Segurança de arquivos e metadados do Firebird

Geoff Worboys

3 Março 2008 – Versão do documento 0.5-pt_br

Tradução para o Português do Brasil: Gabriel Frones

Índice

Introdução

Se você veio parar nessa página e não sabe nada sobre Firebird, veja este link: www.firebirdsql.org

Este artigo visa discutir a questão de segurança dos arquivos de base de dados do Firebird e, particularmente, o acesso aos metadados armazenados nestes, tendo sido escrito em resposta às frequentes perguntas sobre o tema em listas de discussões. O artigo evita especificações técnicas. Para ler sobre como proteger arquivos no seu sistema operacional, especificamente, procure pela documentação relevante a este sistema operacional. Para ler sobre como configurar a segurança de usuários e objetos em sua base de dados Firebird, procure pela documentação disponível no website do projeto (linkado acima) ou compre o livro *Dominando o Firebird* (*The Firebird Book*, no original, em inglês) da autora Helen Borrie.

O Funcionamento em Segundo Plano

Para que uma aplicação (usuário) possa acessar um banco de dados Firebird, ela deve conectar-se a um servidor Firebird que contenha este arquivo. Ao receber uma requisição de conexão, o servidor automaticamente verificará as permissões do usuário definidas no arquivo de segurança (normalmente, isc4.gdb). Se a autenticação for bem sucedida, então o servidor dará à aplicação a permissão de acessar qualquer banco de dados que ela requisitar e então, utilizar os privilégios e roles definidos neste banco de dados para garantir um acesso controlado aos objetos nesse banco de dados.

Em momento algum o usuário que conecta ao banco de dados necessita de acesso diretamente ao arquivo em si. Todo o acesso se dá através do aplicativo servidor, que acessa os arquivos conforme necessário para atender às requisições dos clientes. Fica, portanto, a cargo do servidor restringir o acesso aos usuários logados, conforme as permissões definidas para este usuário naquele banco de dados.

> **Nota**
>
> A versão “embedded” do Firebird funciona de forma diferente e não é apropriada para instalações seguras. O fato de a versão embedded existir não anula os pontos discutidos neste artigo sobre segurança de banco de dados, mas ressalta a importancia de se usar um ambiente de segurança efetivo em instalações seguras. Mais detalhes sobre a versão embedded serão discutidos mais adiante.

Todo servidor Firebird possui um usuário “SYSDBA”, que possui acesso irrestrito a qualquer banco de dados disponível a esse servidor. Os privilégios especificos dos bancos de dados são ignorados quando este é acessado por esse usuário. Quando você copia um banco de dados para dentro de um servidor, você pode utilizar este usuário para determinar os privilégios de acordo com os usuários e as necessidades deste servidor. No entanto, dessa forma, se eu tiver acesso ao arquivo do banco de dados, eu posso copiar esse arquivo para um servidor onde eu saiba a senha do usuário SYSDBA, e obterei acesso irrestrito ao arquivo.

Como você pode notar, a segurança do Firebird presume que seu servidor estará sendo executado em um ambiente de segurança apropriado. O Firebird em si não toma precaução alguma para evitar riscos externos. Se uma pessoa tem acesso físico ao arquivo de banco de dados, não há uma maneira efetiva para prevenir que o usuário leia toda a informação (dados e metadados) contida neste arquivo.

> **Nota**
>
> A segurança “adequada” depende do nível de segurança que é necessário ao ambiente em questão. Este quesito varia muito de instalação para instalação.

Isto significa que, para uma segurança razoável, toda instalação deve manter os arquivos do banco de dados adequadamente seguros. Na maioria dos casos, isto significa que o servidor Firebird deve rodar em um usuário específico no sistema operacional, e apenas esse usuário (e provavelmente o administrador/root) deve ter acesso direto aos arquivos do banco de dados. Estes arquivos não devem ficar em diretórios compartilhados na rede ou que sejam acessíveis por qualquer pessoa que não o pessoal autorizado.

Para uma segurança ainda mais efetiva, é preferível que o computador que será o servidor fique armazenado em um local físico de acesso restrito, de forma a prevenir que alguém tente acessar os arquivos por trás das restrições do sistema operacional.

No entanto, a explicação acima não necessariamente ajuda desenvolvedores que, tendo desenvolvido um banco de dados para distribuição e instalação em locais remotos, desejam proteger as propriedades intelectuais contidas nestes arquivos. Tais preucupações podem incluir desde os metadados (estruturas, relacionamentos, stored procedures e triggers) até os dados em si, incluídos em algumas tabelas. É este o ponto que representa o foco deste artigo.

O Problema

Um desenvolvedor cria um banco de dados (e, normalmente, um aplicativo cliente que o acompanha) para instalação em locais remotos. Nestes locais, é natural que ao menos uma pessoa de lá tenha acesso irrestrito ao computador no qual o servidor Firebird será executado - para poder executar tarefas como backup's e manutenção. Como descrito no tópico acima, acesso direto aos arquivos permitem que se ganhe acesso completo e irrestrito a toda a informação das tabelas (dados e metadados).

Nestes casos, o desenvolvedor pode não confiar que os usuários deste local respeitarão a propriedade intelectual que este banco de dados representa. Então, fica o receio de que o usuário faça a engenharia reversa do banco de dados por interesse próprio, ou que não haja, neste local, uma preocupação real em manter seguros os arquivos a acesso de terceiros.

Isto nos leva às frequentes perguntas nas listas de Firebird como:

“Eu quero—”
: “—proteger os metadados de meu banco de dados (estruturas, stored procedures, triggers etc.) de todos os usuários em uma instalação remota. Como posso fazer isso com Firebird?”

“Eu quero—”
: “—impedir que qualquer usuário de uma instalação remota tenha acesso a estas tabelas em particular. Elas contêm informação proprietária que é usada internamente pelo aplicativo.”

O Firebird (pelo menos até a v1.5) não oferece nenhuma facilidade integrada de encriptação. A resposta simples às questões acima é que isto não pode ser feito com as atuais capacidades do Firebird. Um usuário que tem acesso direto ao arquivo, tem acesso irrestrito às informações dele.

Na primeira questão, nenhum arranjo pode ser feito, porque o servidor em si deve ser capaz de ler os metadados da tabela. Na segunda questão, é possível implementar rotinas de encriptação/desencriptação no aplicativo cliente, mas neste caso, você perde a capacidade de fazer um uso efetivo dos índices e das facilidades de busca do Firebird - e o gerenciamento de senhas de encriptação torna-se um problema ainda maior (mais adiante).

A Solução

Existe apenas uma solução possivel para esses problemas, na realidade: Armazenar o banco de dados em seu próprio computador e usá-lo como servidor remoto, para permitir que os clientes conectem ao banco através da internet. Terminal server (Windows ou Linux/Unix) é uma alternativa interessante para implementar esta estrutura.

Desta forma, você fica com o controle sobre o arquivo de banco de dados e pode restringir o acesso às suas informações utilizando as ferramentas internas de segurança do Firebird (roles e privilégios, etc.).

Dificuldades

É importante ressaltar que você pode encontrar dificuldades até mesmo nesta situação, se sua intenção é proteger a estrutura do banco de dados.

Necessidades da Camada de Acesso

Muitas das bibliotecas de acesso a banco de dados fazem consultas aos metadados da tabela, como chave primária, domínio e outras informações da estrutura, visando tornar o desenvolvimento das aplicações clientes uma tarefa mais simples. Se impedimos os usuários de acessar os metadados da tabela, consequentemente, impedimos também a aplicação de recuperar as informações que precisa.

Isto nos leva a duas alternativas:

- permitir que seus metadados “escapem” do servidor através de uma sofisticada interface de acesso; ou
- gastar o tempo necessário para se desenvolver uma aplicação utilizando uma biblioteca de acesso menos sofisticada.

“Vazamento” por Inferência e Dedução

Há também o problema de que muitas aplicações clientes, inerentemente, deixam escapar informações sobre a estrutura do banco de dados com a qual interagem. É muito raro encontrar uma aplicação de banco de dados que tenha uma interface que não revele detalhes sobre a estrutura das tabelas que usa.

Alguns detalhes podem ser escondidos por meio de Views e Stored Procedures, mas definir essas ferramentas apenas para tentar esconder a informação da estrutura é uma tarefa frustrante. Provavelmente será inútil, pois alguns detalhes escaparão, não importa o que você tente.

Protegendo as Informações do Usuário

Antes de continuar discutindo questões relacionadas à encriptação das informações do banco de dados, eu gostaria de ressaltar que é perfeitamente possível proteger as informações dos usuários por meio de encriptação. Não serve para desenvolvedores que queiram esconder informações de usuários legítimos, mas pode servir para suprir as necessidades de privacidade que seus clientes possam querer com relação ao banco de dados.

Existem algumas situações em que é impraticável colocar o servidor Firebird sob uma infra-estrutura adequadamente segura. Em períodos em que o escritório esteja funcionando, é bem pouco provável que alguém poderá acessar o computador para copiar os arquivos do banco de dados (ou até mesmo roubar o computador ou disco rígido para obter os arquivos posteriormente). No entanto, fora do horário normal de trabalho (noites e finais de semana), a situação pode ser um pouco diferente. Alguém poderia conseguir entrar no escritório e roubar o disco rígido do computador (ou levar o computador inteiro) e levar a algum lugar onde possa obter acesso ao banco de dados.

Encriptação

O Firebird em si não disponibiliza nenhuma ferramenta de encriptação integrada, mas existem excelentes produtos que podem fornecer essa solução. Você poderia instalar um software que crie uma partição virtual encriptada em seu computador e colocar os arquivos do banco de dados (e qualquer outra informação confidencial) nessa partição. Dessa forma, toda a informação ficará armazenada em um arquivo encriptado e ninguém poderá acessá-la a menos que tenha a senha. Sempre que você iniciar o computador, terá que montar a partição virtal encriptada e fornecer a senha secreta para poder acessar as informações. Esse processo manual de inicialização pode ser um pouco inconveniente, mas pode proporcionar uma excelente segurança a computadores que não são constantemente vigiados.

Alguns softwares com essas características são: TrueCrypt (www.truecrypt.org), Bestcrypt da Jetico (www.jetico.com) e PGPDisk (www.pgpi.org/products/pgpdisk/– note que este link te levará a uma antiga versão gratuita. Nesta mesma página existem links para as versões comerciais mais novas). Existem outros, mas estes dois últimos são os que eu mesmo usei.

Por que o Firebird não possui encriptação nativa?

Devido às necessidades descritas anteriormente, é muito comum encontrar usuários pedindo que o Firebird, em um versão futura, tivesse a capacidade de encriptar metadados, dados do usuário, ou até mesmo o banco de dados todo. Como não sou um desenvolvedor do núcleo do Firebird, não posso afirmar categoricamente que não vai acontecer. Mas a questão aqui não é realmente se a encriptação é praticável ou útil, mas se o gerenciamento de senhas de encriptação proporcionaria uma solução para os problemas que estamos examinando.

A encriptação em si pode apenas ser tão boa quanto a senha de desencriptação. Pode ser ainda pior, mas não pode ser melhor. Existem diversos excelentes algoritmos de encriptação disponíveis que poderiam ser utilizados. Quando a encriptação é boa, os ataques costumam centralizar-se contra a senha e não contra a encriptação em si.

Como funcionaria a encriptação?

Então vamos analisar como funcionariam as coisas se o Firebird proporcionasse maneiras de encriptar metadados no banco de dados...

Antes que um banco de dados pudesse ser acessado, a senha deveria ser fornecida. Dar a senha ao usuário seria irrelevante, pois nos levaria de volta ao problema original. Então podemos presumir que, sempre que o cliente reiniciar o servidor, teria que chamar o desenvolvedor, que poderia então entrar com a senha. Mesmo que isso fosse praticável, não necessariamente resolveria o problema. Por exemplo: o cliente poderia instalar algum tipo de software para monitorar o servidor e roubar a senha.

Existem soluções baseadas em hardware para proporcionar a senha para o processo de desencriptação. Mas novamente, este dispositivo precisaria ficar sob a posse do cliente, e se não confiamos nele, não podemos impedí-lo de obter acesso ao banco de dados em outro computador onde a senha do usuário SYSDBA é conhecida.

O Firebird é um produto Open Source. Se as facilidades de encriptação forem nativas ou através de bibliotecas adicionais também open source, seria possível aos usuários desenvolverem suas próprias versões do servidor ou da biblioteca que não apenas providencie as capacidades de encriptação e desencriptação do banco de dados, mas também possa mostrar a senha na tela ou simplesmente mostre os resultados desencriptados diretamente. O desenvolvedor, que não tem o controle do servidor, não pode detectar e nem prever tal atividade.

Você pode vir a considerar a construção de sua própria versão do Firebird server, com a senha de desencriptação escondida no executável. Mas descompiladores são facilmente encontrados. Não levaria muito tempo para descobrir a senha por uma simples comparação das versões descompiladas da sua versão do servidor e da versão normal.

Existem vários bancos de dados que existem com o propósito de proporcionar uma forte encriptação. Talvez a encriptação seja forte, mas a menos que o gerenciamento de senhas sesteja no local para suportar essa ferramenta, a encriptação não alcançará o efeito desejado. Pode até encorajá-lo a acreditar que está protegido, mas você precisa estudar o gerenciamento de senhas para descobrir se é realmente verdade.

A dura realidade é que, se você não tem o controle sobre o hardware no qual o processo de encriptação e desencriptação ocorre, nunca estará protegido. Se a senha de desencriptação não pode ser mantida em segurança, então até mesmo as melhores encriptações tornan-se apenas um pouco mais que segurança por obscuridade.

Limitando a distribuição da informação

Algumas pessoas pedem a encriptação dos dados, para que eles possam tentar limitar a disseminação destes. Elas não se importam com o fato de que um usuário autorizado em particular veja a informação, mas querem, no entanto, limitar a capacidade deste usuário de distribuir essa informação a outras pessoas.

Vamos imaginar por um momento que todos os problemas com o gerenciamento de senhas descritos acimas foram resolvidos, então torna-se impossível ao usuário apenas copiar o banco de dados para poder obter as informações. Nestes casos, o usuário poderia simplesmente escrever um pequeno programa que extraísse as informações que lhe forem interessantes (através do servidor legítimo, instalado) e copiá-las para seu próprio arquivo ou banco de dados.

Eu acredito que seja possível que o Firebird proporcione, no futuro, algum tipo de sistema de autenticação da aplicação que poderia limitar essa maneira de extrair as informações, mas ainda assim, os mesmos problemas persistem. Se você não tem controle sobre o servidor, não pode prevenir que o usuário instale uma versão que não requer esta autenticação.

Removendo o acesso do usuário SYSDBA

Várias vezes, algumas pessoas sugeriram que remover o acesso do usuário SYSDBA a esse banco de dados poderia ser a solução. A idéia por trás desta solução é que mover um banco de dados para um outro servidor onde a senha do usuário SYSDBA é conhecida não ajuda em nada, pois o usuário SYSDBA não tem acesso às tabelas. Algumas pessoas relataram um limitado sucesso nesse contexto, criando uma role SQL com o nome de SYSDBA e assegurando que ela não terá acesso aos objetos do banco de dados.

Mas isso não resolve realmente o problema. O arquivo do banco de dados pode ser visualizado com um editor hexa-decimal ou similar e a lista de usuários disponíveis, descoberta (Descobrir os nomes dos usuários donos dos objetos seria particularmente útil). Dessa forma, estes nomes podem ser adicionados ao novo servidor e usados diretamente.

E um arranjo ainda mais simples seria utilizar a versão embedded do servidor Firebird (mais adiante) ou compilar uma versão própria que ignore as diretivas de segurança.

Nomes personalizados para SYSDBA

Também houveram sugestões de permitir que o nome de usuário SYSDBA seja alterado. Esta solução poderia oferecer alguma proteção limitada contra ataques de força bruta contra a senha do SYSDBA, visto que o ataque teria que adivinha tanto o nome de usuário quanto sua senha. Mas não ajuda em nada a proteger o sistema de uma pessoa que tenha acesso direto ao arquivo.

Excluindo o código fonte das SP's e Triggers

Quando você escreve e define uma Stored Procedure ou Trigger para o banco de dados Firebird, o servidor armazena uma cópia quase completa do código fonte juntamente com a cópia compilada, chamada de BLR (Binary Language Representation). É a BLR que é executada pelo servidor, o código fonte não é utilizado.

Alguns desenvolvedores tentam proteger ao menos uma parcela de seus metadados deletando o código fonte antes de distribuir o banco de dados (apenas um update direto nos campos relevantes da tabela de metadados). Eu recomendo não adotar esta prática por duas razões:

1. BLR é uma codificação muito simples do código fonte. Não seria muito difícil decodificar o BLR de volta para uma forma legível a humanos. A decodificação não traria comentários ou formatação, mas as SQL's que vão em SP's ou Triggers são raramente tão complicadas a ponto de isso causar problemas. Então, a proteção oferecida pela remoção do código fonte não é muito significante.

2. O código fonte pode ser útil para outros propósitos. Ele permite que correções sejam aplicadas diretamente no banco de dados, sem precisar trazer de volta o código fonte completo de algum outro lugar (e então, lembrar de remover novamente após aplicar as correções). O código fonte é também utilizado em vários utilitários, como o meu DBak - uma ferramenta de backup alternativa ao “gbak”. Eu não me importei em escrever meu próprio decodificador de BLR neste momento, portanto, DBak depende da disponibilidade do código fonte para conseguir criar o script DDL para reconstruir o banco de dados.

Servidor Firebird Embedded

Há uma versão especial do servidor Firebird chamada de “embedded”. Esta versão é, na verdade, uma biblioteca cliente especial, que inclui o servidor em si. Quando um aplicativo chama essa biblioteca, ela carrega o servidor e permite acesso direto a qualquer banco de dados acessível ao computador local. Dessa forma, não faz uso do banco de dados de segurança. O nome de usuário especificado durante o “logon” (não existe autenticação de senha) é utilizado para gerenciar o acesso aos objetos do banco de dados (por permissões SQL), mas se este usuário for SYSDBA (ou o dono do banco de dados), então, o acesso irrestrito é possível.

As características do servidor embedded são úteis a desenvolvedores que querem criar aplicativos monousuários simples de se distribuir e que não requerem muita segurança.

Dessa breve descrição, pode parecer que ter um aplicativo com servidor embedded instalado em um servidor que esteja armazenando outros bancos de dados pode representar um grave risco à segurança. Na realidade o risco não é maior do que se o servidor embedded não existisse.

Quando um aplicativo carrega o servidor embedded, ele opera no contexto de segurança do aplicativo (e portanto, do usuário). O que significa que o servidor embedded terá acesso apenas aos arquivos que o usuário pode acessar

diretamente pelo sistema operacional. Conceder a um usuário suspeito acesso para instalar programas em um servidor seguro já não é boa idéia, mas se você tiver especificado permissões de arquivo apropriadas em arquivos de banco de dados seguros, o servidor embedded não representa uma ameaça.

A ameaça vem de todas as outras coisas que o usuário poderia instalar.

O fato de o servidor embedded existir serve apenas para ressaltar que é possível acessar diretamente as informações de um arquivo de banco de dados, especialmente em um ambiente open source. Se já não existisse, certamente seria possível que alguém compilasse um equivalente.

Outras Formas de Obscuridade

Várias outras formas de segurança por obscuridade foram propostas. Coisas como eventos especiais que são acionados no login e no logoff para chamar funções de usuários que liberem ou neguem acesso. Essas implementações podem oferecer alguma segurança limitada em sistemas de código fonte proprietário, onde a obscuridade da implementação ajuda a esconder como, exatamente, a informação está sendo protegida. Mas para um sistema open source, basta que se compile uma nova versão do servidor para pular os eventos ou códigos que estão prevenindo o acesso para acessar totalmente a tabela. É muito difícil oferecer obscuridade em um sistema open source.

Considere, por exemplo, o que acontece quando você distribui seus executáveis compilados. Executáveis compilados são ótimos exemplos de obscuridade. Não há uma encriptação (normalmente), e todos os passos do código estão lá para serem analisados por qualquer um que tenha o tempo e o conhecimento, e além disso, ainda existem descompiladores que podem auxiliar o processo. Tão logo uma pessoa descubra com que bibliotecas seu código foi compilado, isolar os resultados para apenas o código “secreto” fica muito fácil. Você já escreveu à Borland, Microsoft ou qualquer outro solicitando algum tipo de encriptação dos binários compilados?

Baixa Segurança Aceitável

Meus comentários até então se direcionaram a um ideal de forte segurança, e talvez o conceito de segurança por obscuridade tenha sido escrito com algum desprezo. No entato, eventualmente uma baixa segurança é suficiente, quando a informação não é assim tão valiosa. Você quer apenas impedir uma navegação casual e ao menos deixar o trabalho um pouco inconveniente aos “ladrões” mais avançados.

Eu mesmo já utilizei esse tipo de esquema várias vezes. Frequentemente, não tem porque utilizar Twofish, AES ou qualquer outro porque eles são apenas forte encriptação. São pesados, causando sobrecarda do processador e complicação com relação a manter uma forte segurança. Um simples XOR contra uma string conhecida (uma senha) já deve ser suficiente. Se a chave puder ser descoberta pelo ladrão, então não importa se a encriptação é fraca ou forte, a brincadeira de qualquer forma.

> **Nota**
>
> A maior parte dos algoritmos simples baseados em XOR podem ser facilmente quebrados. Consulte um bom guia de referencia sobre encriptação para mais informaçoes e opções.

Optando por Obscuridade

O problema com segurança por obscuridade é que ela precisa ser obscura. Se o Firebird implementasse algum tipo de encriptação em suas rotinas de leitura e escrita em disco, ela não seria obscura, pois este é um projeto Open Source. Não levaria mais que alguns minutos para recompilar o código fonte para descobrir a senha utilizada e então, tudo está perdido.

Então, se você realmente precisa dessa ferramenta, você teria que copiar o código fonte do Firebird, inserir suas próprias rotinas de obscuridade e compilar sua própria variação do servidor Firebird. (O código fonte ainda pode ser descompilado, mas para isso seria necessário um avançado conhecimento de programação.)

Antes de tentar fazer isso, verifique se realmente resolveria o problema se o usuário pudesse obter uma cópia dos executáveis especiais compilados junto com o banco de dados. Ou se ainda seria possível ao usuário extrair os segredos diretamente de seu servidor.

O Argumento Filosófico

Há também a seguinte questão filosófica: Por que você escolheria um banco de dados Open Source para construir um produto com banco de dados proprietário. Muitas pessoas contribuiram com o projeto com a crença de que a melhor forma de se fazer software é Open Source.

Mais particularmente, quando se trata de armazenamento de informações do usuário, eu acredito que os usuários precisem ter a habilidade de acessar sua própria informação - o que normalmente inclui a necessidade de entender a estrutura e os processos que você construiu (metadados). Se você sair do mercado ou ficar, de alguma forma, indisponível, será de crítica importancia que o usuário consiga ao menos extrair sua própria informação (em formatos apropriados) de forma a conseguir migrar para sistemas alternativos.

Você pode confiar em seus usuários para respeitarem sua propriedade intelectual enquanto você ainda estiver no mercado e disponível? Disponibilize os serviços e facilidades necessários e provavelmente, eles respeitarão. Caso contrário, então há uma grande chance de que não há nada que você possa fazer.

Conclusões

O problema é que muitas pessoas não entendem de segurança e como é difícil fazer bem feito. Lamentavelmente existem muitos softwares que estimulam esses desentendidos por implementar obscuridade no lugar de segurança de verdade. Observe o número de empresas que existem que oferecem serviços de “recuperação de informação”, que na verdade, nada mais é que *quebrar a suposta segurança da informação protegida por obscuridade*.

Encriptação não é um milagre da segurança. Se você não tiver o controle sobre a estrutura (o hardware, o sistema operacional e todos os softwares neste sistema), você não terá controle sobre a segurança - não importa que esquemas de encriptação você implemente no local. Essa é a situação que você tem quando precisa distribuir seu banco de dados para instalações externas.

Se você realmente precisa proteger as informações (dados e metadados) de seu banco de dados, então você precisa reter o controle do arquivo do banco de dados e da estrutura em que ele será acessado. Nenhuma outra solução oferece o mesmo nível de segurança.

Agradecimentos

Eu gostaria de agradecer a todas as pessoas que avaliaram e comentaram este artigo. Gostaria também de agradecer as pessoas que contribuem para a lista de suporte do Firebird, que é fonte de grande parte das informações disponíveis neste artigo.

Apêndice A: Histórico do Documento

O histórico exato do arquivo - começando pela versão 0.5 - está gravado no módulo `manual` na nossa árvore do CVS; veja http://sourceforge.net/cvs/?group_id=9028

Histórico de Revisões

N/A	14 Fev 2005	GW	Primeira edição.
N/A	11 Abr 2005	GW	A seção “Baixa Segurança Aceitável” foi revisada para tentar ressaltar que algoritmos XOR simples são fracos demais, para assegurar que os leitores investiguem mais a fundo, se estiverem interessados nessa solução.
N/A	26 Apr 2005	GW	Seção adicional sobre o servidor Embedded (e referencias a ela). Movidas notas de rodapé para uma nota em itálico, notas de rodapé não funcionam bem em HTML. Adicionado TOC.
N/A	4 Dez 2005	GW	Adicionada referencia ao TrueCrypt. Adicionada seção *Uso deste documento*. Adicionada seção de agradecimentos.
0.5	7 Dez 2005	PV	*Histórico do Documento* e *Uso deste Documento* movidos para apêndices. Adicionado número de versão para uso juntamente com o projeto Firebird. Documento adicionado ao repositório CVS do Firebird.
0.5-pt_br	3 Mar 2008	GF	Tradução para o Portugues do Brasil por Gabriel Frones.

Apêndice B:
Uso deste Documento

Eu tentei fazer este documento o mais preciso possível no momento de escrita, mas não posso garantir que não hajam enganos. Segurança é um assunto complexo. Onde segurança é importante ao seu produto ou instalação, você deveria procurar orientação profissional.

Eu não imponho nenhuma restrição em particular ao uso deste documento. Você está livre para reproduzir, modificar ou traduzi-lo. No entanto, versões alteradas do documento devem ser anotadas com as alterações efetuadas e o nome do autor (de forma que meu nome não fique associado ao texto que não escrevi). —G.W.