Um Módulo de Autenticação para o Fiddle

Mairo Pedrini, Denise Stringhini, Philippe Olivier
Alexandre Navaux

Universidade Federal do Rio Grande do Sul
Instituto de Informática e PPGC
Av. Bento Gonçalves, 9500 - Caixa Postal 15064 - CEP 90501-970
Porto Alegre, RS

{mpedrini,navaux}@inf.ufrgs.br
dstring@terra.com.br

Introdução

Em se tratando de programação paralela e distribuída, uma grande deficiência é, ainda hoje, a baixa quantidade de ferramentas de depuração, bem como a falta de padronização em suas interfaces com o usuário e, também, a falta de interfaces padronizadas para acesso, por parte das ferramentas, às bibliotecas de programação.

É no intuito de reduzir esta deficiência de ferramentas que está sendo desenvolvido o projeto da PADI [STR 02]. A PADI é uma interface gráfica para depuração paralela e distribuída, que oferece mecanismos de seleção e agrupamento de processos, de forma simples e de fácil compreensão. A PADI está montada sobre um pacote para depuração, o Fiddle [LOU 03].

O Fiddle consiste de um conjunto de bibliotecas e servidores, construídos de modo a auxiliar na criação e integração de novas ferramentas de depuração. O Fiddle atua como uma camada intermediária entre interfaces de depuração (clientes Fiddle) e ferramentas sequenciais de depuração (os *node debuggers*), oferecendo uma forma única de acesso, bem como um ponto central de controle para múltiplos processos e o compartilhamento de sessões de depuração entre múltiplos clientes.

Este trabalho apresenta o desenvolvimento de uma solução para um problema do Fiddle, problema este que dificulta seu uso por mais de um usuário, bem como expõe o ambiente a um risco de segurança. Nas próximas sessões, será feita uma caracterização do Fiddle e descrição das causas do problema, juntamente com a forma encontrada para resolvê-lo.

Arquitetura do Fiddle

Em seu projeto, o Fiddle é dividido em quatro camadas, cada qual usando a anterior e agregando novas funcionalidades. Qualquer destas camadas pode ser utilizada por um cliente.

A primeira camada, camada 0s, é responsável pela comunicação com o *node debugger*. Esta camada é responsável por efetuar a interpretação das saídas do *node debugger* e também por enviar comandos devidamente formatados para o mesmo. Esta camada

já permite que múltiplos processos sejam depurados, embora eles devam ser locais, e o cliente seja restrito a uma única *thread*. A camada 0m, por sua vez, apenas adiciona suporte à clientes com múltiplas *threads*.

A camada 1m é responsável por tratar de processos distribuídos, permitindo que os clientes especifiquem em qual máquina da rede o processo deve ser disparado. Desta forma, o controle de uma aplicação distribuída fica centralizado em um único ponto, libertando os clientes da complexidade de criar mecanismos próprios para controlar a distribuição dos processos.

O objetivo da camada 2m é permitir que ferramentas distintas possam trabalhar em conjunto, possivelmente oferecendo visões complementares sobre a aplicação (por exemplo é possivel executar ume depuração *online*, e ao mesmo tempo capturar um rastro da execução). Esta camada será vista com mais detalhes na seção seguinte, por ser ela o alvo deste trabalho.

Existe ainda um nível 3m, que ainda não se encontra totalmente definido e implementado. Este nível propõe uma semântica orientada a eventos para o Fiddle, permitindo que os clientes possuam um conhecimento compartilhado sobre a execução do programa.

Camada 2m: Problemas decorrentes do funcionamento interno

Conforme explicado acima, a camada 2m permite que múltiplos clientes utilizem o Fiddle simultaneamente. Entretanto, se esta é uma característica bastante desejável no Fiddle, ela também é fonte de algumas complicações. Em especial, a necessidade de que a sessão de depuração seja *compartilhada* por mais de um cliente, que força o descarte do uso de múltiplos servidores independentes.

A solução implementada no Fiddle é o uso de um único servidor com múltiplas *threads*, capaz de comunicar-se com os clientes através de *sockets* TCP/IP. O uso de *threads* garante o compartilhamento da sessão, ao passo que o uso de TCP/IP permite a conexão de múltiplos clientes. Esta solução possui ainda uma outra vantagem: ela permite que os clientes rodem em máquinas diferentes uns dos outros, e diferentes do servidor, permitindo um comportamente ainda mais distribuído do conjunto todo (clientes + servidor + aplicação).

Entretanto, esta implementação baseada em *sockets* sofre de dois problemas não muito óbvios. O primeiro problema decorre do fato de o servidor do Fiddle ficar escutando em uma porta pré-definida. Assim, embora a sessão de depuração possa ser compartilhada por várias ferramentas, ela é única naquela máquina onde o servidor está executando. Assim, apenas um usuário pode utilizar o Fiddle por vez.

O servidor do Fiddle fornece uma solução não muito elegante para este problema, que consiste em uma opção de linha de comando para informar em qual porta o servidor deve ficar escutando. Assim, cada usuário colocaria seu servidor para escutar em uma porta específica. Esta solução é visivelmente trabalhosa se o grupo de usuários da ferramenta não for reduzido.

O segundo problema, no entanto, é o mais grave e, para este, não existe solução implementada no Fiddle. Todo e qualquer cliente que se conectar ao servidor do Fiddle

tem acesso total às funções de depuração, que executam com os privilégios do usuário que iniciou o servidor. Esta permissividade configura uma séria brecha de segurança, algo totalmente indesejável em um ambiente de rede. Naturalmente, a configuração correta de um *firewall* dificultaria ataques externos, mas não existe solução para ataques interno.

Uma Proposta de Solução

A resolução do controle de acesso ao servidor do Fiddle pode ser resolvido através de uma identificação e validação de qual usuário está requisitando a conexão, juntamente com a manutenção de uma lista de usuários a quem é permitido conectar àquele serviço. O modelo classicamente utilizado faz esta identificação através de um *login* e uma senha.

Duas formas principais de utilizar este modelo foram pensadas, a saber: manter uma senha por servidor ativo, ou manter uma senha por usuário do sistema. Cada qual possui suas vantagens e desvantagens.

O uso de uma senha por servidor ativo seria a solução mais simples de implementar: O servidor, no início, requisitaria a senha a ser utilizada, e cada cliente que se conectar deve responder esta senha. Esta implementação possui a vantagem de necessitar pouquíssima alteração no Fiddle e, Também, a senha utilizada não necessita ser a mesma do usuário no sistema.

A segunda alternativa, o uso de uma senha por usuário, é a forma convencionalmente utilizada por outros serviços, como *ftp*, *ssh*, *e-mail*, etc. É assim por ser capaz de resolver tanto o problema da autenticação quanto o problema dos múltiplos usuários. Nesta solução, o sistema mantém uma tabela mapeando usuário, senha e identificador de usuário. Quando de uma conexão, o servidor requisita usuário e senha, confere-os na tabela e, se estiver correto, inicia um novo servidor, com os privilégios daquele usuário (utilizando o identificador). Este aspecto torna esta implementação mais interessante que a anterior e, por isso mesmo, foi a escolhida para este trabalho.

Entretanto, da forma como é normalmente implementado, este modelo **não** serve para o Fiddle, visto que cada conexão é repassada para um novo servidor, ao passo que o Fiddle requer que cada conexão de um usuário x seja repassada para um servidor S_x. A adaptação deste modelo para o Fiddle consiste em armazenar, na parte responsável pela autenticação, que servidores estão ativos e qual o usuário associado a cada um deles. Assim, após identificar-se, se não houver servidor ativo para este usuário, um é disparado, e a conexão é repassada para o servidor associado ao usuário. Esta é, exatamente, a solução proposta neste trabalho.

Detalhes da implementação

Inicialmente, e objetivando reduzir esforços de manutenção do serviço, manter a homogeneidade do ambiente e simplificar o desenvolvimento, optamos por utilizar os módulos de autenticação atualmente em uso em muitas aplicações para Linux, módulos estes fornecidos pela biblioteca PAM [PAM 04]. Esta biblioteca fornece uma interface padronizada e modular para autenticação. Assim, os usuários e senhas utilizados para autenticação no Fiddle são os mesmos utilizados no sistema.

O repasse das conexões recebidas pelo módulo de autenticação para o servidor do Fiddle foi a parte mais complexa do projeto. As alternativas pensadas incluiam multiplexar as diversas conexões dos clientes, em uma única conexão com o servidor do Fiddle. Entretanto, esta solução exige que as conexões sejam multiplexadas e demultiplexadas, e também que os dados sejam repassados byte-a-byte, tarefa onerosa para o sistema.

Também seria possível manter o servidor do Fiddle utilizando TCP/IP, sendo que o módulo de autenticação implementaria alocação automática de portas, e gerenciaria o repasse das conexões. Embora elimine a necessidade de multiplexação/demultiplexação das comunicações, esta solução novamente abre portas TCP/IP no servidor do Fiddle, e continua exigindo a transferência byte-a-byte dos dados entre o autenticador e o servidor do Fiddle.

Por fim, a solução implementada para este problema utiliza uma funcionalidade bastante interessante de *sockets* UNIX (um tipo de *sockets* utilizados para comunicação entre processos locais)A funcionalidade em questão é que este tipo de *socket* permite que descritores de arquivos sejam transferidos de um processo para o outro. Assim, o módulo de autenticação repassa, através de um *socket* UNIX, o *socket* TCP/IP dos clientes para o servidor do Fiddle. Esta solução não sofre com nenhum dos problema das outras alternativas.

Conclusões e continuação dos trabalhos

A solução descrita foi implementada, e uma interface básica para autenticação foi inserida na PADI para fins de teste. O módulo se comportou dentro do esperado, validando assim seu uso. A continuação dos trabalhos neste módulo prevêem a correção de alguns *bugs* menores, bem como melhorias no desempenho e no código.

Outro aspecto a ser tratado é no que diz respeito a confidencialidade das informações transferidas. É uma preocupação importante o fato de informações sensíveis serem enviadas pela rede sem proteção, como é atualmente o caso dos *tokens* de identificação. Algumas alternativas estão sendo estudadas, embora atualmente o mais interessante pareça ser o uso de um túnel SSL ([STU 04], por exemplo) para proteger toda a comunicação.

Referências

[LOU 03] LOURENÇO, J.; A Debugging Engine for Parallel and Distributed Programs. Tese (Doutorado) – Departamento de Informática, Universidade Nova de Lisboa, Lisboa, 2003

[PAM 04] Linux-PAM project. Pluggable Authentication Modules for Linux, disponível em: http://www.kernel.org/pub/linux/libs/pam. Acesso em: 06 outubro 2004.

[STR 02] STRINGHINI, D. Depuração de Programas Paralelos - Projeto de uma Interface Intuitiva. Tese (Doutorado) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2002.

[STU 04] STUNNEL. **Stunnel** – Universal SSL Wrapper, disponível em: http://www.stunnel.org. Acesso em: 06 outubro 2004.