Universidade do Algarve

Identificação, Análise e Exploração de Vulnerabilidades

Pós-Graduação em Cibersegurança, 1º Semestre, 2023/24

**Hard-coded Credentials in GitHub - CVE-2023-2138**

Luís Ramos, 71179

Roberto Santos, 71214

Jorge Silva, 72480

Docente: Joel Guerreiro

Índice

[Description 2](#_Toc153030613)

[What affects 3](#_Toc153030614)

[Exploitation 4](#_Toc153030615)

[Recommended Mitigations 5](#_Toc153030616)

[Conclusion 6](#_Toc153030617)

**​**

**​**

**​**

**​**

**​**

​​

**Description**

**Introdução à Vulnerabilidade:**

* A prática de incorporar credenciais, como nomes de usuário e senhas, diretamente no código-fonte de uma aplicação é uma vulnerabilidade significativa na segurança da informação. Este método, conhecido como "credenciais codificadas" (Hard-coded Credentials), representa uma fragilidade grave, especialmente em ambientes onde o código é compartilhado ou público, como é o caso do GitHub, uma das maiores plataformas de hospedagem de código-fonte no mundo.

**Por Que é uma Prática Arriscada:**

* Em um cenário ideal de segurança cibernética, as credenciais devem ser protegidas e mantidas confidenciais. Contudo, quando essas informações são codificadas diretamente no código-fonte, tornam-se vulneráveis a uma série de riscos, incluindo exposição inadvertida, especialmente em plataformas de colaboração como o GitHub, onde o código pode ser facilmente acessado por terceiros. A exposição de credenciais pode levar a violações de segurança, como acessos não autorizados e ataques cibernéticos, comprometendo não apenas a integridade da aplicação, mas também a segurança de dados sensíveis dos utilizadores.

**Métodos Avançados de Detecção:**

1. **Auditorias de Segurança Código-Fonte:**
   * A realização de auditorias regulares e sistemáticas do código-fonte por parte de especialistas em segurança cibernética é essencial. Estes profissionais estão treinados para identificar padrões de codificação inseguros, incluindo a presença de credenciais codificadas.
2. **Utilização de Ferramentas SAST Sofisticadas:**
   * Ferramentas de Análise Estática de Código (Static Application Security Testing - SAST) são fundamentais na identificação automática de vulnerabilidades. Estas ferramentas utilizam algoritmos complexos para examinar o código-fonte e identificar padrões potencialmente perigosos, incluindo credenciais codificadas.
3. **Integração de Varreduras de Repositório no Ciclo de Desenvolvimento:**
   * A integração de ferramentas que varrem repositórios em busca de credenciais expostas no pipeline de integração contínua/desdobramento contínuo (CI/CD) é crucial. Isto permite que as vulnerabilidades sejam identificadas e corrigidas de forma proativa, antes que o código seja implementado em um ambiente de produção.

**What affects**

**Impactos Diretos da Exploração:**

* A exploração de credenciais codificadas diretamente no código-fonte pode ter múltiplos efeitos devastadores. Uma vez que essas credenciais são descobertas, elas podem ser utilizadas para ganhar acesso não autorizado a sistemas e bases de dados. Isso pode levar à alteração, exclusão ou roubo de dados sensíveis, comprometendo a integridade e a confidencialidade das informações. Além disso, um invasor pode utilizar esse acesso para realizar ações maliciosas dentro do sistema, como a implantação de malwares ou a execução de scripts prejudiciais.

**Efeitos a Longo Prazo:**

* A exposição de credenciais pode ter efeitos duradouros na reputação e credibilidade de uma organização. A violação de dados resultante pode levar a perdas financeiras significativas, danos à reputação da marca e perda de confiança dos clientes e parceiros. Além disso, pode haver implicações legais e regulatórias, especialmente se a violação resultar na exposição de dados pessoais protegidos por legislações como o GDPR.

**Objetivos e Ações do Hacker:**

1. **Acesso e Controle:**
   * O principal objetivo de um hacker ao explorar essa vulnerabilidade é obter acesso não autorizado a sistemas e aplicações. Ao usar credenciais codificadas, eles podem se autenticar como utilizadores legítimos, ganhando assim controle sobre o sistema e seus dados.
2. **Espionagem e Roubo de Dados:**
   * Hackers podem usar o acesso adquirido para espionar atividades do sistema, acessar informações confidenciais e roubar dados sensíveis. Isto pode incluir informações pessoais dos clientes, segredos comerciais, dados financeiros, entre outros.
3. **Ataques Secundários:**
   * Uma vez dentro do sistema, o hacker pode utilizar a posição privilegiada para lançar ataques secundários, tanto internamente (como a propagação de ransomware) quanto contra outras entidades (como ataques de negação de serviço a partir do sistema comprometido).
4. **Criação de Backdoors:**
   * Em alguns casos, os hackers podem instalar backdoors para garantir acesso contínuo ao sistema, mesmo depois que a vulnerabilidade original for corrigida.

**Exploitation**

**Metodologia Implementada pelo Hacker:**

1. **Análise de Repositórios Públicos:**
   * Hackers frequentemente varrem repositórios públicos no GitHub em busca de projetos que possam conter credenciais codificadas. Eles utilizam ferramentas automatizadas que podem identificar padrões de credenciais em grandes volumes de código rapidamente

**Contexto:**

* A "Análise de Repositórios Públicos" é uma metodologia empregada na segurança cibernética para identificar vulnerabilidades em códigos disponíveis publicamente, como aqueles hospedados no GitHub.
* O foco é encontrar informações sensíveis ou padrões de codificação inseguros que possam comprometer a segurança da aplicação.

**Processo de Exploração Utilizado:**

1. **Identificação do Alvo:**
   * Esta etapa envolve a escolha do repositório ou projeto específico a ser analisado. O objetivo é selecionar um alvo onde a probabilidade de encontrar credenciais codificadas seja alta.
2. **Análise da Tecnologia Utilizada:**
   * Antes de iniciar a varredura, identificamos a tecnologia utilizada no projeto. No nosso caso, focamos em um projeto desenvolvido com Django, um framework web Python, que é propenso a este tipo de vulnerabilidade devido à sua popularidade e uso generalizado.
3. **Preparação do Script de Varredura:**
   * Desenvolvemos um script em Python (passSearch.py) especialmente projetado para varrer os arquivos do projeto Django. O script busca por padrões que indicam a presença de credenciais, como username, password, user e pass.

Uma imagem com texto, captura de ecrã, software, Tipo de letra

Descrição gerada automaticamente

1. **Execução do Script de Varredura:**
   * O script é executado contra o código-fonte do projeto alvo. Ele percorre todos os arquivos relevantes (neste caso, arquivos .py) e regista qualquer correspondência com os padrões definidos num arquivo de log.
2. **Análise Manual do Log:**
   * Após a varredura, realizamos uma análise manual do arquivo de log. Este passo é crucial para identificar quais resultados são verdadeiras vulnerabilidades e quais são falsos positivos.

Uma imagem com texto, eletrónica, captura de ecrã, software

Descrição gerada automaticamente

1. **Teste de Logins Encontrados:**
   * Para validar as vulnerabilidades, tentamos utilizar os logins identificados. Este teste é feito em um ambiente controlado e ético para confirmar se as credenciais encontradas são válidas e representam uma falha de segurança real.

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

**Recommended Mitigations**

**Situação Inicial:**

* No início, o projeto Django tinha um problema significativo de segurança: as credenciais de administrador (nome de utilizador e palavra-passe) estavam codificadas diretamente no código-fonte, no arquivo views.py. Este é um exemplo clássico de uma vulnerabilidade de segurança, pois expõe informações sensíveis e facilita ataques maliciosos.

**Problema Identificado:**

* Ter credenciais estáticas no código é uma prática insegura, especialmente quando o código está em repositórios públicos como o GitHub. Isso permite que qualquer pessoa com acesso ao repositório veja essas credenciais, colocando o sistema em risco.

**Metodologia Aplicada:**

1. **Remoção das Credenciais Estáticas:**
   * A primeira etapa foi remover as credenciais estáticas do arquivo views.py.

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

1. **Criação de um Arquivo de Configuração Seguro:**
   * Em seguida, criamos um arquivo de configuração config.json para armazenar as credenciais de forma segura. Esse arquivo não é codificado diretamente no repositório, protegendo as informações.

Uma imagem com texto, captura de ecrã, Tipo de letra, Software de multimédia

Descrição gerada automaticamente

1. **Geração e Armazenamento da Chave de Criptografia:**
   * Uma chave de criptografia foi gerada pelo script generate\_key.py e armazenada de forma segura. Esta chave é essencial para descriptografar e criptografar as informações do arquivo de configuração config.json que irá se transformar em config\_encrypted.json quando encriptado.

Uma imagem com texto, captura de ecrã, software, ecrã

Descrição gerada automaticamente



Uma imagem com texto, Tipo de letra, captura de ecrã, Gráficos

Descrição gerada automaticamente

1. **Criptografia do Arquivo de Configuração:**
   * Usamos um script Python (encrypt.py) para criptografar o arquivo de configuração, garantindo que as credenciais não sejam expostas mesmo se alguém tiver acesso ao arquivo criptografado.

Uma imagem com texto, captura de ecrã, software, Tipo de letra

Descrição gerada automaticamente

1. **Descriptografia das Credenciais na Aplicação:**
   * No projeto Django, um script adicional (decrypt.py) foi usado para descriptografar as credenciais no momento da execução do aplicativo, utilizando a chave de criptografia armazenada.

Uma imagem com texto, captura de ecrã, software, Tipo de letra

Descrição gerada automaticamente

1. **Uso Seguro das Credenciais:**
   * Com esse método, garantimos que as credenciais sejam carregadas de forma segura na aplicação Django sem expor as informações no código-fonte ou no repositório.

**Nome da Metodologia:**

* A metodologia usada pode ser descrita como "Gerenciamento Seguro de Configurações e Credenciais". Envolve a remoção de informações sensíveis do código-fonte e sua proteção por meio de criptografia e descriptografia seguras.

**Conclusion**

O processo de identificar e abordar a vulnerabilidade de credenciais codificadas é vital para a segurança de qualquer software. Iniciamos com a análise de repositórios públicos, usando um script Python para varrer o código em busca de padrões que indicassem a presença de credenciais como 'username' e 'password'. Essa varredura é essencial para detetar possíveis falhas de segurança que poderiam ser exploradas maliciosamente.

A análise manual dos logs gerados pelo script foi um passo crucial, ajudando a diferenciar entre falsos positivos e vulnerabilidades reais. Este processo não só revelou práticas inseguras de codificação, mas também enfatizou a importância de métodos seguros de armazenamento de credenciais, como o uso de arquivos de configuração criptografados.

Este método de varredura e análise reforça a necessidade de práticas diligentes de segurança cibernética, destacando como a revisão cuidadosa e a gestão segura de credenciais são fundamentais para proteger os sistemas de software contra acessos não autorizados e violações de dados.