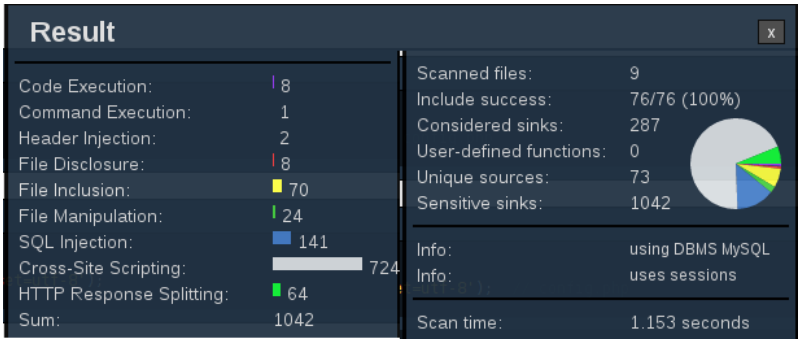

Figura 7. Saída recortada do RIPS para análise de website PHP

4. Considerações Finais e Trabalhos Futuros

Muitas outras estratégias podem ainda serem usadas para proteger o ambientes web. A técnicas apresentadas neste artigo, fizeram parte de um processo de estudo e implantação que visou verificar além de tudo, a eficácia destas estratégias, e estabelecer um alicerce para o futuro da infraestrutura de segurança na universidade.

Constatou-se que, apesar de haver espaço para maior investigação, customizações e trabalhos correlatos, as medidas apresentadas podem trabalhar adequadamente em conjunto, trazendo benefícios rápidos e de fácil administração por parte das equipes de segurança de redes (ETIR), utilizando ferramentas abertas e largamente suportadas por suas respectivas comunidades.

Como trabalhos futuros, pretende-se: i) implementar soluções de auto-recuperação para bloqueios realizados pelo mod-security a partir do feedback dos usuários; ii) desenvolver e implantar ferramentas de checagem contínua de vulnerabilidades conhecidas; iii) consolidar um processo de teste de intrusão de aplicações web; iv) analisar o uso de Docker como mecanismo para isolamento de sites em complemento ao CloudLinux; e v) promover treinamentos para a equipe de sistemas sobre padrões seguros de desenvolvimento de aplicações web.

Referências

Barnett, R. C. (2013). *Web Application Defender's Cookbook*. John Wiley & Sons, Inc, 10475 Crosspoint Boulevard, Indianapolis, IN 46256.

CERT.br (2016). Incidentes reportados ao cert.br – janeiro a dezembro de 2015. Publicado em http://www.cert.br/stats/incidentes/2015-jan-dec/analise.html.

Chess, B. e McGraw, G. (2004). Static analysis for security. *IEEE Security & Privacy*, (2):76–78.

Gomes, R. e Souza, R. (2015). Docker - infraestrutura como código, com autonomia e replicabilidade. IX Workshop de TI das IFES.

Meucci, M. e Muller, A. (2014). Owasp testing guide v4. Publicado em https://www.owasp.org/images/5/52/OWASP_Testing_Guide_v4.pdf.

Morana, M. (2013). *Application Security Guide For CISOs v1.0*. OWASP Foundation.

OWASP (2013). *OWASP Top 10 2013*. OWASP Foundation.

Ristic, I. (2002). *Mod Security Handbook*. Feisty Duck, 1 edition.

Silva, R. F. e Cunha, J. A. (2005). Arquitetura de segurança em aplicações baseadas em web services.

Teixeira, E. P. L. (2007). Ferramenta de análise de código para detecção de vulnerabilidades. Master's thesis, Universidade de Lisboa, Portugal.