# Proposta de Projeto de Doutoramento a Desenvolver no Âmbito do 1º Concurso para Atribuição de Bolsas de Investigação na Área de Engenharia Informática

## 1. Título do projeto

**Título:** Deteção e sugestão automática de patches de segurança

**Palavras-chave:** Análise de Segurança, Execução Simbólica, Monitorização de Sistemas, Auto Patching

**Referência**: CEE_EI_IST1

## 2. Instituições envolvidas

**Instituição onde o doutoramento será realizado:** IST, U.Lisboa

**Outras instituições participantes no projeto de investigação:**

Colaborações científicas com instituições dos PALOP / Timor Leste serão consideradas no decorrer do projeto.

## 3. Equipa de Orientação

**Orientador:** Rui Maranhão, Professor Associado com Agregacão

**Coorientador:** Pedro Adão, Professor Auxiliar

## 4. Descrição do Projeto

A presença de bugs no software é um dos principais problemas associados ao desenvolvimento de software e estima-se que custa cerca de US$1,1 biliões anuais à economia mundial. Para reduzir a probabilidade de acontecerem erros, aumentando dessa forma a qualidade do software, e reduzindo o tempo para comercialização, os projetos de software devem incorporar uma fase de testes minuciosa durante a fase de desenvolvimento. Na verdade, o desenvolvimento de software representa um custo massivo para a economia global, dos quais mais de 50% do custo vai para validação e verificação. Mesmo na presença de testes exaustivos, infelizmente os defeitos residuais são uma grande ameaça à dependabilidade na fase operacional.

O objetivo deste projeto é desenvolver um sistema que mitigue automaticamente situações potencialmente faltosas em sistemas de software e sem comprometer os requisitos funcionais e não funcionais. Neste projeto vamo-nos focar na mitigação de erros que levam à exploração de vulnerabilidades de segurança.

A estratégia de curto prazo é parar a execução do programa assim que uma vulnerabilidade seja detectada. Através de testes de software, podemos detectar erros, por exemplo, monitorando a execução à procura de comportamentos potencialmente incorretos. Uma vez detectado um problema, a estratégia de mitigação por omissão proposta pelas técnicas mais recentes é terminar o programa e fornecer um conjunto de sugestões de reparo que os programadores podem inspecionar e aplicar manualmente. A longo prazo, como em muitos cenários importantes a disponibilidade do sistema é um requisito importante pelo que planeamos investigar estratégias de auto correção (*AutoPatching*), servindo o pedido mesmo perante erros de segurança.

Para realizar esta visão, prevemos um sistema com três componentes principais:

**1.** Monitorização e aprendizagem: uma ideia chave do AutoPatching é aprender o que é legítimo a partir de um conjunto de execuções (normais/defeituosas) e inferir um modelo que caracteriza a execução correta. O modelo é uma coleção de propriedades ou prováveis invariantes sobre os valores observados. Cada invariante foi sempre satisfeita durante as execuções normais. À medida que existem mais observações, o modelo vai tornando-se mais preciso. A implementação deste módulo irá alavancar as ferramentas de monitoramento disponíveis em ambientes de teste e produção, gerando casos de teste usando técnicas de fuzzing, genéticas e metamórficas para criar um conjunto mais diversificado de execuções [1]. Também será aproveitada a existência de conjuntos de dados que contenham provas de vulnerabilidades (PoV) que exercitem os traços defeituosos [2], bem como grandes conjuntos de dados (como o tráfego trocado durante as competições de Capture the Flag (CTF)), onde há uma maneira simples de identificar os traços que representam execuções defeituosas. As perguntas interessantes da pesquisa dentro deste módulo são (i) o trade-off entre a sobrecarga e os pontos monitorados assim como a quantidade de treinamento necessário e (II) qual é o conjunto de monitores que reduz o número de falsos negativos e de falsos positivos [3].

**2.** Geração de patches: geramos um conjunto de candidatos com correções de reparo que impõem uma invariante. A hipótese é que alguns erros violam invariantes, que impor invariantes violadas pode corrigir os efeitos desses erros, e que corrigir esses efeitos pode alterar a execução do serviço para eliminar a falha correspondente. O objetivo é encontrar um patch que corrija a execução após a ocorrência do primeiro erro sem conduzir a uma degradação séria nos requisitos [4]. Se nenhum candidato corretivo for possível, será gerado um candidato que fará o programa terminar ao chegar a esse estado. Em ambos os casos, o patch será incluído num pull request como uma sugestão para o programador para corrigir o problema (enquanto não é corrigido, o programa irá falhar em produção para esses cenários faltosos).

**3.** Aplicação e avaliação do patch: um candidato pode nem ter nenhum efeito, ou mesmo um efeito negativo, na aplicação. Os patches são inicialmente avaliados nos vários ambientes de teste (como fuzzing) mencionados acima antes de serem submetidos para revisão. Planeamos ainda avaliar patches observando continuamente programas corrigidos à medida que são executados [4, 5]. Isso servirá como um feedback loop a fim de gerar patches que minimizam a probabilidade de efeitos negativos, aplicando patches mais promissores primeiro. Outra fonte para o feedback loop é aprender com os patches feitos por programadores em cenários reais. Estas duas fontes de informação ajudarão o sistema a fazer suposições informadas sobre as sugestões de reparos.

## 5. Referências Bibliográficas


[1] Alexandre Perez, Rui Abreu, and Arie van Deursen. "A test-suite diagnosability metric for spectrum-based fault localization approaches." In Proc. of ICSE'17.

[2] DARPA: Sample challenges. https://github.com/CyberGrandChallenge/samples/tree/master/examples

[3] Alessio Viticchié, Cataldo Basile, and Antonio Lioy. "Remotely assessing integrity of software applications by monitoring invariants: Present limitations and future directions." International Conference on Risks and Security of Internet and Systems. 2017.

[4] Claire Le Goues, et al. "Genprog: A generic method for automatic software repair." IEEE transactions on software engineering 38.1 (2011): 54-72.

[5] Thomas Durieux, Youssef Hamadi, and Martin Monperrus. "Production-driven patch generation." In Proc. ICSE-NIER'17.