Datei- und Metadaten-Sicherheit in Firebird

Geoff Worboys, Telesis Computing
14. Februar 2005
Übersetzt ins Deutsche von Daniel Albuschat
am 02. Januar 2009

Wenn Sie über diese Seite gestolpert sind und nicht wissen, was Firebird ist, können Sie unter diesem Link mehr über Firebird erfahren: www.firebirdsql.org

Dieser Artikel beschreibt die Sicherheit von Firebird Datenbank-Dateien und, im Speziellen, den Zugriff auf die Metadaten, die in diesen Dateien gespeichert sind. Er wurde als Antwort auf die häufig gestellten Fragen in der Firebird Mailing-Liste bezüglich dieses Problems verfasst. Der Artikel vermeidet technische Einzelheiten. Um herauszufinden, wie Sie Dateien in einem bestimmten Betriebssystem absichern können, ziehen Sie bitte die Dokumentation für dieses Betriebssystem zu Rate. Um herauszufinden, wie Zugriffsschutz für Benutzer und Objekte in Firebird realisiert werden kann, ziehen Sie bitte die Dokumentation auf der Firebird-Webseite (siehe Link weiter oben) zu Rate oder legen Sie sich das Buch “The Firebird Book” von Helen Borrie zu.

Inhalt

Hintergrund

Damit eine Anwendung (ein Benutzer) auf eine Firebird-Datenbank zugreifen kann, muss sie sich mit dem Firebird Server-Prozess verbinden. Erhält der Server eine Verbindungsanfrage, vergleicht er die übermittelten Anmeldedaten (Benutzername und Passwort) mit den Daten, die in der Security-Datenbank hinterlegt sind, um den Benutzer zu authentifizieren. Wenn die Authentifizierung erfolgreich war, gewährt der Server der Anwendung Zugriff auf jede Datenbank und benutzt dann die Privilegien und Rollen innerhalb dieser Datenbank, um kleinmaschigere

Kontrolle über Objekte in dieser Datenbank zu ermöglichen.

Der Benutzer, der sich mit der Datenbank verbindet, benötigt niemals direkten Zugriff auf die Datenbank-Datei selbst. Jede Art von Zugriff läuft über den Server-Prozess, der frei nach Bedarf (auf Datei-Ebene) auf die Datenbank-Datei zugreift, um Anfragen zu bearbeiten. Es ist die Aufgabe des Servers, Zugriffe je nach hinterlegten Berechtigungen für den angemeldeten Benutzer zu begrenzen oder zuzulassen.

Hinweis: Die "Embedded"-Variante des Firebird-Servers arbeitet anders und ist für eine abgesicherte Installation nicht geeignet. Die Tatsache, dass es den Embedded Server gibt, ändert nichts an den Sicherheits-Problemen, um die sich dieser Artikel dreht. Es wird durch den Embedded Server lediglich die Wichtigkeit verdeutlicht, eine sichere Umgebung für zu schützende Datenbanken zu schaffen. Der Embedded Server wird später im Detail behandelt.

Jede Installation eines Firebird-Servers hat einen 'SYSDBA'-Benutzer. Dieser Benutzer hat unbegrenzten Zugriff auf alle Datenbanken, die auf diesem Server zur Verfügung stehen. Datenbank-spezifische Privilegien werden ignoriert, wenn auf die Datenbank mit dem Benutzer SYSDBA zugegriffen wird. Wenn eine Datenbank auf einen Server kopiert wird, kann der SYSDBA benutzt werden, um Privilegien an die Benutzer und Anforderungen des Servers anzupassen. Dies bedeutet aber gleichzeitig, dass eine Datenbank durch direkten Zugriff auf die Datenbank-Datei von einem Server, auf dem das SYSDBA-Passwort nicht bekannt ist, auf einen anderen Server kopiert werden kann. Wenn auf diesem Server das SYSDBA-Passwort bekannt ist, erhält man unbegrenzten Zugriff auf die Datenbank.

Dieser Umstand verdeutlicht, dass die Sicherheit von Firebird auf der Annahme basiert, dass der Firebird Server-Prozess in einer angemessen gesicherten Umgebung läuft. Firebird selbst trifft keine Vorkehrungen, um externe Sicherheit zu gewährleisten. Hat eine Person erst einmal physikalischen Zugriff auf eine Datenbank-Datei, gibt es keinen effektiven Weg, diesen Benutzer daran zu hindern, alle Daten (oder Metadaten) aus dieser Datenbank zu lesen.

Hinweis: "Angemessene" Sicherheit hängt von den genauen Anforderungen ab. Dies wird sich von Installation zu Installation stark unterscheiden.

Dies bedeutet, dass jede Installation sicherstellen muss, dass die Datenbank-Dateien diesen Anforderungen entsprechend geschützt sind, um eine angemessene Absicherung zu gewährleisten. In den meisten Fällen bedeutet dies, dass der Firebird Server-Prozess als sein eigener Benutzer laufen sollte und nur dieser Benutzer (und gegebenenfalls der Administrator/root-Benutzer) direkten Zugriff auf Ebene des Betriebssystems auf die Datenbank-Dateien hat. Die Datenbank-Dateien sollten nicht in einem Verzeichnis liegen, auf das aus dem Netzwerk heraus oder von anderen lokalen Benutzern zugegriffen werden kann. Zusätzlich wird der Server-Computer an einem sicheren Ort aufgestellt, um den direkten Zugriff auf die Server-Hardware, ohne die Einschränkungen des Betriebssystems, zu verhindern.

Die oben beschriebene Lösung hilft Entwicklern trotzdem nicht weiter, die Datenbanken zur Distribution und Installation in fremden Umgebungen entwickelt haben und wünschen, ihr geistiges Eigentum, das in dieser Datenbank enthalten ist, zu schützen. Diese Bedenken betreffen vor allem die Metadaten (Tabellenstrukturen und Zusammenhänge, Stored Procedures, Trigger), können aber auch spezielle Daten, die in Tabellen enthalten sind, beinhalten. Diese Bedenken stellen den Kern dieses Artikels dar.

Das Problem

Ein Entwickler erstellt eine Datenbank (und für gewöhnlich eine zugehörige Client-Anwendung) zur Installation auf Servern in fremden Umgebungen. In solchen Umgebungen ist es normal, dass eine Person auf den Computer, auf dem der Firebird-Server betrieben wird, vollen Zugriff hat – zum Beispiel um Datensicherungen und Wartungsarbeiten durchführen zu können. Wie im Abschnitt "Hintergrund" erklärt wird, ermöglicht direkter Zugriff auf die Datenbank-Datei auch vollen und unbegrenzen Zugriff auf alle Inhalte der Datenbank – sowohl Daten als auch Struktur.

In diesen Fällen ist dem Entwickler nicht garantiert, dass der Benutzer sein geistiges Eigentum, das in der Datenbank enthalten ist, vertraulich behandelt. Es ist zu befürchten, dass der Benutzer die Datenbank durch Reverse Engineering zu eigenen Zwecken ausnutzt oder dass die Sicherheit seiner Umgebung nicht ausreicht, um unerlaubten Zugriff auf die Datenbank zu verhindern.

Dies führt zu häufig gestellten Fragen auf der Firebird Mailing-Liste, in etwa:

> "Ich möchte mein Datenbank-Design (Tabellen, Strukturen, Stored Procedures, Triggers, usw.) vor Zugriff von allen Benutzern, die diese auf dem eigenen Arbeitsplatz installieren, schützen. Wie kann ich das mit Firebird machen?"

oder

> "Ich möchte alle Benutzer mit Installationen in der eigenen Umgebung davon abhalten, die Daten in diesen bestimmten Tabellen einzusehen. Sie enthalten proprietäre Informationen, die intern von der Anwendung benutzt werden."

Firebird (zumindest bis Version 2.1) bietet keine eingebauten Verschlüsselungs-Funktionen. Die simple Antwort auf beide dieser Fragen lautet, dass dies mit den derzeitigen Möglichkeiten von Firebird nicht möglich ist. Ein Benutzer, der direkten Zugriff auf die Datei erlangt, kann alle Details und Informationen in dieser Datei einsehen.

In erster Instanz ist keine Notlösung möglich, da der Server selbst die Metadaten lesen können muss. In zweiter Instanz wäre es möglich, Client-seitige Ver- und Entschlüsselung zu implementieren, aber dann verliert man die Möglichkeit der effektiven Nutzung von Datenbank-Indexen und Suchmöglichkeiten – und die Verwaltung von Schlüsseln bleibt ein großes Problem (mehr dazu weiter unten).

Die Lösung

Es gibt tatsächlich nur eine mögliche Lösung für dieses Problem: Die Datenbank und den Server selbst in einer eigenen Umgebung bereitzustellen und Clients nur remote zugreifen zu lassen. Dial-up, Internet, Terminal Server (Windows oder Linux/Unix) oder Ähnliches bieten Möglichkeiten, den gewünschten Anforderungen zu entsprechen.

Auf diese Art wird die Kontrolle über die Datenbank-Dateien behalten und der Zugriff auf die verschiedenen Teile und Strukturen der Datenbanken kann durch die normalen internen Sicherheits-Features von Firebird kontrolliert werden.

Es ist erwähnenswert, dass selbst in dieser Situation Schwierigkeiten auftreten, wenn auch die Struktur der Datenbank geschützt werden soll.

Viele Datenbank-Entwicklungsbibliotheken durchsuchen Metadaten wie Primärschlüssel, Domains

und ähnliche strukturelle Informationen, um die Entwicklung von Client-Anwendungen zu vereinfachen. Dementsprechend wird sich gegebenenfalls herausstellen, dass der Zugriff auf die Metadaten nicht verhindert werden kann, ohne die Anwendung daran zu hindern, Informationen zu sammeln, die es zur korrekten Funktionsweise benötigt. Dies bedeutet, dass man sich entscheiden muss, Metadaten aus der Datenbank auslesbar zu halten oder eine speziellere Schnittstelle für den Zugriff auf die Datenbank zu benutzen, mit der es unter Umständen erheblich mehr Zeit kostet, eine Anwendung zu entwickeln.

Ein weiteres Problem ist, dass viele Client-Anwendungen zwangsläufig strukturelle Information über die Datenbank, mit der sie interagieren, freigeben. Es ist sehr selten, dass eine Datenbank-Anwendung Details über die interne Tabellen-Struktur in der Benutzerschnittstelle nicht Preis gibt. Manche Details wurden gegebenenfalls hinter Views und Stored Procedures versteckt, aber solche Umwege zu implementieren, um strukturelle Informationen zu verstecken, ist eine frustrierende Aufgabe. Es ist möglicherweise sowieso vergebens, da manche Details immer einsehbar sein werden, was auch immer versucht wird.

Benutzerdaten schützen

Bevor wir weiter von der Verschlüsselung der Firebird-Daten reden, möchte ich anmerken, dass es möglich ist, Datenbank-Dateien durch Verschlüsselung zu schützen. Dies hilft Entwicklern nicht dabei, Informationen vor der Extrahierung von legitimen Benutzern zu schützen, aber es kann dabei helfen, die Anforderungen von Kunden an die Sicherheit ihrer Datenbank zu erfüllen.

In manchen Büroumgebungen mag es praktikabel sein, den Firebird Server-Computer an einem sicheren Ort zu platzieren. Während der Arbeitszeiten ist die Wahrscheinlichkeit, dass jemand auf den Compter zugreifen kann, um eine Kopie der Datenbank-Dateien zu erstellen (oder den kompletten Computer oder die Festplatte zu stehlen), sehr gering. Außerhalb der Arbeitszeiten (nachts und an Wochenenden) mag dies allerdings anders sein. Jemand könnte sich Zutritt zum Büro verschaffen und die Festplatte entfernen (oder den kompletten Computer mitnehmen), um Zugriff auf die Datenbank zu erhalten.

Während Firebird selbst keine eingebauten Verschlüsselungs-Features bietet, gibt es einige exzellente Produkte, die dies können. Man könnte Software installieren, um eine verschlüsselte Partition auf dem Computer zu erstellen, um dort die Datenbank-Dateien (und andere vertrauliche Daten) abzulegen. Wenn der Computer heruntergefahren wird, existieren die Daten nur noch in einer verschlüsselten Datei und können ohne das Passwort nicht gelesen werden. Wenn der Computer dann gestartet wird, muss die verschlüsselte Partition mit Hilfe des Passworts eingehangen werden, um auf die Daten zuzugreifen. Dieser zusätzliche und notwendigerweise manuelle Schritt im Boot-Vorgang mag unpraktisch sein, bietet aber als einzige Möglichkeit optimale Sicherheit für unbeaufsichtigte Computersysteme.

Software mit diesen Möglichkeiten sind beispielsweise:”TrueCrypt” (www.truecrypt.org), “BestCrypt” von Jetico (www.jetico.com) und ”PGPDisk” (www.pgpi.org/products/pgpdisk/ - *dieser Link führt zu einer alten Freeware-Version. Die Seite hat jedoch Links zu neuen, kommerziellen Versionen dieses Produkts*). Es gibt weitere, doch dies sind die Produkte, die ich bereits selbst eingesetzt habe.

Wieso bietet Firebird keine Verschlüsselung?

Auf Grund der genannten Bedürfnisse wird häufig gewünscht, dass Firebird in einer zukünftigen Version ausgewählte Benutzerdaten, Metadaten oder gar die komplette Datenbank verschlüsseln können soll. Da ich kein Firebird Kern-Entwickler bin, kann ich nicht kategorisch behaupten, dass dies nicht geschehen wird. Die Frage ist jedoch nicht, ob Verschlüsselung praktisch oder nützlich ist, sondern ob die Verwaltung des Schlüssels/Passworts eine Lösung für das Problem, das wir behandelt haben, bieten kann.

Verschlüsselung kann nur so gut sein wie der geheime Schlüssel, der für die Entschlüsselung benötigt wird. Sie kann schlechter sein, aber niemals besser. Es sind mehrere sehr gute Verschlüsselungs-Algorithmen verfügbar, die genutzt werden könnten. Wenn gute Verschlüsselung benutzt wird, ist es sehr wahrscheinlich, dass Angriffe eher auf den Schlüssel als auf die Verschlüsselung selbst abzielen.

Also schauen wir uns einmal an, wie die Dinge funktionieren könnten, wenn Firebird die Metadaten der Datenbank verschlüsseln würde...

Bevor auf die Datenbank zugegriffen werden kann, müsste der geheime Schlüssel übermittelt werden. Dem Benutzer das Entschlüsselungs-Passwort zu geben wäre nutzlos, da es uns zurück zum Ursprungsproblem führt, dass dieser Benutzer nicht auf die zu versteckenden Informationen zugreifen können soll. Also nehmen wir an, dass der Kunde bei jedem Neustart des Servers den Entwickler anruft, der sich dann einwählt und das benötigte Passwort eingibt. Selbst wenn dies praktikabel wäre, löst es nicht zwangsläufig das Problem. Zum Beispiel: Der Kunde könnte Mittschnittprogramme installieren, um das Passwort zu ermitteln, wenn es eingegeben wird.

Es gibt Hardware-basierende Lösungen, um ein Passwort für die Entschlüsselung bereitzustellen. Aber dieses Gerät müsste ebenfalls in Besitz des Kunden sein und wir können nicht verhindern, dass dieser es benutzt, um auf die Datenbank von einem Server aus zuzugreifen, auf dem das SYSDBA-Passwort bekannt ist.

Firebird ist ein Open-Source Produkt. Wenn Verschlüsselungs-Möglichkeiten eingebaut wären oder Open-Source Bibliotheken benutzt würden, wäre es für einen Nutzer möglich, eine eigene Version des Firebird-Servers oder der Bibliothek zu erstellen. Dieser könnte nicht nur die benötigte Ver- und Entschlüsselung vornehmen sondern auch das Passwort oder die entschlüsselten Informationen direkt ausgeben. Der Entwickler kann dies, da er nicht die Kontrolle über den verwendeten Server hat, weder erkennen noch verhindern.

Man könnte darüber nachdenken, eine eigene Version des Firebird-Servers zu erstellen, der das Entschlüsselungs-Passwort direkt in die ausführbare Datei eingebaut hat. Es existieren jedoch "Decompiler", mit denen es nicht lange dauern würde, das Passwort einfach durch den Vergleich der angepassten und der originalen Version herauszufinden.

Es gibt verschiedene Datenbank-Produkte, die vorgeben, sie würden sichere Verschlüsselung bieten. Vielleicht mag die Verschlüsselung selbst sicher sein, aber solange die Handhabung des Passworts nicht sicher ist, wird dies nicht zum gewünschten Ziel einer sicheren Datenbank führen. Der Entwickler wird in den Glauben versetzt, die Daten seien geschützt. Aber solange die Passwortübermittlung nicht überprüft wurde, kann nicht genau festgestellt werden, ob dies auch der Fall ist.

Die schmerzhafte Wahrheit ist, dass in dem Moment, in dem man die Kontrolle über die Hardware verliert, auf der die Ver- und Entschlüsselung stattfindet, alle Wetten gegen einen gehen. Wenn das

Entschlüsselungspasswort nicht ausreichend gesichert untergebracht werden kann, wird selbst sichere Verschlüsselung zu “Security by Obscurity” (Sicherheit durch Verschleierung).

Die Verbreitung von Daten verhindern

Manche fordern die Verschlüsselung von Daten in der Datenbank, um die Verbreitung von Daten zu verhindern. Sie sind damit einverstanden, dass ein bestimmter, autorisierter Benutzer die Daten sehen kann, aber sie möchten diesen Benutzer davon abhalten, die Daten an andere Personen weiterzugeben.

Stellen wir uns einmal vor, dass alle oben genannten Probleme mit der Passwortverwaltung gelöst seien, so dass es für den Benutzer unmöglich wird, einfach die Datenbank zu kopieren. In diesen Fällen könnte man einfach ein kleines Programm schreiben, das die Daten, an denen man interessiert ist, von dem offiziell installierten Server kopiert und in eine eigene Datei oder Datenbank schreibt.

Ich denke, dass es möglich ist, dass Firebird in Zukunft eine Art von Anwendungs-Authentifizierung bereitstellt, um diese Form der Daten-Extrahierung zu verhindern. Aber bei dieser Authentifizierung bestehen die gleichen Probleme wie bei der Verschlüsselung. Wenn keine Kontrolle über den Server besteht, kann nicht verhindert werden, dass der Benutzer sich seine eigene Version vom Server installiert, der keine Authentifizierung benötigt.

SYSDBA-Zugriff entfernen

Mehrfach wurde vorgeschlagen, dass es eine mögliche Lösung sei, den SYSDBA-Zugriff auf eine Datenbank zu entfernen. Die Idee dahinter ist, dass auf eine Datenbank, wenn sie zu einem neuen Server verschoben wird, nicht mit dem bekannten SYSDBA-Passwort zugegriffen werden kann. Manche haben bereits eingeschränkten Erfolg vermelden können, indem sie eine SQL-Rolle namens SYSDBA erstellt und sichergestellt haben, dass diese keinen Zugriff auf die Datenbank-Objekte hat.

Dies löst allerdings das Problem nicht wirklich. Die Datenbank-Datei kann beispielsweise mit einem Hex-Editor oder ähnlichem Werkzeug geöffnet werden, um eine Liste der möglichen Benutzer herauszufinden (den Besitzer der Datenbank-Objekte herauszufinden wäre besonders nützlich). Ist dies einmal bekannt, können diese Benutzer zu dem neuen Server hinzugefügt und direkt benutzt werden. Ein noch einfacherer Workaround wäre die Nutzung des Firebird Embedded Servers (siehe unten), oder sich eine eigene Version des Firebird-Servers zu compilieren, die diese Sicherheitseinschränkungen ignoriert.

Es wurde auch der Vorschlag vorgetragen, es zu ermöglichen, den SYSDBA-Benutzer umzubenennen. Dies bietet begrenzten Schutz gegen Brute-Force Attacken über ein Netzwerk, um das SYSDBA-Passwort herauszufinden. Solche Angriffe würden dann nicht nur das Passwort, sondern auch den Benutzernamen erraten müssen. Es hilft allerdings nicht weiter, wenn eine Person direkten Zugriff auf die Datenbank-Datei hat.

Den Quelltext von Stored Procedures und Triggern löschen

Wenn eine Stored Procedure oder ein Trigger für eine Firebird-Datenbank geschrieben und definiert wird, speichert der Server eine beinahe komplette Kopie des Quelltexts zusammen mit einer "compilierten" Version, die BLR (Binary Language Representation) genannt wird. Die BLR wird vom Server ausgeführt, der Quelltext bleibt jedoch ungenutzt.

Manche Entwickler versuchen wenigstens einen Teil der Datenbank-Metadaten zu schützen, indem sie den Quelltext dieser Stored Procedures und Trigger löschen, bevor sie die Datenbank ausliefern (durch ein simples update auf die entsprechenden Tabellen, in denen die Metadaten gespeichert sind). Ich empfehle, dies nicht zu machen, aus zwei Gründen...

BLR ist eine ziemlich simple Kodierung des Quelltextes. Es wäre nicht schwierig, die BLR zurück in menschenlesbare Form zu bringen. Diese Dekodierung wäre ohne Kommentare und Formatierung, aber das SQL, das in einer Stored Procedure oder einem Trigger benutzt wird, ist selten so kompliziert, dass das Fehlen von Kommentaren irgendwelche Probleme für den Lesenden verursachen würde. Deshalb ist der Schutz durch die Entfernung der Quelltexte nicht sonderlich effektiv.

Zweitens kann der Quelltext zu anderen Zwecken nützlich sein. Er erlaubt, Korrekturen direkt in einer Datenbank vorzunehmen, ohne den kompletten Datenbank-Quelltext bei sich zu haben (und sich daran zu erinnern, den Quelltext der Stored Procedure wieder aus der Kundendatenbank zu entfernen, nachdem die Korrektur eingespielt wurde). Der Quelltext wird auch von verschiedenen Werkzeugen wie meiner eigenen DBak-Anwendung – eine Alternative zum Backup-Programm "gbak" - genutzt. Ich habe mich noch nicht daran gesetzt, meinen eigenen BLR-Decoder zu programmieren. DBak verlässt sich auf die Verfügbarkeit vom Quelltext, um DDL-Skripte zu erstellen, die die Datenbank rekonstruieren.

Embedded Firebird-Server

Es gibt eine spezielle Version des Firebird-Servers, die als "Embedded" bezeichnet wird. Es handelt sich dabei um eine spezielle Client-Bibliothek, die den Server selbst beinhaltet. Wenn eine Anwendung diese Biblitohek benutzt, ist sie selbst der Server und erlaubt damit direkten Zugriff auf jede Datenbank, die auf dem lokalen Computer verfügbar ist. Diese Version des Servers benutzt nicht die Security-Datenbank. Der Benutzername, der während des "Logins" (es erfolgt keine Passwortüberprüfung) angegeben wird, wird lediglich für die Verwaltung des Zugriffs auf Datenbank-Objekte (per SQL-Zugriffsrechte) benutzt. Wenn dieser Benutzername aber SYSDBA oder der des Datenbankbesitzers ist, ist uneingeschränkter Zugriff möglich. Der Embedded Server ist für Entwickler nützlich, die eine einfach auszuliefernde Single-User-Anwendung schreiben, die keine Benutzersicherheit benötigt.

In dieser kurzen Einführung erscheint es so, als stelle ein installierter Embedded Client auf einem Server, der auch andere Datenbanken bereitstellt, ein großes Sicherheitsproblem dar. In Wirklichkeit ist die Gefahr nicht größer, als gäbe es gar keinen Embedded Client.

Wenn eine Anwendung den Embedded Server lädt, operiert dieser Server im Sicherheitskontext der Anwendung (also des Benutzers, als der die Anwendung gestartet ist). Dies bedeutet, dass der Embedded Server nur Zugriff auf Datenbank-Dateien hat, auf die der Benutzer auch direkt über das Betriebssystem zugreifen könnte. Nicht vertrauenswürdigen Benutzern uneingeschränkte Rechte zur Installation von Programmen auf einem sicheren Server zu geben ist in jedem Fall eine schlechte Idee. Da die Datei-Zugriffsrechte für die Datenbanken aber sowieso entsprechend vergeben sein sollten, ist der Embedded Server keine Gefahr mehr. Die Gefahr durch den Embedded Server ist

gleichwertig zu allen anderen Dingen, die der Nutzer installieren kann.

Die Tatsache, dass es den Embedded Server gibt, verdeutlicht, was alles möglich ist, wenn man direkten Zugriff auf die Datenbank-Dateien hat. Dies gilt besonders im Open-Source-Umfeld. Wenn es den Embedded Server nicht bereits gäbe, wäre es sicherlich für jemanden möglich, eine solche Version aus dem original Firebird-Quelltext zu erstellen.

Andere Formen der Verschleierung

Es wurden weitere mögliche Formen von Security by Obscurity vorgeschlagen, beispielsweise spezielle Ereignisse, die beim Login und Logout ausgelöst werden, um Zugriff zu gewähren oder zu verhindern. Solche Features bieten selbst für Closed-Source-Systeme nur eingeschränkten Nutzen, in denen die fehlende Kenntnis über die Programmierung dieser Mechanismen die Informationen schützt. Bei einem Open-Source-System ist es ein Leichtes, diese Mechanismen auszuhebeln, indem man sich eine eigene Version compiliert, die diese einfach nicht ausführt. Es ist unmöglich, in einem Open-Source-System etwas zu verschleiern.

Zu betrachten ist außerdem, was passiert, wenn compilierte, ausführbare Dateien herausgegeben werden. Compilierte Programme sind großartige Beispiele der Verschleierung. Es wird (normalerweise) keine Verschlüsselung benutzt und alle ausgeführten Schritte des Programms können von jemandem mit dem nötigen Wissen und der nötigen Zeit genauestens analysiert werden. Es gibt sogar “Decompiler”, die bei dieser Aufgabe unterstützen können. Stellt eine solche Person erst einmal fest, mit Hilfe welcher Programmbibliothek die Anwendung erstellt wurde, wird es noch einfacher, nur den “geheimen” Code zu isolieren, was die Analyse erheblich beschleunigen würde. Haben Sie an Borland, Microsoft - oder welchen Hersteller auch immer - jemals geschrieben, dass sie compilierte Programme verschlüsseln sollen?

Akzeptabel niedrige Sicherheit

Meine bisherigen Anmerkungen waren bezüglich starker Sicherheit (“stark” bedeutet, dass es sich um ein Verfahren handelt, bei dem der Schlüssel zwingend notwendig ist, um die Daten zu entschlüsseln) und ich denke, dass das Konzept von Security by Obscurity mit angemessener Verachtung beschrieben wurde. Andererseits ist schwache Sicherheit manchmal ausreichend. Manchmal sind die Daten einfach nicht so wertvoll. Es soll lediglich vor dem gelegentlichen Fummler versteckt und für den fortgeschrittenen Daten-Dieb lästig gestaltet werden.

Ich habe solche Systeme selbst an verscheidenen Stellen benutzt. Manchmal gibt es einfach keinen Grund, Twofish, AES oder sonstwas in jeder Situation anzuwenden, weil diese Algorithmen für starke Sicherheit gedacht sind. Sie machen es dem Entwickler schwer und sind kompliziert zu benutzen. Eine simple bitweise Verschlüsselung mit einem festgelegten Schlüssel mag ausreichen. Wenn der Schlüssel vom Dieb ermittelt werden kann, macht es keinen Unterschied, ob es sich um einen schwachen oder einen starken Verschlüsselungsalgorithmus handelt. Das Spiel ist sowieso zu Ende.

Hinweis: Die meisten XOR-basierenden, bitweisen Verschlüsselungen können mit sehr wenig Aufwand geknackt werden. Schlagen Sie in einer guten Verschlüsselungs-Referenz nach, um mehr Informationen darüber zu erhalten und weitere Optionen kennen zu lernen.

Der Knackpunkt bei Security by Obscurity ist, dass es “obskur” (also nicht leicht herauszufinden) sein muss! Wenn Firebird eine Art von Verschlüsselungsalgorithmus in die Daten-Lese- und -Schreiboperationen einbauen würde, wären diese nicht obskur, da es sich um ein Open-Source-Projekt handelt. Es würde kaum Zeit kosten, den Quelltext zu analysieren, um den verwendeten Schlüssel herauszufinden. Und dann ist alles verloren.

Wenn dieses Feature aber nun wirklich benötigt wird, ist die einzige Möglichkeit, den Quelltext von Firebird zu nehmen, eigenen obskuren Verschlüsselungs-Code einzufügen und eine eigene Variation des Firebird-Servers zu compilieren. (Solcher Code könnte durch Reverse Engineering des Programms gefunden werden, aber es bedürfte schon eines ernsthaft interessierten Diebs). Bevor dies jedoch getan wird, muss überprüft werden, ob es nicht reichen würde, diesen speziell compilierten Firebird-Server zusammen mit der Datenbank-Datei zu kopieren, um wieder Zugriff auf die Daten über den Firebird-Server zu erlangen.

Die philosophische Betrachtungsweise

Es stellt sich auch die philosophische Frage, warum ein Open-Source-System genutzt wird, um eine Closed-Source-Datenbank zu erstellen. Viele Menschen haben dem Projekt in dem Glauben beigetragen, dass Open Source der beste Weg sei, Software bereitzustellen.

Generell bin ich der Meinung, dass, wenn es um das Speichern von Benutzerdaten geht, es dem Benutzer immer möglich sein sollte, auf seine eigenen Daten zuzugreifen – wofür er häufig die Strukturen und Prozesse verstehen muss, die genutzt wurden, um diese Daten zu erstellen und abzulegen. Wenn ein Unternehmen den Betrieb einstellt oder anderweitig nicht mehr erreichbar ist, ist es unter Umständen von extremer Wichtigkeit, dass der Benutzer wenigstens seine eigenen Daten in angebrachter Form extrahieren kann, um auf alternative Systeme umzusteigen.

Kann den Nutzern zugetraut werden, geistiges Eigentum zu respektieren während das Unternehmen noch betriebstätig und erreichbar ist? Wenn die nötigen Dienste vom Unternehmen angeboten werden, werden sie dies wahrscheinlich. Wenn nicht, gibt es wenig, was man tun kann, um sie daran zu hindern.

Schlussfolgerung

Das Problem ist, dass zu viele Leute Sicherheit nicht verstehen und wie schwierig es ist, richtige Sicherheit zu gewährleisten. Leider gibt es viele Softwareprodukte, die dieses Missverständnis bestärken, indem sie Verschleierung statt wirkliche Sicherheit einbauen. Betrachte man nur die Anzahl der Unternehmen, die “Datenwiederherstellungs”-Dienste betreiben. Diese müssen die sogenannte Sicherheit der Produkte durchbrechen, um auf die Informationen hinter den verfälschten Daten Zugriff zu erlangen.

Verschlüsselung ist kein Allheilmittel, um Sicherheit zu erlangen. Wenn man nicht die Kontrolle über die Umgebung (die Hardware, das Betriebssystem und jede darauf laufende Software) hat, hat man keine Kontrolle über die Sicherheit – egal, welche Verschlüsselungsmethoden eingesetzt werden. Und genau das trifft zu, wenn eine Datenbank auf fremde Server-Installationen übertragen wird.

Wenn man wirklich Daten oder Metadaten sicher speichern möchte, darf man die Kontrolle über die

Datenbank-Dateien und die Umgebung nicht abtreten. Keine andere Lösung kann eine Sicherheit auf diesem Level gewährleisten.

Dokumentenhistorie

11. April 2005: Das Kapitel “Akzeptierbar niedrige Sicherheit” wurde überarbeitet und versucht, klar zu stellen, dass XOR-Algorithmen schwach sind und der Anwender weitere Informationen einholen muss, wenn er sich weiter mit diesem Thema beschäftigen möchte.

26. April 2005: Zusätzliches Kapitel zum Embedded Server (und Verweise darauf) wurden hinzugefügt. Fußnote wurde in einen kursiv geschriebenen Hinweis gewandelt, weil Fußnoten nicht in HTML funktionieren. Inhaltsverzeichnis hinzugefügt.

04-December-2005: Verweis auf TrueCrypt hinzugefügt. Kapitel “Nutzung dieses Dokuments” hinzugefügt. Kapitel “Danksagungen” hinzugefügt.

25. Oktober 2005: Kleinere Änderungen, um Klarheit zu verbessern.

Danksagungen

Ich möchte den verschiedenen Leuten danken, die das Dokument durchgesehen und Kommentare dazu abgegeben haben. Ich möchte auch den vielen Leuten danken, die der Firebird Support Mailing-Liste beitragen, die die Quelle vieler Informationen ist, die in diesem Artikel genannt werden.

Nutzung dieses Dokuments