FridaForge: Geração Automatizada de Scripts Frida com Contexto Dinâmico da Aplicação

Modalidade: Código Aberto

Francisco Sales[1], **Eduardo Souto**[1], **Horácio Fernandes**[1]

[1]Instituto de Computação – Universidade Federal do Amazonas (UFAM)
69080-900 – Manaus – AM – Brasil

{francisco.sales,esouto,horacio}@icomp.ufam.edu.br

Abstract. *Dynamic instrumentation with Frida allows Android security analysts to inspect internal behavior, intercept sensitive data, and bypass protection mechanisms, but it requires expertise in both the Frida API and the target application's internal structure. This paper presents FridaForge, an open-source tool that generates and executes Frida scripts from natural language queries, using information collected from the running application to adapt scripts to the target. Evaluated on ten intentionally vulnerable Android applications (83 tasks, 92 queries, 460 executions), FridaForge achieved a 95.7% success rate. Source code, documentation, and demonstration materials are publicly available.*

Resumo. *A instrumentação dinâmica com Frida permite inspecionar operações internas, interceptar dados sensíveis e contornar mecanismos de proteção em aplicações Android, mas exige conhecimento da API do Frida e da estrutura interna da aplicação-alvo. Este artigo apresenta o FridaForge, ferramenta de código aberto que gera e executa scripts Frida a partir de consultas em linguagem natural, utilizando informações coletadas da aplicação em execução para adaptar os scripts ao alvo. Avaliado em dez aplicações Android intencionalmente vulneráveis (83 tarefas, 92 consultas, 460 execuções), o FridaForge alcançou taxa de sucesso de 95,7%. O código-fonte, a documentação e os materiais de demonstração estão disponíveis publicamente.*

1. Introdução

A instrumentação dinâmica é amplamente utilizada em atividades de análise de segurança de aplicações Android, pois permite observar e modificar o comportamento da aplicação em tempo de execução para interceptar dados sensíveis e contornar mecanismos de proteção [Sutter et al. 2024]. Nesse contexto, o Frida [Frida 2026] tornou-se uma das principais ferramentas de instrumentação, permitindo a injeção de scripts JavaScript em processos em execução sem exigir a recompilação do APK [D'Elia et al. 2019].

Apesar de sua flexibilidade, o uso efetivo do Frida exige tanto o domínio da API do framework quanto o conhecimento da estrutura interna da aplicação-alvo. Grande parte do esforço de análise ocorre antes da escrita do script, na identificação dos pontos a instrumentar, o que aumenta o tempo de análise e cria uma barreira de entrada para usuários menos experientes.

As soluções existentes reduzem parcialmente esse esforço. Ferramentas como Objection [Objection 2026] e MEDUSA [MEDUSA 2026] oferecem comandos predefinidos para tarefas recorrentes, como *bypass* de SSL pinning e detecção de root. Esses recursos são úteis quando a tarefa está contemplada no repertório da ferramenta, mas apresentam menor flexibilidade diante de mecanismos específicos da aplicação. Outra alternativa consiste em buscar scripts compartilhados em repositórios como o Frida CodeShare [Frida CodeShare 2026]. Nesse caso, o analista precisa localizar, compreender e adaptar manualmente o script ao alvo analisado, mantendo parte da dependência de conhecimento técnico e da estrutura da aplicação.

Modelos de linguagem de grande escala oferecem uma terceira possibilidade: gerar scripts sob demanda a partir de descrições em linguagem natural [Chen et al. 2021, Rozière et al. 2023], inclusive em tarefas de segurança [Xu et al. 2025]. Entretanto, sem informações sobre a aplicação-alvo, a geração direta tende a produzir scripts baseados em classes, métodos ou bibliotecas plausíveis, mas inexistentes no software analisado.

Este artigo apresenta o *FridaForge*, uma ferramenta de código aberto desenvolvida para apoiar testes de penetração e outras atividades de análise dinâmica de segurança em aplicações Android. A ferramenta gera e executa scripts Frida a partir de consultas em linguagem natural, nas quais o analista descreve o objetivo da análise, como interceptar credenciais, observar operações criptográficas ou contornar mecanismos de proteção. Quando necessário, o FridaForge coleta automaticamente informações da aplicação em execução, incluindo classes carregadas, métodos, bibliotecas de terceiros, módulos nativos e mecanismos de armazenamento local. Essas informações são incorporadas ao prompt enviado ao modelo de linguagem, favorecendo a geração de scripts adaptados ao alvo instrumentado.

As principais contribuições deste trabalho são: i) o *FridaForge*, ferramenta descrita acima; ii) um mecanismo de coleta e incorporação de contexto obtido em tempo de execução, utilizado para orientar a geração de scripts específicos para a aplicação-alvo; e iii) uma avaliação experimental em dez aplicações Android intencionalmente vulneráveis, abrangendo 83 tarefas de segurança, 92 consultas e 460 execuções, na qual a ferramenta alcançou taxa geral de sucesso de 95,7%.

O restante deste artigo está organizado da seguinte forma: a Seção 2 apresenta a arquitetura e as funcionalidades do FridaForge; a Seção 3 descreve a avaliação experimental, a comparação com ferramentas existentes, as limitações e a demonstração da ferramenta; e a Seção 4 apresenta as conclusões e os trabalhos futuros.

2. FridaForge: Arquitetura e Funcionalidades

O FridaForge é uma ferramenta de linha de comando e código aberto, implementada em Kotlin, que apoia testes de penetração e outras atividades de análise dinâmica de segurança em aplicações Android. A ferramenta automatiza a geração e a execução de scripts Frida a partir de consultas em linguagem natural, reduzindo o esforço necessário para identificar pontos de instrumentação e desenvolver scripts específicos para cada aplicação.

2.1. Visão Geral

A interação com o FridaForge inicia quando o analista informa a aplicação Android a ser instrumentada e descreve, em linguagem natural, o objetivo da análise. Exemplos de consultas incluem interceptar credenciais, observar operações criptográficas, inspecionar dados persistidos e contornar mecanismos de proteção, como SSL pinning e detecção de root.

O fluxo de operação é composto por quatro etapas principais: i) interpretação da consulta submetida pelo analista; ii) coleta de informações da aplicação em execução; iii) construção do prompt e geração do script por um modelo de linguagem; e iv) validação, injeção e execução do script na aplicação-alvo. A Figura 1 apresenta esse fluxo.

A ferramenta conecta-se a um dispositivo ou emulador Android por meio do ADB e estabelece uma sessão de instrumentação com o Frida. A partir dessa sessão, o FridaForge coleta informações sobre a aplicação, seleciona o contexto adequado à consulta e constrói a entrada enviada ao modelo de linguagem. O script produzido é processado, injetado no processo da aplicação e executado, enquanto sua saída é apresentada ao analista no ambiente de linha de comando.

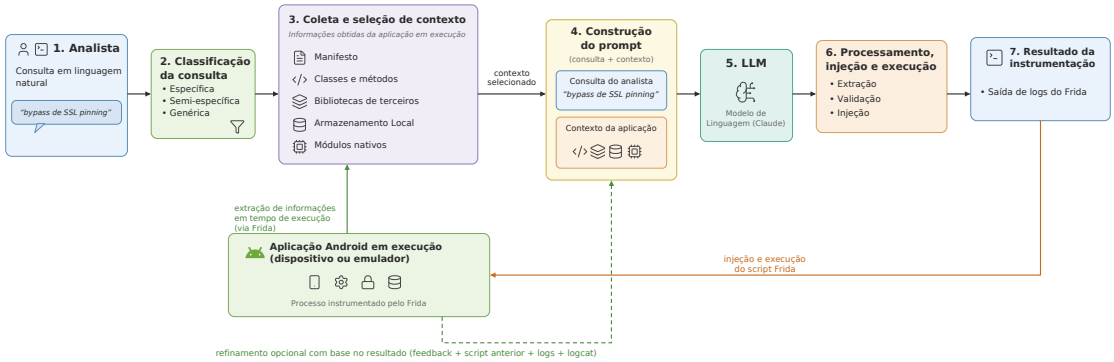

Figura 1. Fluxo de operação do FridaForge.

Esse processo reduz a dependência de conhecimento prévio detalhado sobre a API do Frida e sobre a estrutura interna da aplicação-alvo. Em vez de localizar manualmente classes, métodos e bibliotecas relevantes, o analista descreve a tarefa desejada, enquanto o FridaForge utiliza informações observadas em tempo de execução para orientar a geração do script.

A classificação do nível de especificidade ocorre automaticamente, com base em padrões textuais identificados na consulta submetida, sem que o modelo de linguagem participe dessa etapa. Consultas com referência totalmente qualificada, como `pacote.Classe.metodo()`, são classificadas como específicas. Quando cita nomes de classes ou métodos de forma incompleta, sem indicar o pacote, é considerada semi-específica. Consultas que não seguem nenhum desses padrões recaem na categoria genérica, descrevendo apenas a intenção de análise em alto nível.

Essa classificação determina a quantidade de contexto fornecida ao modelo. Para consultas específicas, o alvo da instrumentação já está delimitado pelo próprio analista, dispensando a etapa de coleta de contexto. Consultas semi-específicas recebem

informações complementares sobre classes e métodos relacionados. Consultas genéricas recebem um contexto mais amplo, incluindo classes, bibliotecas, mecanismos de armazenamento e módulos nativos, de modo a reduzir a ambiguidade na escolha dos pontos de instrumentação.

2.2. Coleta Dinâmica de Contexto

O principal diferencial técnico do FridaForge é a coleta automática de informações da aplicação em tempo de execução antes da geração do script. Após estabelecer conexão com a aplicação, a ferramenta extrai informações do arquivo de manifesto, como permissões declaradas e indicadores de configuração relevantes para segurança, como *debuggable* e *allowBackup*, e enumera classes carregadas pertencentes ao escopo do aplicativo, seus métodos e assinaturas, bibliotecas de terceiros identificadas e classificadas por categoria, como rede, serialização e segurança, módulos nativos carregados e mecanismos locais de armazenamento, como bancos de dados SQLite, arquivos internos e chaves persistidas em *SharedPreferences*.

A Figura 2 apresenta um exemplo de contexto coletado da aplicação InsecureBankv2. Essas informações podem orientar, por exemplo, a geração de scripts destinados à interceptação do fluxo de autenticação ou ao acesso a dados armazenados de forma insegura.

```
Package: com.android.insecurebankv2
Classes: 27 (CryptoClass, DoTransfer, TrackUserContentProvider, ...)
Permissions: INTERNET, SEND_SMS, READ_CONTACTS, ... (19)
Database: mydb (tables: names)
SharedPreferences: superSecurePassword, EncryptedUsername)
Native modules: 1 app / 366 total (no protections)
```

Figura 2. Exemplo de contexto dinâmico coletado pelo FridaForge para a aplicação InsecureBankv2.

2.3. Construção do Prompt e Geração de Scripts

Após a coleta, o contexto é selecionado de acordo com o nível de especificidade da consulta e incorporado ao prompt enviado ao modelo de linguagem. O prompt é composto por três elementos principais: i) instruções sobre a tarefa de instrumentação; ii) descrição estruturada do contexto coletado; e iii) consulta formulada pelo analista.

A seleção progressiva do contexto busca equilibrar dois riscos: informações insuficientes para localizar os pontos corretos de instrumentação, e um volume excessivo de informações irrelevantes, que pode aumentar a ambiguidade da geração.

O prompt de consultas específicas se limita às classes e métodos que o próprio analista já indicou. Nas consultas semi-específicas, o FridaForge acrescenta as classes e métodos pertencentes ao pacote referido na consulta, somados aos metadados das bibliotecas de terceiros detectadas, sem descartar nenhum item desse grupo por não parecer relevante. Nas consultas genéricas, o prompt passa a reunir tudo o que foi coletado da aplicação até aquele momento, sem exceção: classes, bibliotecas, armazenamento local, permissões e módulos nativos.

Por exemplo, ao receber a consulta “bypass de SSL pinning” em uma aplicação que utiliza a biblioteca OkHttp, o FridaForge pode incluir no prompt a presença da classe

CertificatePinner e de seus métodos de validação. Isso reduz a probabilidade de o modelo produzir um script baseado em bibliotecas ou pontos de interceptação inexistentes no alvo.

O resultado esperado é um script JavaScript compatível com a API do Frida e adaptado aos elementos observados na aplicação. O FridaForge solicita que a resposta contenha apenas o código necessário para a instrumentação, reduzindo a presença de explicações ou blocos textuais incompatíveis com a execução.

2.4. Validação, Execução e Refinamento

A resposta produzida pelo modelo passa por uma etapa de processamento antes da injeção na aplicação. O FridaForge extrai o código JavaScript da resposta, removendo elementos textuais que não pertencem ao script, como delimitadores Markdown e explicações adicionais, e submete o resultado extraído a uma checagem sintática antes de liberar a execução, confirmando o fechamento correto de todos os delimitadores e a presença de chamadas típicas de um script Frida válido, como `Java.perform` e `Interceptor.attach`.

Em seguida, o script é carregado por meio do Frida e executado no processo da aplicação-alvo. A saída da instrumentação é apresentada ao analista, permitindo verificar se o script atingiu o objetivo descrito na consulta, como capturar um dado sensível, observar uma operação interna ou contornar um mecanismo de proteção.

Quando o resultado não atende à tarefa solicitada, o analista pode iniciar uma nova tentativa, opcionalmente descrevendo o problema observado e, se desejar, sugerindo uma possível solução. O FridaForge incorpora esse feedback à nova entrada, juntamente com o script anterior, a saída produzida pelo Frida e quando disponível, os registros do logcat, orientando uma nova geração sem recoletar todo o contexto da aplicação. Dessa forma, a ferramenta automatiza a construção e a execução da instrumentação sem realizar um teste de penetração completo, mantendo o analista no controle da atividade.

3. Avaliação Experimental

3.1. Configuração Experimental

A avaliação do FridaForge foi conduzida em um emulador com Android API Level 36 (Android 16), configurado com permissões de *root* por meio do comando `adb root`. O Frida 17.4.1 foi utilizado como motor de instrumentação. A geração dos scripts foi realizada com o modelo Claude Sonnet 4, escolhido com base na experiência prévia da equipe de desenvolvimento em tarefas de geração de código, tendo sido adotado desde as etapas iniciais de desenvolvimento do FridaForge.

O conjunto experimental foi composto por dez aplicações intencionalmente vulneráveis, nove delas do catálogo de referência do OWASP Mobile Application Security Testing Guide (MASTG) [OWASP MASTG 2026], selecionadas para abranger diferentes tecnologias e classes de vulnerabilidade. A Tabela 1 apresenta o conjunto utilizado.

Tabela 1. Aplicações utilizadas na avaliação experimental.

ID	Aplicação	Ref.	Tarefas	Consultas
A1	AndroGoat	MASTG-APP-0001	15	20
A2	DIVA	MASTG-APP-0007	13	17
A3	DodoVulnerableBank	MASTG-APP-0008	8	8
A4	InsecureBankv2	MASTG-APP-0010	8	8
A5	OVAA	MASTG-APP-0013	6	6
A6	Finstergram	MASTG-APP-0016	6	6
A7	MASTestApp-NETWORK	MASTG-APP-0018	6	6
A8	BugBazaar	MASTG-APP-0029	6	6
A9	VulnForum	MASTG-APP-0031	7	7
A10	Damn Vulnerable Bank	EXT-001	8	8
	Total		**83**	**92**

Nota: algumas tarefas foram avaliadas com mais de uma consulta, variando o nível de especificidade.

As 83 tarefas correspondem a objetivos concretos de análise e instrumentação definidos a partir das vulnerabilidades, dos fluxos sensíveis e dos mecanismos de proteção de cada aplicação, incluindo contorno de proteções, interceptação de dados sensíveis, extração de informações armazenadas e observação de operações criptográficas e de comunicação.

Entre execuções consecutivas, o estado da aplicação foi reiniciado por meio dos comandos `adb shell am force-stop` e `adb shell pm clear`. Esse procedimento buscou reduzir a influência de estados residuais, dados persistidos e efeitos produzidos por execuções anteriores.

A classificação de cada execução seguiu um critério binário de sucesso ou falha: uma execução era bem-sucedida quando o script gerado executava corretamente e atingia integralmente o objetivo da tarefa, e era considerada falha caso não atingisse o objetivo solicitado ou produzisse comportamento incompatível com a consulta original.

3.2. Resultados

A Tabela 2 apresenta os resultados consolidados por aplicação, com taxa geral de sucesso de 95,7%.

Tabela 2. Resultados consolidados por aplicação.

Aplicação	Tarefas	Execuções	Sucessos	Taxa
A1 – AndroGoat	15	100	99	99,0%
A2 – DIVA	13	85	85	100%
A3 – DodoVulnerableBank	8	40	29	72,5%
A4 – InsecureBankv2	8	40	32	80,0%
A5 – OVAA	6	30	30	100%
A6 – Finstergram	6	30	30	100%
A7 – MASTestApp-NETWORK	6	30	30	100%
A8 – BugBazaar	6	30	30	100%
A9 – VulnForum	7	35	35	100%
A10 – Damn Vulnerable Bank	8	40	40	100%
Total	**83**	**460**	**440**	**95,7%**

Das dez aplicações avaliadas, sete alcançaram 100% de sucesso. As maiores concentrações de falhas ocorreram em DodoVulnerableBank, com 72,5%, e InsecureBankv2, com 80,0%.

A análise das 20 falhas identificou três padrões principais: interceptações em APIs de alto volume de chamadas, que podem sobrecarregar o processo sem atingir o ponto de interesse; incompatibilidades com bibliotecas legadas, como o Apache HttpClient, para as quais o modelo tende a gerar scripts baseados em APIs mais recentes; e lacunas no contexto coletado, como classes internas de *AsyncTask* ou sobrecargas de métodos não identificadas corretamente.

Os resultados também foram analisados segundo o nível de especificidade das consultas. As consultas específicas (25 execuções) e semi-específicas (40 execuções) obtiveram 100% de sucesso; as consultas genéricas representaram 395 execuções, das quais 375 foram bem-sucedidas (94,9%), concentrando todas as 20 falhas observadas. Isso sugere que consultas mais detalhadas reduzem a ambiguidade na escolha dos pontos de instrumentação, embora as genéricas permaneçam o cenário mais dependente do contexto coletado pelo FridaForge.

A consistência entre repetições também foi examinada: das 92 consultas, 89 produziram o mesmo resultado nas cinco execuções, e apenas três apresentaram resultados mistos, indicando impacto restrito da variabilidade do modelo, concentrado em tarefas com maior dependência da estrutura interna da aplicação.

3.3. Comparação com Ferramentas Existentes

A Tabela 3 compara o FridaForge com o Objection [Objection 2026] e o Frida CodeShare [Frida CodeShare 2026]. Uma tarefa foi considerada diretamente suportada quando a solução permitia executá-la sem exigir escrita ou adaptação manual de scripts.

Tabela 3. Comparação entre FridaForge, Objection e Frida CodeShare.

Critério	FridaForge	Objection	CodeShare
Cobertura de tarefas	83/83 (100%)	3/83 (3,6%)	0/83 (0%)
Automação	Ling. natural	Built-in	Manual
Contexto do app	Sim	Não	Não
Requer expertise Frida	Não	Parcial	Sim

O FridaForge permite submeter as 83 tarefas ao fluxo automático de geração e execução com contexto da aplicação. Esse valor representa cobertura funcional, e não taxa de sucesso: das 460 execuções realizadas, 440 atingiram o objetivo solicitado, correspondendo a 95,7%.

3.4. Limitações e Ameaças à Validade

A ferramenta depende de um ambiente Android preparado, com ADB, Frida e permissões adequadas. A avaliação utilizou aplicações intencionalmente vulneráveis, um único modelo de linguagem e um único ambiente experimental, de modo que os resultados não podem ser generalizados diretamente para aplicações comerciais, pois estas costumam apresentar mecanismos de ofuscação, detecção de instrumentação e componentes nativos, que podem dificultar a coleta de contexto e a identificação dos pontos de interesse. A natureza não determinística do modelo também pode produzir scripts distintos para

uma mesma consulta, ameaça mitigada pela repetição de cada consulta cinco vezes. Adicionalmente, o critério binário de sucesso foi definido e aplicado pela mesma equipe responsável pelo desenvolvimento da ferramenta e pela formulação das tarefas, o que introduz risco de viés de confirmação, parcialmente mitigado pela objetividade do critério adotado (execução correta do script e captura do alvo).

3.5. Demonstração e Disponibilidade

O FridaForge está disponível publicamente como software de código aberto. A demonstração apresenta o fluxo completo da ferramenta, desde a seleção da aplicação e a submissão da consulta em linguagem natural até a coleta do contexto, a geração do script, sua execução com o Frida e o refinamento após falhas de instrumentação. O código-fonte, a documentação e os materiais de demonstração estão disponíveis em: `https://github.com/FranciscoSalesCarvalho/android_instrumentation`. O vídeo de demonstração está disponível em: `https://drive.google.com/drive/folders/1adbSMc6c9pAS0T3duMDPOnGjah7tJcJK?usp=sharing`.

4. Conclusão

Este artigo apresentou o FridaForge, uma ferramenta de código aberto para apoiar testes de penetração e outras atividades de análise dinâmica de segurança em aplicações Android. A ferramenta recebe consultas em linguagem natural, coleta informações da aplicação em execução e utiliza esse contexto para gerar e executar scripts Frida adaptados ao alvo. Na avaliação com dez aplicações intencionalmente vulneráveis, 83 tarefas de segurança e 460 execuções, o FridaForge alcançou taxa geral de sucesso de 95,7%, reduzindo o esforço necessário para localizar pontos de instrumentação e desenvolver scripts específicos, inclusive em consultas genéricas.

Como trabalhos futuros, pretende-se ampliar a coleta de contexto para componentes nativos, avaliar outros modelos de linguagem e conduzir experimentos com aplicações comerciais e analistas de segurança.

Agradecimentos

O presente trabalho foi realizado com o apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (AUXPE-CAPES-PROEX) - Código de Financiamento 001. Adicionalmente, este trabalho foi parcialmente financiado pela Fundação de Amparo à Pesquisa do Estado do Amazonas - FAPEAM - por meio dos projetos PDPG-CAPES e POSGRAD 2025-2026 e POSGRAD 2026-2027.

Declaração de Uso de IA Generativa

Em conformidade com o Código de Conduta para Autores em Publicações da SBC, declaramos os seguintes usos de Inteligência Artificial Generativa neste trabalho: (i) o modelo Claude (Anthropic) é utilizado como componente central da ferramenta FridaForge para geração de scripts Frida, conforme descrito na Seção 2, constituindo parte do objeto de estudo do artigo; e (ii) ferramentas de IA generativa foram utilizadas como apoio à revisão linguística do manuscrito. Os autores assumem total responsabilidade pelo conteúdo final do trabalho.

Referências

Chen, M., Tworek, J., Jun, H., et al. (2021). Evaluating large language models trained on code. *ArXiv*, abs/2107.03374.

D'Elia, D. C., Coppa, E., Nicchi, S., Palmaro, F., and Cavallaro, L. (2019). Sok: Using dynamic binary instrumentation for security (and how you may get caught red handed). In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, ACM Asia CCS '19, pages 15–27, New York, NY, USA.

Frida (2026). Frida: A world-class dynamic instrumentation toolkit. `https://frida.re`. Acesso em: abril de 2026.

Frida CodeShare (2026). Frida CodeShare: Community Frida scripts repository. `https://codeshare.frida.re/`. Acesso em: abril de 2026.

MEDUSA (2026). MEDUSA: Mobile edge-dynamic unified security analysis. `https://github.com/Ch0pin/medusa`. Acesso em: abril de 2026.

Objection (2026). Objection: Runtime mobile exploration. `https://github.com/sensepost/objection`. Acesso em: abril de 2026.

OWASP MASTG (2026). OWASP MASTG – apps. `https://mas.owasp.org/MASTG/apps/`. Acessado em: abril de 2026.

Rozière, B., Gehring, J., Gloeckle, F., et al. (2023). Code llama: Open foundation models for code. *ArXiv*, abs/2308.12950.

Sutter, T., Kehrer, T., Rennhard, M., Tellenbach, B., and Klein, J. (2024). Dynamic security analysis on android: A systematic literature review. *IEEE Access*, 12:57261–57287.

Xu, H., Wang, S., Li, N., Wang, K., Zhao, Y., Chen, K., Yu, T., Liu, Y., and Wang, H. (2025). Large language models for cyber security: A systematic literature review. *ACM Trans. Softw. Eng. Methodol.*