Universidade Federal do Rio de Janeiro
Escola Politécnica / COPPE

Controle Descentralizado do Fluxo de Informações na Camada de Aplicação

Implementação de um protótipo para o ambiente automotivo

Daniel Vega Simões

Poli / COPPE

Março de 2013

Universidade Federal do Rio de Janeiro
Escola Politécnica / COPPE

Controle Descentralizado do Fluxo de Informações na Camada de Aplicação

Implementação de um protótipo para o ambiente automotivo

Autor:

Daniel Vega Simões

Orientador:

Prof. Henrique Cukierman, D.Sc.

Examinador:

Prof. Aloysio Pedroza, D.Sc.

Examinador:

Prof. Ricardo Marroquim, D.Sc.

Poli / COPPE

Março de 2013

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica – Departamento de Eletrônica e de Computação
Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária.
Rio de Janeiro – RJ – CEP 21949-900

Simões, Daniel Vega

Controle Descentralizado do Fluxo de Informações na Camada de Aplicação / Daniel Vega Simões – Rio de Janeiro: UFRJ/POLI-COPPE, 2013

VIII, 22 p.: il.; 29,7 cm.

Orientador: Henrique Cukierman

Projeto (graduação) – UFRJ/POLI/Departamento de Eletrônica e de Computação – COPPE, 2013.

Referências Bibliográficas: p. 19-21.

1. Controle de Fluxo de Informações. 2. DIFC. 3. Dispositivos Eletrônicos. 4. Ambiente Automotivo I. Cukierman, Henrique. II. Universidade Federal do Rio de Janeiro, Poli/COPPE. III. Controle Descentralizado do Fluxo de Informações na Camada de Aplicação.

AGRADECIMENTO

Agradeço aos professores Yves Roudier, Aloysio Pedroza, Henrique Cukierman e Ricardo Marroquim pela participação na validação desse projeto junto à UFRJ. Agradeço também aos amigos de graduação, Hugo Eiji, Pedro Pisa e Diogo Ferrazani, entre tantos outros, por momentos marcantes dentro e fora das salas de aula.

Gostaria de mostrar minha gratidão aos departamentos de intercâmbio e relações internacionais da UFRJ devido à oportunidade e o acompanhamento ao realizar um duplo diploma com uma universidade estrangeira. A experiência permitiu um crescimento profissional e pessoal que carregarei para o resto da vida.

Agradeço finalmente a minha mãe, meu pai, meus irmãos e outros membros da família pelo apoio e incentivo durante os anos de graduação no Brasil e durante o programa de mobilidade acadêmica no exterior.

RESUMO

CONTROLE DESCENTRALIZADO DO FLUXO DE INFORMAÇÕES NA CAMADA DE APLICAÇÃO

Daniel Vega Simões

Março/2013

Orientador: Henrique Cukierman
Curso: Engenharia de Computação e Informação

Os automóveis e seus sistemas embarcados evoluíram de forma significativa nas últimas décadas e oferecem hoje em dia uma vasta gama de opções de integração a seus usuários, particularmente com seus dispositivos eletrônicos. No entanto, os fabricantes de automóveis precisam lidar com os problemas de segurança que surgem a partir dessa integração e que podem causar vazamento de dados privados. Os sistemas automotivos embarcados atuais contêm vários métodos de controle de acesso usados para aumentar os aspectos da segurança da comunicação, mas eles não consideram a propagação dos dados entre os componentes da rede interna e os aparelhos eletrônicos integrados.

Esse trabalho aborda o problema de proteção de dados sensíveis em um ambiente automotivo aplicando as noções de Controle Descentralizado do Fluxo de Informações (*Decentralized Information Flow Control* – DIFC). Essas noções são usadas em um protótipo baseado em um cenário realista e na nova arquitetura IP do automóvel. O protótipo serve como uma prova de conceito e é avaliado tanto do ponto de vista do desempenho como da segurança. Esse trabalho permite aos fabricantes de automóveis considerarem as noções do DIFC para a integração futura entre os dispositivos eletrônicos e os sistemas automotivos embarcados.

Palavras-Chave: Controle Descentralizado do Fluxo de Informações, DIFC, Sistemas Embarcados, Dispositivos Eletrônicos, CE-Devices.

ABSTRACT

DECENTRALIZED INFORMATION FLOW CONTROL AT APPLICATION LEVEL

Daniel Vega Simões

March/2013

Advisor: Henrique Cukierman
Course: Computer and Information Engineering

Vehicles and their embedded systems have evolved significantly in the last decades and offer users a wide range of integration options nowadays, specially with their CE-Devices. However, vehicle manufacturers have to cope with the security issues that arise from this integration and that can lead to private information leakage. Current automotive embedded systems include several access control methods to enhance the security aspects of the communication, but do not consider the propagation of the information between the components of the internal network and the integrated CE-Devices.

This work tackles the issue of protecting sensitive data in an automotive environment by applying the concepts of Decentralized Information Flow Control (DIFC). These concepts are used in a prototype based on a realistic scenario and on the new vehicle IP-based architecture. The prototype serves as a proof of concept and is evaluated from both the performance and the security points of view. This work allows vehicle manufacturers to consider DIFC concepts for future integration of CE-Devices and vehicle embedded systems.

Keywords: Decentralized Information Flow Control, Embedded Systems, CE-Devices.

PREFÁCIO

Este trabalho representa o Projeto Final de Graduação do aluno Daniel Vega Simões no contexto do curso de Engenharia de Computação e Informação da Universidade Federal do Rio de Janeiro - UFRJ. O Projeto Final foi realizado junto à empresa BMW Forschung und Technik, em Munique, na Alemanha, tendo em vista os benefícios e as regras do programa de Duplo Diploma Acadêmico entre a Universidade Federal do Rio de Janeiro e a École Nationale de Télécommunications - Télécom ParisTech.

Esse trabalho descreve as motivações, a pesquisa bibliográfica, o trabalho realizado e os resultados obtidos de forma sucinta e serve como referência ao trabalho completo, que deve ser anexado de forma definitiva no Anexo A.

SIGLAS

CAN Controller Area Network (Rede da área de controle)

CE-Device Consumer Electronic Device (Dispositivo Eletrônico de Consumo)

DIFC Decentralized Information Flow Control (Controle Descentralizado do Fluxo de Informações)

DL-Manager Driving Log Manager (Controlador dos registros de direção)

ECU Electronic Control Unit (Unidade de Controle Eletrônica)

GPS Global Positioning System (Sistema de Posicionamento Global)

IFC Information Flow Control (Controle do Fluxo de Informações)

IP Internet Protocol (Protocolo da Internet)

LIN Local Interconnect Network (Rede de interconexão local)

MOST Media Oriented System Transport (Transporte do sistema orientado a mídias)

SEIS Security in Embedded IP-based Systems (Segurança em Sistemas Embarcados baseados no IP)

SQL Structured Query Language (Linguagem estruturada de interrogativas)

SSL Secure Sockets Layer (Camada de soquetes segura)

TCP Transmission Control Protocol (Protocolo de controle de transmissão)

TLS Transport Layer Security (Segurança da camada de transporte)

VM Virtual Machine (Máquina virtual)

Sumário

Introdução

1.1. Contextualização

Os automóveis e os sistemas que os compõem mostraram uma mudança rápida de paradigma nas últimas décadas. Após o surgimento dos carros no início do século XX, compostos principalmente por partes puramente mecânicas, o paradigma se manteve estável por várias décadas. No entanto, com a evolução da engenharia eletrônica e de telecomunicações, os veículos rapidamente adaptaram seus sistemas internos, de forma a melhorar a acurácia e o tempo de resposta dos elementos mecânicos. Hoje em dia, um único veículo pode possuir mais de 70 unidades de controle eletrônicas (*Electronic Control Units* – ECUs), que atuam como controladores e medidores do desempenho e da segurança oferecida ao motorista e aos passageiros.

Os sistemas de comunicação internos a um veículo evoluíram de forma rápida, com foco em desempenho, segurança e robustez. Exemplos desses sistemas são LIN, CAN, MOST e FlexRay. No entanto, eles não proveem segurança na comunicação e ataques são possíveis, como mostrados em [5, 12, 22]. Esses ataques podem ser extremamente perigosos em vários aspectos, como a segurança da informação e privacidade dos usuários ou até mesmo segurança física destes.

Ao mesmo tempo, novas tecnologias estão sendo inseridas no ambiente automotivo, integrando o sistema veicular a uma gama de outros dispositivos de comunicação. Essa integração, caso feita de forma irresponsável, pode acarretar em prejuízos econômicos e risco de violação da segurança ou privacidade.

1.2. Objetivo

Tendo em vista os aspectos apresentados anteriormente, o Projeto SEIS (*Sicherheit in Engebetteten IP-basierten Systemen* - Segurança em Sistemas Embarcados baseados no IP) [9] tem por objetivo remodelar e propor uma nova arquitetura nos sistemas embarcados, de forma a acatar as rápidas mudanças e integrá-los à nova era de comunicação sem perder o foco da segurança de dados, informações,

privacidade e comunicação. Esse projeto foi iniciado em 2009 e conta com diversos grupos das indústrias automobilística e de eletrônicos alemães, além de laboratórios e universidades.

A principal característica do projeto é a utilização do protocolo IP (*Internet Protocol*), já usado mundialmente, dentro dos ambientes automotivos, implantando-o tanto para comunicações internas, entre ECUs, e externas, com redes como 3G e LTE. Com isso, o projeto propõe um sistema homogêneo e compatível com as tecnologias atuais, simplificando o modelo do sistema embarcado, pois não há mais a necessidades de tradutores de comunicação externa e interna ao sistema embarcado. No entanto, a implantação do IP traz consigo os desafios de segurança, que se potencializam ao interagir com componentes utilizados para manter a segurança dos motoristas e passageiros. Nesse cenário, tanto a proposta quanto a implementação do novo sistema automobilístico deve ser idealizada com as premissas da segurança em sistemas de comunicação.

A integração segura dos dispositivos eletrônicos (*Consumer Electronic Devices* - CE-Devices) é uma tarefa difícil, pois deve levar em conta a heterogeneidade desses dispositivos, assim como suas restrições. Além disso, o ciclo de vida de um dispositivo eletrônico é significativamente menor do que o de um veículo, caracterizando um cenário no qual um sistema automotivo deve se conectar com vários dispositivos de vários usuários.

1.3. Contribuição

A principal contribuição desse trabalho para o projeto SEIS e a comunidade acadêmica de segurança em sistemas embarcados é uma implementação para a integração segura entre dispositivos eletrônicos e sistemas automotivos. O foco dessa implementação é a proteção de dados confidenciais através do Controle Descentralizado do Fluxo de Informações (*Descentralized Information Flow Control* - DIFC) trocadas entre as partes, utilizando os conceitos publicados na comunidade acadêmica. Um caso de uso específico é utilizado para demonstrar a importância do controle do fluxo de informações e extrair os requisitos de segurança associados. A proposta é implementada e, finalmente, avaliada na prática em termos de desempenho e segurança.

1.3.1. **Caso de Uso**

O caso de uso especificado nesse trabalho trata de um componente localizado no interior do veículo e responsável por controlar o armazenamento, os acessos, as modificações e a remoção dos dados de acordo com as políticas de controle de fluxo de informações especificadas.

Armazenamento O armazenamento consiste em persistir informações do veículo, coletadas enquanto um motorista se utiliza do carro. Essas informações são coletadas, propagadas através da rede interna e armazenadas num componente de armazenamento persistente, assim como informações para controle de acesso e origem.

Acesso O acesso a informações armazenadas no veículo depende de quem está requisitando o acesso e qual tipo de informação está sendo requisitada. Para avaliar se a requisição é válida e se o requisitante tem o direito de acessar a informação, é necessário estabelecer um componente capaz de filtrar e rotular requisições externas (Proxy) e utilizar esses rótulos como comparação para as informações associadas ao dado requisitado. Existem dois tipos de controle de acesso, a saber: o direito à informação em si e o direito de propagar essa informação para um ambiente externo e inseguro.

Modificação A modificação de informações armazenadas no veículo se caracteriza como um acesso, uma manipulação e um novo armazenamento. Por esse motivo, as premissas de modificação seguem as de acesso e armazenamento. Um detalhe importante é que a informação modificada não necessariamente possui as mesmas propriedades da informação original.

Remoção A remoção de uma informação armazenada no veículo depende estritamente do acesso e modificação de suas propriedades. Mais especificamente, é necessário que o requisitante possua o direito de modificar essa informação a ponto de removê-la.

Um caso de uso que possui essas quatro propriedades é o caso de uma empresa de aluguel de carros, que oferece veículos equipados com esse sistema. Para ilustrar a integração com dispositivos eletrônicos, o sistema oferece acesso tanto para dispositivos da empresa, para avaliação e cobrança, quanto para o motorista, para controle e acompanhamento. Enquanto um motorista utiliza o veículo, dados do carro provenientes das ECUs e associados a esse motorista são armazenados no componente de

armazenamento persistente com os respectivos rótulos. Dados privados do motorista, como GPS, são acessíveis apenas pelo motorista, enquanto que dados do veículo, como nível de óleo e quilometragem, são acessados pela empresa. O sistema deve ser capaz de diferenciar quem está requisitando qual informação, de forma a controlar o acesso.

Ao final do período de aluguel, a empresa pode oferecer um desconto baseado no tipo de pavimento em que o carro rodou: mais barato para asfalto, mais caro para estradas de terra. Para isso, é necessário que tanto a localização (privada ao motorista) e os preços (privados à empresa) sejam utilizados no cálculo. Esse é um caso de desconfiança mútua e ambas as partes podem delegar a responsabilidade ao veículo, que tem acesso às informações, mas não precisa divulgá-las. O resultado final, no entanto, pode ser divulgado, pois não contém nem informações da localização do motorista nem dos preços vigentes e diferenciados. No entanto, o resultado final ainda pode ser restringido para leitura apenas pelo motorista em questão e pela empresa, impossibilitando uma futura releitura por outro motorista.

Por fim, a empresa pode querer reiniciar o veículo ao seu estado original. Caso ela tenha permissão de deleção de todos os dados, ela pode assim fazê-lo, sem, no entanto, ter acesso de leitura.

Esse caso de uso serve como ilustração para os requisitos de confidencialidade dos dados trocados entre duas partes que não depositam confiança entre si. Esse trabalho trata principalmente desses casos de confidencialidade, assim como alguns casos de integridade dos dados armazenados.

1.4. Estrutura do documento

O resto do documento está estruturado da seguinte forma: no capítulo 2, apresentamos uma pesquisa bibliográfica sobre o tema Controle do Fluxo de Informações; no capítulo 3, mostramos sucintamente a implementação realizada e a aplicação dos conceitos; no capítulo 4, avaliamos a implementação do ponto de vista do desempenho e da segurança; e, finalmente, no capítulo 5, apresentamos uma conclusão e uma sequência de possíveis trabalhos futuros.

Controle do Fluxo de Informações

Nesta seção, fazemos uma análise bibliográfica sucinta no tópico principal desse trabalho, Controle do Fluxo de Informações, avaliando os diferentes tipos e definindo os conceitos básicos para as outras seções. O estudo bibliográfico completo se encontra no trabalho original, em inglês, anexado ao final desse documento.

2.1. Introdução

A proteção da privacidade dos dados é um tópico de intensa pesquisa devido à quantidade crescente de dados sendo transmitidos pela rede e manipulados por entidades não confiáveis. Enquanto que métodos de segurança como *firewalls* e criptografia previnem que os dados sejam acessados de forma não autorizada, eles não garantem que esses dados não sejam propagados de forma indevida depois de acessados. Por exemplo, a criptografia permite que dados sejam trocados por meio de um canal não seguro, mas não garante a confidencialidade desses dados após decriptografados no lado receptor.

O Controle do Fluxo de Informações (*Information Flow Control* - IFC) ataca esse problema analisando o fluxo das informações dentro do sistema, atribuindo níveis de segurança a dados e entidades que os manipulam. O modelo básico consiste em dois níveis, Baixo (B) e Alto (A), que representam, respectivamente, a informação disponível publicamente e a informação secreta. Tanto a confidencialidade quanto a integridade são garantidas a partir do controle do fluxo de informações entre um nível e outro. No caso da confidencialidade, o dado não pode seguir o fluxo de A para B, ou seja, uma informação secreta não pode se tornar publicamente disponível a partir de manipulação ou sistemas de computação. De forma inversa, a integridade do dado é mantida desde que não siga o fluxo de B para A. O conceito global é que dados só podem seguir fluxos em que fiquem mais restritos, formando um modelo de comparações entre dados e entidades [6].

No entanto, devido ao fluxo sempre mais restritivo, é necessário definir privilégios, chamados de desclassificação e endosso, para permitir que dados sejam divulgados e se tornem úteis [15, 17].

2.2. Tipos de IFC

O Controle de Fluxo de Informações é um conceito antigo, pesquisado e refinado desde os anos 70, e possui diversos tipos diferentes.

2.2.1 IFC estático e dinâmico

O IFC estático é responsável por determinar e analisar todos os possíveis fluxos de informação durante a compilação de um programa. Para isso, é necessário estabelecer diretrizes de programação e anotações, além de um compilador especial, capaz de determinar os possíveis fluxos, analisar as violações das políticas de confidencialidade e integridade e produzir um resultado para o programador. Exemplos como JFlow [16] implementam o IFC estático e garantem o controle do fluxo de informações, apesar de necessitarem do código fonte *a priori*, o que caracteriza um cenário não realístico.

Por outro lado, o IFC dinâmico acontece durante a execução do programa, aplicando rótulos às estruturas de dados e canais de entrada e saída do sistema. Podendo ser aplicados nos níveis de aplicação, sistemas operacionais ou ambos, os sistemas que implementam o IFC dinâmico garantem um controle mais flexível e adaptado aos cenários reais de desconfiança mútua, além da possibilidade de externalizar a programação do aplicativo, mas sofrem no desempenho e podem acarretar em problemas sérios em sistemas direcionados ao desempenho. Exemplos de IFC dinâmico são encontrados nos sistemas Asbestos [7], HiStar [24], Flume [13] e Laminar [20].

2.2.2 IFC concentrado e distribuído

Um IFC concentrado acontece quando os fluxos que devem ser controlados são entre usuários ou processos de uma única máquina física, sem o tráfego por canais de comunicação. Apesar de ainda existirem, esses sistemas não são mais predominantes e não são o foco desse trabalho. Um exemplo de IFC concentrado é TaintDroid [8].

Já o IFC distribuído acontece, como o próprio nome já sugere, em um sistema distribuído, que depende da comunicação em um canal externo, frequentemente inseguro. Apesar de diversos mecanismos já protegerem os dados enquanto estes estão

em trânsito, ou seja, no canal de comunicação, o IFC distribuído visa proteger os dados após eles terem sido recebidos com sucesso, como no DStar [25].

2.2.3 IFC centralizado e descentralizado

A distinção entre IFC centralizado e descentralizado depende unicamente de como as políticas do controle de fluxo estão distribuídas. Caso estejam centralizadas em uma única entidade, que é reconhecida e aceita como autoridade por todas as outras entidades, denominamos esse sistema de IFC centralizado. Esse tipo de IFC ajuda a manter o controle das definições das políticas em um único lugar, facilitando a manutenção ou modificação destas. No entanto, não proveem uma forma de resolver o cenário de desconfiança mútua, pois todas as entidades devem depositar a confiança na autoridade central.

O modelo descentralizado (*Decentralized Information Flow Control* - DIFC) é caracterizado pela definição local das políticas de controle de fluxo de informações. Nesse cenário, o cumprimento das restrições de fluxo é feito por cada entidade localmente, independente das outras entidades. Apesar de mais complexo, o DIFC se mostra mais genérico e realista, principalmente nos cenários de desconfiança mútua. No entanto, todas as entidades precisam concordar nos mecanismos e nos rótulos a serem utilizados para que o sistema funcione corretamente.

Os exemplos anteriores Asbestos [7], TaintDroid [8] e JFlow [16] se enquadram no modelo centralizado, enquanto que Flume [13], Jif [18], Laminar [20], HiStar [24] e DStar [25] correspondem ao modelo descentralizado.

2.3 Conceitos

Neste capítulo, são definidos os conceitos relevantes para as outras seções, tendo em vista a revisão bibliográfica apresentada na seção anterior. A implementação se enquadra em um sistema IFC descentralizado e dinâmico no nível de aplicação, implantados em um ambiente automotivo simulado composto por várias entidades. O sistema operacional, o *middleware* utilizado e a parte da aplicação responsável pelo controle de fluxo são considerados confiáveis. O objetivo é controlar a propagação da informação pelo sistema se utilizando de rótulos nos dados e nas entidades que os manipulam, à semelhança do projeto DEFCon [15].

Rótulo (L)	
Conjunto de Confidencialidade (S)	Conjunto de Integridade (I)
car, user	*ecu*

Tabela 1. Exemplo de um rótulo.

O primeiro conceito importante é o conceito de rótulos de segurança, que são aplicados aos dados de acordo com as entidades que os manipulam. Como mostrados na Tabela 1, os rótulos são divididos em um conjunto de confidencialidade (S) e um conjunto de integridade (I). Cada conjunto é composto de etiquetas, que representam uma política de confidencialidade (em S) ou uma política de integridade (em I). As etiquetas de confidencialidade determinam quem pode acessar o dado rotulado, enquanto que as etiquetas de integridade identificam quem produziu ou modificou o dado e são, portanto, responsáveis pela integridade do dado. Rótulos são propagados pelo sistema e todas as entidades participantes devem ser capazes de entendê-los e utilizá-los para controlar o fluxo de informações.

O segundo conceito importante é o conceito da execução do controle de fluxo de informações, ou seja, as primitivas matemáticas que regem o controle. Definimos por $\subseteq$ a relação "pode haver fluxo para". Dessa forma, o conjunto de confidencialidade do dado deve ser um subconjunto do conjunto de confidencialidade da entidade, ou $S_{dado} \subseteq S_{entidade}$, para que possa haver o fluxo desse dado para essa entidade. Isso garante que um dado com um conjunto mais restritivo de confidencialidade, ou mais secreto, não possa fluir para uma entidade que não possua no mínimo o mesmo conjunto de confidencialidade.

Já a integridade funciona de forma oposta. O conjunto de integridade do dado precisa ser um superconjunto do conjunto de integridade da entidade, ou $I_{dado} \supseteq I_{entidade}$, para que possa haver o fluxo desse dado para essa entidade. Isso garante que o nível de integridade do dado precisa ser no mínimo igual ao da entidade para a qual o dado está indo.

Ambas as propriedades caracterizam o modelo Bell-LaPadula de controle de fluxo de informações [2] e podem ser resumidas abaixo:

$$L_{dado} \preccurlyeq L_{entidade}$$

$$\textit{se e somente se}$$

$$S_{dado} \subseteq S_{entidade} \; e \; I_{dado} \supseteq I_{entidade},$$

$$\textit{onde } L_{dado} = \{S_{dado}, I_{dado}\} \; e \; L_{entidade} = \{S_{entidade}, I_{entidade}\}$$

Com esse modelo, uma vez que uma etiqueta de confidencialidade tenha sido inserida em um dado, não haverá fluxo desse dado para nenhuma entidade que não possua no mínimo o mesmo nível de confidencialidade, ou seja, as etiquetas de confidencialidade persistem, a menos que um privilégio seja exercido. Por outro lado, etiquetas de integridade são frágeis e são destruídas com qualquer manipulação sobre o dado rotulado.

Os últimos conceitos aqui apresentados são os privilégios de desclassificação e endosso. O privilégio de desclassificação permite remover uma etiqueta de confidencialidade e, portanto, reduzir o nível de confidencialidade do dado rotulado. Isso habilita fluxos a entidades que não possuíam o mesmo nível de confidencialidade. Da mesma forma, o privilégio de endosso permite incluir uma nova etiqueta de integridade, ou seja, garantir a integridade de um determinado dado rotulado. Ambos os privilégios reduzem o nível de segurança do rótulo e, consequentemente, do dado rotulado e devem ser aplicados apenas por entidades que possuem autoridade para tal.

Implementação

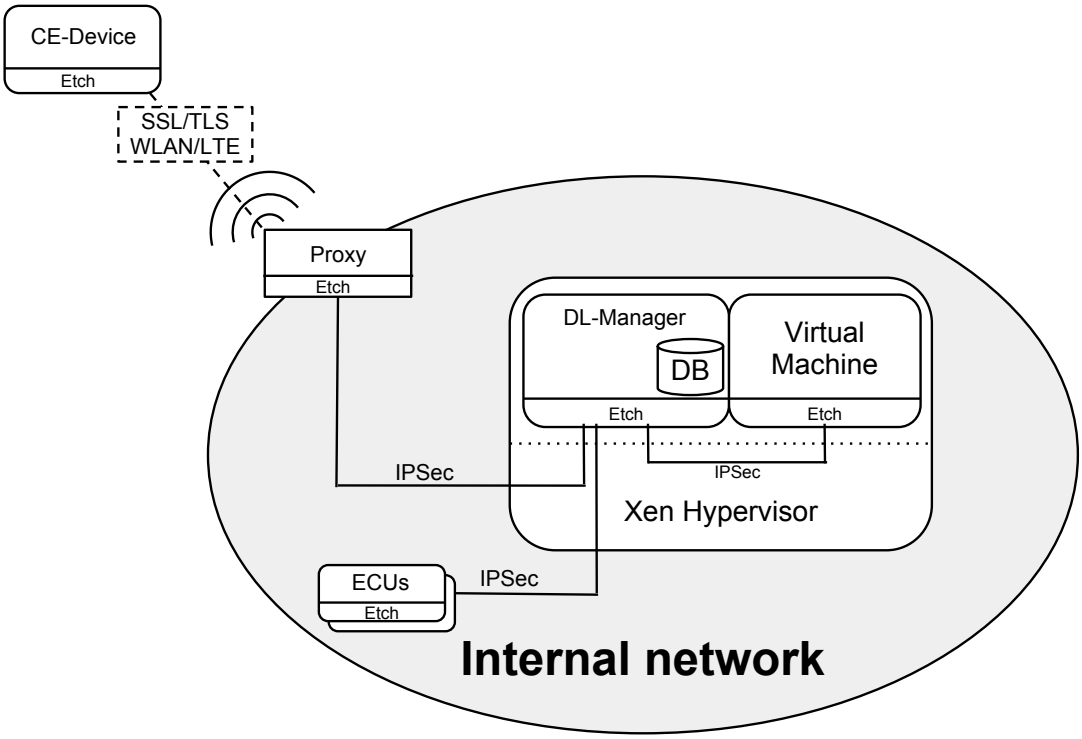

Figura 1. Arquitetura geral do protótipo.

A arquitetura do sistema implementado pode ser vista na Figura 1, extraída do trabalho original, em inglês. Ela é dividida entre a região interna ao veículo e a região externa, ambas conectadas unicamente por um componente de tradução e controle, chamado Proxy. A seguir, detalhamos cada um dos componentes representados:

DL-Manager Esse componente é o servidor e recebe requisições de todas as outras entidades. É o único componente com acesso ao banco de dados, no qual dados rotulados estão armazenados de forma persistente. Em nosso protótipo, esse componente também é responsável pela execução do controle de fluxo de informações.

Banco de dados O banco de dados é o único componente que armazena dados de forma persistente. Contém apenas dados rotulados e só recebe requisições e comandos a partir do DL-Manager.

Unidade de Controle Eletrônica (ECU) O componente ECU representa todas as unidades de controle eletrônicas que existem dentro de um veículo hoje em dia. Ele produz dados relativos ao veículo e associados aos motoristas.

Máquina Virtual (VM) A máquina virtual executa aplicativos não confiáveis em um ambiente protegido, como detalhado em [14]. Isso permite que entidades externas executem programas na rede interna para manipulação de dados que não podem ser propagados para a rede externa ao veículo.

Dispositivo Eletrônico (CE-Device) O dispositivo eletrônico externo ao veículo é considerado não confiável e se comunica com a rede interna apenas através do componente Proxy. Suas requisições são monitoradas e controladas de acordo com o fluxo de dados.

Todos os componentes são executados em cima do *middleware* Etch [1], que inicialmente foi lançado pela Apache Foundation como um arcabouço de serviços de redes multiplataforma e independente da linguagem e do protocolo de transporte, mas foi customizado pela BMW para as suas necessidades [3]. Esse *middleware* é responsável por padronizar a comunicação e os serviços de rede, permitindo que o programador foque nas funcionalidades. No nosso protótipo, Etch executa com Java e todos os serviços de rede foram definidos e customizados para o nosso caso de uso específico.

Já o componente Proxy [4] se impõe como um separador de domínios: rede interna do veículo e rede externa ao veículo. Foi desenvolvido inteiramente por colegas da BMW Forschung und Technik e atua como um mediador da comunicação entre o carro e o mundo exterior, monitorando todos os pacotes, em ambas as direções, e determinando seu nível de confiança e segurança, além da origem, através de certificados SSL/TLS. No nosso caso, o Proxy é responsável por determinar a origem da requisição e o nível de segurança e rotular a entidade externa. Esse rótulo é então propagado para a rede interna ao veículo, junto com a requisição, permitindo a decisão de processar ou não a requisição por parte do DL-Manager. No caminho contrário, o DL-Manager pode determinar um nível mínimo de segurança para um determinado dado ser propagado e o Proxy pode filtrar os pacotes que seguiriam para a rede externa de acordo com as informações coletadas do cliente conectado.

A implementação segue o caso de uso mencionado anteriormente da empresa de aluguel de carros e foi inteiramente programada em Java. O fluxo principal do sistema é apresentado a seguir:

1º Passo A empresa aluga um carro para um novo motorista/usuário. Esse carro possui o sistema DIFC implementado, assim como o banco de dados e o Proxy.

2º Passo O motorista, ao utilizar o veículo, associa seu dispositivo eletrônico e o sistema determina esse usuário como o motorista atual.

3º Passo À medida que o veículo é utilizado, a(s) ECU(s) produz(em) dados sobre o carro regularmente e envia(m) ao DL-Manager, que é responsável por armazenar de forma persistente no banco de dados, junto com o rótulo que melhor se adequa a esses dados, de acordo com a política de controle de fluxo de informações. No nosso caso, alguns dados são privados ao motorista, alguns ao carro (e, consequentemente, à empresa) e alguns a ambos.

4º Passo Ainda em posse do veículo, o motorista pode querer acessar os dados armazenados relativos a si. O DL-Manager é responsável por tratar essa requisição e retornar apenas os dados acessíveis pelo motorista, executando o controle de fluxo de informações de acordo com as políticas previamente estabelecidas.

5º Passo Após o fim do período de aluguel, tanto o motorista quanto a empresa podem querer acessar as informações armazenadas. O DL-Manager, novamente, deve distinguir e garantir que apenas dados relativos ao requerente são retornados.

6º Passo Para calcular o preço final do aluguel, a empresa gostaria de oferecer um preço diferenciado caso o carro tenha rodado em estradas de asfalto ou de terra. Para isso, é necessário acessar informações de localização, privadas ao motorista, e informações de preço, privadas à empresa. O cálculo, portanto, deve ser realizado dentro do veículo, para que nenhuma informação chegue à rede externa. Um aplicativo é executado dentro do ambiente protegido (Máquina Virtual) e calcula o preço final. Esse preço final pode, após o DL-Manager exercer privilégios de desclassificação, ser divulgado tanto para o motorista quanto para a empresa.

7º Passo Esse passo é opcional e ilustra a possibilidade da empresa apagar todos os dados armazenados, de todos os motoristas, sem ter acesso a eles. Essa remoção acontece dentro do veículo e não é propagada à rede externa.

Apesar de existirem diversos fluxos alternativos, o fluxo principal oferece uma boa ilustração dos requisitos de segurança e privacidade do sistema. Após um estudo e avaliação do modelo do banco de dados a ser utilizado, que podem ser vistos na versão integral do trabalho, em inglês, assim como o código SQL utilizado para a construção do banco, explicitamos a seguir a execução do controle de fluxo de informação em detalhes.

3.1. Armazenamento

O armazenamento apenas acontece quando um motorista está conectado ao DL-Manager através do componente Proxy. O dispositivo eletrônico do usuário oferece um certificado que prove sua identidade, do qual o Proxy extrai as informações de origem e a segurança da conexão, rotulando a entidade externa com um nível de segurança. Esse rótulo é passado para a rede interna, junto com a requisição ao DL-Manager, que confere a existência de tal usuário e suas informações.

Enquanto o veículo é utilizado, diversas ECUs produzem diversos tipos de dados, que são enviados, através da rede interna, ao DL-Manager. Este, por sua vez, determina de acordo com as políticas de confidencialidade o rótulo de cada tipo de dado, antes de armazená-lo no banco de dados. Dados privados ao motorista são armazenados com rótulo $L_{motorista} = \{motorista.private;\ ecu\}$, dados privados à empresa com rótulo $L_{carro} = \{empresa.private;\ ecu\}$ e dados que pertencem a ambos são armazenados com rótulo $L_{ambos} = \{motorista.public, empresa.public;\ ecu\}$.

3.2. Acesso

Ao tentar acessar um dado, o sistema DIFC, através do DL-Manager, tem um papel crucial na manutenção da confidencialidade dos dados. Como explicado anteriormente, um acesso externo possui um rótulo estabelecido pelo Proxy que

identifica o requerente externo e, com isso, o DL-Manager pode determinar se o dado em questão pode fluir para o requerente ou não, de acordo com:

$$L_{dado} \preccurlyeq L_{cliente}, ou$$
$$S_{dado} \subseteq S_{cliente}.$$

Nesse trabalho, o foco foi dado em confidencialidade e a integridade foi deixada como trabalho futuro. O algoritmo completo pode ser visto no trabalho original em anexo, em inglês.

3.3. Manipulação

Considerando o caso no qual dados privados não podem fluir para a rede externa, como no caso do cálculo do preço final diferenciado na empresa de aluguel de carros, estabelecemos um ambiente seguro em cima do hipervisor Xen [23] e cada máquina virtual é rotulada pelo DL-Manager de acordo com os dados que manipula. Dessa forma, uma máquina virtual é criada com o intuito de manipular dados específicos e não tem acesso a outros dados, pois seu rótulo é definitivo e mantido no DL-Manager. A reutilização de máquina virtual não é permitida para prevenir vazamento de dados.

Havendo uma requisição de manipulação, o DL-Manager é responsável por encontrar a máquina virtual associada aos rótulos dos dados que devem ser manipulados. Essa máquina virtual, por sua vez, executa um programa desenvolvido externamente, que acessa os dados de forma parecida com a explicada na subseção anterior, mas baseada nos rótulos da máquina virtual e não nos rótulos do Proxy. Ao final da execução, a máquina virtual retorna um resultado, que possui o mesmo rótulo. O DL-Manager é responsável por analisar esse resultado e, não violando as políticas de privacidade, desclassificá-lo, antes de enviá-lo para a rede externa.

3.4. Remoção

A remoção de dados acontece de forma similar ao acesso dos dados, mas sem o retorno ao requerente original. O DL-Manager deve verificar que o requerente possui nível de segurança suficiente para acessar e remover os dados antes de fazê-lo.

Avaliação

A implementação foi avaliada de acordo com o desempenho e a segurança. Neste capítulo, apresentamos o ambiente de avaliação e os principais resultados. A avaliação completa pode ser vista no trabalho original em anexo.

O ambiente de avaliação era composto por computadores 2x Intel Core 2 Duo e sistema operacional Linux/Ubuntu 10.04.1 ou Linux/Fedora 16. O banco de dados foi implementado utilizando o MySQL 5.1.41 [19] e as configurações de rede incluíam uma rede cabeada Ethernet Gigabit entre todos os computadores. Na camada de rede, o IP foi executado entre o Dispositivo Eletrônico e o Proxy, enquanto que a comunicação interna foi executada em cima do IPSec [10]. Por fim, a comunicação na rede externa ao veículo também possuía uma camada SSL/TLS com autenticação bilateral.

A avaliação de desempenho foi realizada através de requisições do Dispositivo Eletrônico para a Máquina Virtual, passando pelo Proxy e pelo DL-Manager. Note que essa avaliação não incluiu nenhum acesso ao banco de dados nem aplicação do controle de fluxo de informações. Os resultados dessa avaliação, em comparação com a avaliação realizada em [14], são:

	Tempo (nanoseg.)	Vazão (req./seg.)
Conexão insegura	20,262	98,707
SSL/TLS + IPSec	96,569	20,711

Tabela 2. Resultados de desempenho geral.

Esse resultado mostra que o desempenho é bem inferior, devido principalmente à conexão SSL/TLS. Por esse motivo, a avaliação do desempenho do sistema DIFC foi realizada sem SSL/TLS para permitir a percepção da diferença entre um sistema sem controle de fluxo de informações e um sistema que executa esse controle.

A avaliação do sistema DIFC foi realizada através de requisições na rede interna do veículo ao DL-Manager, que acessa o banco de dados para coletar os dados requisitados, aplica o controle de fluxo de informações de acordo com as políticas de

confidencialidade para determinar se os dados podem fluir para o requerente e retorna, ou não, esses dados. Mostramos no gráfico da Figura 2 as médias de 12.000 requisições.

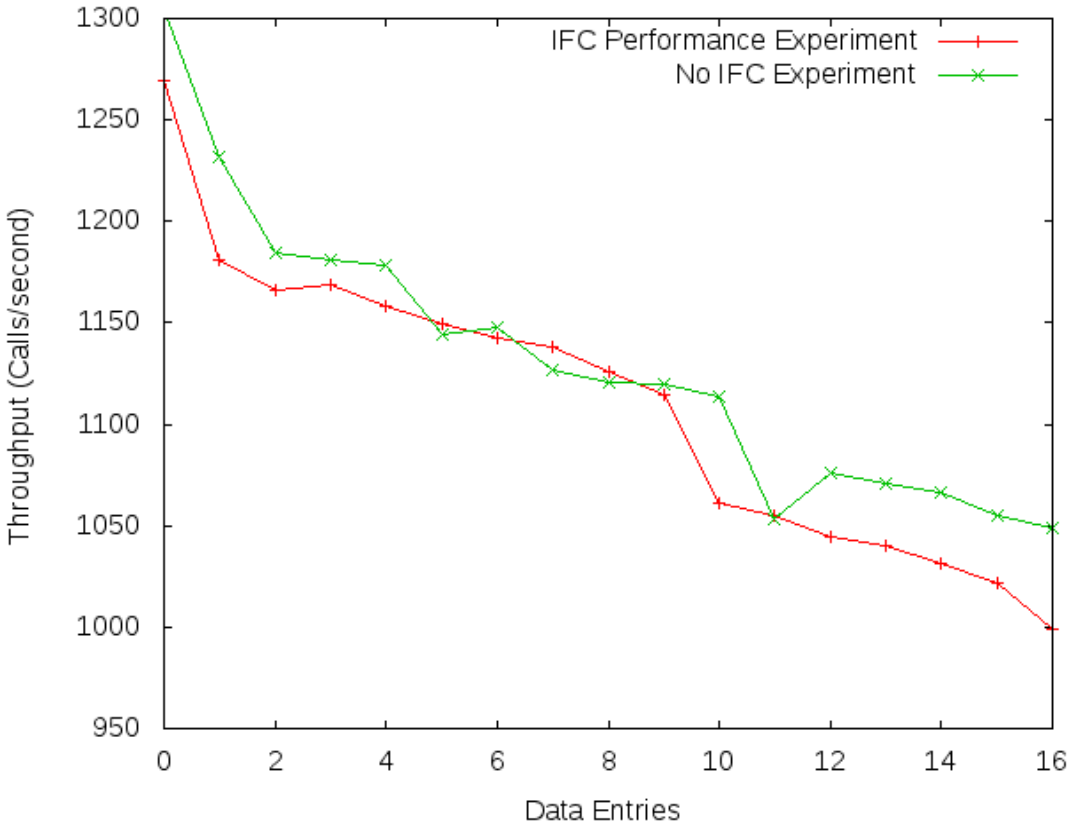

Figura 2. Comparação de desempenho do DIFC.

Esse resultado, depois de tratado e normalizado em retas do tipo $f(x) = B + Ax$, mostra que a inserção do sistema DIFC reduziu o desempenho em algo entre 1,16% e 2,68%. No entanto, esse resultado tende a piorar com o aumento de resultados do banco de dados. Isso provavelmente se deve ao algoritmo de acesso ao banco de dados.

Em termos de segurança, o protótipo obteve sucesso ao garantir a confidencialidade de acordo com os requisitos mencionados no início desse trabalho. Para ilustrar essa afirmação, consideremos os seguintes casos:

Dispositivo do motorista malicioso No caso de um dispositivo do motorista malicioso que se conecta ao veículo, mas não possui a habilidade de fraudar um certificado SSL/TLS, o componente Proxy o rotula da forma correta e o DL-Manager, ao acessar os dados, impedirá que estes sigam para o requerente malicioso.

Dispositivo da empresa malicioso O dispositivo da empresa possui mais privilégios do que o dispositivo do motorista, como a remoção de todos os dados, mas as requisições são tratadas da mesma forma. O DL-Manager consegue restringir o fluxo de dados rotulados para o requerente malicioso.

Máquina virtual maliciosa A máquina virtual sempre tem um propósito e um rótulo associado. Por esse motivo, uma máquina virtual maliciosa não consegue acessar os dados de outro motorista a não ser aquele designado para essa máquina. Além disso, a máquina virtual se encontra em um ambiente protegido, baseado nos conceitos de virtualização.

É importante ressaltar que os certificados SSL/TLS são cruciais no sistema apresentado. Caso eles não sejam confiáveis ou sejam suscetíveis a erros ou fraudes, todo o sistema DIFC pode estar comprometido.

Conclusão e Trabalhos Futuros

Esse trabalho tratou do problema da proteção de confidencialidade de fluxos de dados em um ambiente automotivo. Para isso, utilizou e aplicou os conceitos de Controle Descentralizado de Fluxo de Informações em um protótipo, que serviu como prova de conceito.

O protótipo simulou um ambiente automotivo composto de rede interna e rede externa, conectadas entre si por um Proxy. Dispositivos eletrônicos localizados na rede externa fizeram requisições à rede interna e um componente, DL-Manager, foi responsável por aplicar o controle de fluxo de dados baseado nos rótulos inseridos pelo Proxy. Os dados ficaram armazenados, junto com os respectivos rótulos, em um banco de dados na rede interna.

Essa prova de conceito oferece uma forma de controlar a integração entre dispositivos eletrônicos e os veículos no futuro. No entanto, o desempenho observado mostra que muito trabalho ainda precisa ser feito para ser implantado num ambiente direcionado ao desempenho, como é a indústria automotiva.

Como trabalhos futuros, oferecemos as seguintes sugestões:

Políticas de integridade Esse sistema DIFC focou principalmente nas políticas de confidencialidade. No entanto, políticas de integridade ofereceriam ainda mais uma fonte de sabedoria na tomada de decisão do sistema DIFC e aumentaria o nível de segurança.

Algoritmos de banco de dados A otimização dos algoritmos de acesso e armazenamento do/ao banco de dados utilizados nesse trabalho foge do escopo, permitindo uma melhoria no futuro tendo em vista a melhora do desempenho global do sistema.

Escala O sistema apresentado não escala para grandes quantidades de dados, devido a restrições do *middleware* utilizado. Isso poderia ser melhorado, permitindo uma serialização de dados mais eficiente.

Implantação em um veículo real Para a compreensão global do sistema e suas propriedades, ele ainda deve ser implantado e testado em um veículo real.

Referências Bibliográficas

[1] Apache Foundation. Apache Incubator Etch, 2012. http://incubator.apache.org/etch/ acessado em 19/Jul/2012.

[2] Bell, D., e La Padula, L. Secure computer system: Unified exposition and multics interpretation. Tech. rep., DTIC Document, 1976.

[3] Bouard, A. Software development for a security middleware in car. Master's thesis, EURECOM, 2010.

[4] Bouard, A., Schanda, J., Herrscher, D., e Eckert, C. Automotive Proxy-based Security Architecture for CE Device Integration. Em *Proceedings of the 5th International Conference on Mobile Wireless Middleware, Operating Systems, and Applications* (2012).

[5] Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F., e Kohno, T. Comprehensive experimental analyses of automotive attack surfaces. *Proceedings of the 2011 Usenix Security* (2011).

[6] Denning, D. A lattice model of secure information flow. *Communications of the ACM 19, 5* (1976), 236-243.

[7] Efstathopoulos, P., Krohn, M., VanDeBogart, S., Frey, C., Ziegler, D., Kohler, E., Mazieres, D., Kaashoek, F., e Morris, R. Labels and event processes in the asbestos operating system. *ACM SIGOPS Operating Systems Review 39, 5* (2005), 17-30.

[8] Enck, W., Gilbert, P., Chun, B., Cox, L., Jung, J., McDaniel, P., e Sheth, A. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. Em *Proceedings of the 9th USENIX conference on Operating systems design and implementation* (2010), USENIX Association, pp. 1-6.

[9] eNOVA. SEIS - Sicherheit in Eingebetteten IP-basierten Systemen, 2012. http://strategiekreis-elektromobilitaet.de/public/projekte/seis/ acessado em 19/Jul/2012.

[10] Gollmann, Dieter. *Computer Security*. John Wiley & Sons, Chichester, 2004. ISBN: 0-471-97844-2.

[11] Gryc, Andy. Making sense of the Smartphone-Vehicle Cacophony, 2011.

[12] Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., *et al.* Experimental security analysis of a modern automobile. Em *Security and Privacy (SP), 2010 IEEE Symposium* (2010), IEEE, pp. 447-462.

[13] Krohn, M., Yip, A., Brodsky, M., Cliffer, N., Kaashoek, M., Kohler, E., e Morris, R. Information flow control for standard os abstractions. Em *ACM SIGOPS Operating Systems Review* (2007), vol. 41, ACM, pp. 321-334.

[14] Mattila, E. Isolation and networking aspects of executing untrusted applications in the automotive environment. Master's thesis, EURECOM, 2012.

[15] Migliavacca, M., Papagiannis, I., Eyers, D., Shand, B., Bacon, J., e Pietzuch, P. Defcon: high-performance event processing with information security. Em *Proceedings of the 2010 USENIX conference on USENIX annual technical conference* (2010), USENIX Association, pp. 1-1.

[16] Myers, A. Jflow: Practical mostly-static information flow control. Em *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (1999), ACM, pp. 228-241.

[17] Myers, A., e Liskov, B. A decentralized model for information flow control. Em *ACM SIGOPS Operating Systems Review* (1997), vol. 31, ACM, pp. 129-142.

[18] Myers, A., e Liskov, B. Protecting privacy using the decentralized label model. *ACM Transactions on Software Engineering and Methodology (TOSEM) 9, 4* (2000), 410-442.

[19] Oracle Corporation. MySQL Relational Database Management System, 2012. http://www.mysql.com/ acessado em 23/Jul/2012.

[20] Roy, I., Porter, D., Bond, M., McKinley, K., e Witchel, E. Laminar: practical fine-grained decentralized information flow control, vol. 44. ACM, 2009.

[21] Weckemann, K., Lim, H., e Herrscher, D. Practical experiences on a communication middleware for ip-based in-car networks. Em *Proceedings of the 5th International Conference on Communication System Software and Middleware* (2011), ACM, p. 12.

[22] Wolf, M., Weimerskirch, A., e Paar, C. Security in automotive bus systems. Em *Workshop on Embedded IT-Security in Cars* (2004), pp. 11-12.

[23] Xen.org. Xen Hypervisor, 2012. http://www.xen.org/ acessado em 25/Jul/2012.

[24] Zeldovich, N., Boyd-Wickizer, S., Kohler, E., e Mazières, D. Making information flow explicit in histar. Em *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation* (2006), pp. 19-19.

[25] Zeldovich, N., Boyd-Wickizer, S., e Mazières, D. Securing distributed systems with information flow control. Em *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation* (2008), USENIX Association, pp. 293-308.

Anexo A

Encontra-se a seguir, em anexo, o projeto original, em inglês, como apresentado e aprovado na obtenção do grau de Ingénieur na École Nationale de Télécommunications / Télécom ParisTech, durante o programa de duplo diploma realizado entre Set/2010 e Set/2012.

Decentralized Information Flow Control at Application Level

A prototype implementation for an automotive environment

CONFIDENTIAL

Daniel Vega Simoes

Master's Thesis
Sophia Antipolis, August 31st, 2012

Academic Supervisor:	Professor Yves Roudier, Ph.D., Telecom ParisTech
Company Supervisor:	Alexandre Bouard, M.Sc., BMW Group
Company:	BMW Forschung und Technik GmbH
Period:	March 1st, 2012 - August 31st, 2012
Location:	Munich, Germany

Abstract

Vehicles and their embedded systems have evolved significantly in the last decades and offer users a wide range of integration options nowadays, specially with their CE-Devices. However, vehicle manufacturers have to cope with the security issues that arise from this integration and that can lead to private information leakage. Current automotive embedded systems include several access control methods to enhance the security aspects of the communication, but do not consider the propagation of the information between the components of the internal network and the integrated CE-Devices.

This thesis tackles the issue of protecting sensitive data in an automotive environment by applying the concepts of Decentralized Information Flow Control (DIFC). These concepts are used in a prototype based on a realistic scenario and on the new vehicle IP-based architecture. The prototype serves as a proof of concept and is evaluated from both the performance and the security points of view. This work allows vehicle manufacturers to consider DIFC concepts for future integration of CE-Devices and vehicle embedded systems.

Résumé

Les voitures et ses systèmes embarqués ont évolué de forme significative dans les dernières années et offrent actuellement une vaste gamme d'options aux usagers, particulièrement avec leurs appareils électroniques. Cependant, les constructeurs automobiles doivent faire face aux problèmes de sécurité qui surviennent à partir de cette intégration et qui peuvent entamer des fuites d'informations privées. Les systèmes automobiles embarqués actuels comprennent plusieurs méthodes de contrôle d'accès pour renforcer les aspects de la sécurité de la communication, mais ils ne considèrent pas de propagation de données entre les composants du réseaux interne et les appareils électroniques integrés.

Cette thèse aborde le problème de protéger les données sensibles dans le milieu automobile en appliquant les notions du Contrôle Décentralisé du Flux d'Informations (*Decentralized Information Flow Control (DIFC)*). Ces notions sont utilisées dans un prototype basé sur un scénario realistique et sur la nouvelle architecture IP de la voiture. Le prototype sert comme une preuve de concept et il est évalué des points de vue de la performance et de la sécurité. Ce travail permet les constructeurs automobiles à considérer les notions du DIFC pour l'intégration future entre les appareils électroniques et les systèmes automobiles embarqués.

Kurzfassung

Fahrzeuge und deren eingebettete Systeme haben sich in den letzten Jahrzehnten signifikant weiterentwickelt und bieten den Nutzern heutzutage eine große Vielfalt an Integrationsmöglichkeiten inbesondere durch den Einsatz elektronischer Endgeräte. Fahrzeughersteller müssen sich nun jedoch Sicherheitsproblemen stellen, die durch diese Integration entstehen und die zum Verlust privater Informationen führen können. Obwohl aktuelle, eingebettete Fahrzeugsysteme zahlreiche Methoden der Zugriffskontrolle zur Verstärkung der Kommunikationssicherheit beinhalten, berücksichtigen sie nicht die Informationsweitergabe zwischen den einzelnen Komponenten des fahrzeuginternen Netzwerks und den integrierten, elektronischen Endgeräten.

Diese Diplomarbeit geht auf die Frage des Schutzes sensibler Daten durch die Anwendung der Konzepte der dezentralen Informationsflusskontrole (*Decentralized Information Flow Control (DIFC)*) ein. Diese Konzepte werden in einem Prototyp, der auf einem realistischen Szenario und der neuen IP Architekture basiert, angewandt. Der Prototyp dient als eine Bestätigung des Konzepts und wird sowohl aus dem Blickwinkel der Performanz als auch der Sicherheit evaluiert. Diese Arbeit erlaubt Automobilherstellern diese DIFC-Konzepte bei der Integration von elektronischen Endgeräten und eingebetteten Fahrzeugsystemen zukünftig zu berücksichtigen.

Resumo

Os automóveis e seus sistemas embarcados evoluíram de forma significativa nas últimas décadas e oferecem hoje em dia uma vasta gama de opções de integração a seus usuários, particularmente com seus aparelhos eletrônicos. No entanto, os fabricantes de automóveis precisam lidar com os problemas de segurança que surgem a partir dessa integração e que podem causar vazamento de dados privados. Os sistemas automobilísticos embarcados atuais contêm vários métodos de controle de acesso usados para aumentar os aspectos da segurança da comunicação, mas eles não consideram a propagação dos dados entre os componentes da rede interna e os aparelhos eletrônicos integrados.

Esta tese aborda o problema de proteção de dados sensíveis em um ambiente automotivo aplicando as noções de Controle Descentralizado do Fluxo de Informações (*Decentralized Information Flow Control (DIFC)*). Essas noções são usadas em um protótipo baseado em um cenário realista e na nova arquitetura IP do automóvel. O protótipo serve como uma prova de conceito e é avaliado tanto do ponto de vista da performance como da segurança. Esse trabalho permite aos fabricantes de automóveis considerar as noções do DIFC para a integração futura entre os aparelhos eletrônicos e os sistemas automobilísticos embarcados.

Acknowledgements

I wish to thank Alexandre Bouard for his guidance and knowledge, as well as Professor Yves Roudier, for his supervision. Furthermore, I'd like to thank my colleagues at BMW Forschung und Technik for their support and for making my internship a great experience. I am also thankful for the help and dedication of Esko Mattila throughout the development of this thesis.

I would like to show my gratitude to my friends, both in France and Germany, who helped me living and working in a foreign country. Finally, I wish to express my appreciation for my family for all their support and encouragement, without which this major step of my life would not have been possible.

Sophia Antipolis, August 31st, 2012

Daniel Vega Simoes

Authenticity declaration

I warrant that the thesis is my original work and that I have not received outside assistance. Only the cited sources have been used in this thesis. Parts that are direct quotes or paraphrases are identified as such.

Sophia Antipolis, August 31st, 2012

Daniel Vega Simoes

Abbreviations and Acronyms

CAN	Controller Area Network
CE-Device	Consumer Electronic Device
DIFC	Decentralized Information Flow Control
DL-Manager	Driving Log Manager
DoS	Denial of Service
ECU	Electronic Control Unit
GPS	Global Positioning System
IFC	Information Flow Control
IP	Internet Protocol
LIN	Local Interconnect Network
MOST	Media Oriented System Transport
OS	Operating System
OSI	Open Systems Interconnection
PC	Personal Computer
SEIS	Security in Embedded IP-based Systems
SQL	Structured Query Language
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
VM	Virtual Machine

Contents

Introduction

Since the beginning of car manufacturing, vehicles and their embedded systems have changed very fast. A few decades ago, a vehicle was composed mainly of mechanical parts, interacting by means of mechanical controllers. However, as electronic systems evolved, they were quickly adopted by vehicle manufacturers, increasing accuracy and response time of the mechanical parts. Nowadays, vehicles can contain over 70 electronic control units (ECUs) providing performance and safety to their drivers, as shown in Figure 1.1.

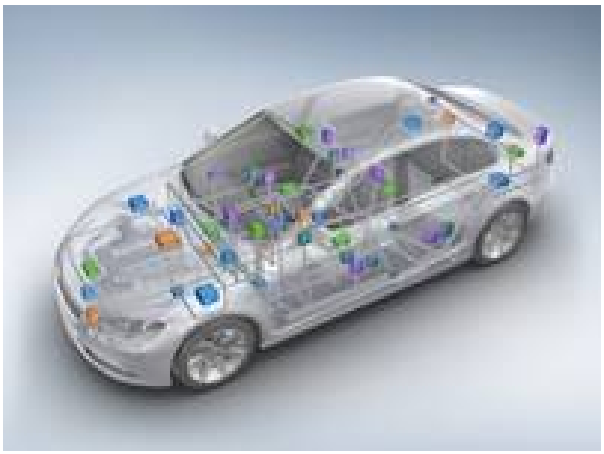

Figure 1.1: ECUs in a modern vehicle.

Throughout the years, a complex communication system was developed in order to allow information exchange between ECUs and it is composed of several networks and buses, such as LIN, CAN, MOST and FlexRay. The current automotive embedded system is focused mainly on performance, safety and robustness regarding the driver experience, but neglects information security, as shown in [23]. Attacks performed by [6, 13] show that vehicles are highly vulnerable and its misuse can lead to hazardous situations.

Simultaneously, new technologies find their way into drivers' everyday life,

such as smartphones. These new technologies allow drivers to integrate several devices to a vehicle and, therefore, the vehicular communication system need a full make-over to consider external communication and information security. The SEIS project aims to tackle this issue by rethinking the onboard communication infrastructure in the future vehicles.

1.1 The SEIS Project

The SEIS Project (*Sicherheit in Engebetteten IP-basierten Systemen* - Security in Embedded IP-based Systems) [10] was launched in 2009 by the German Federal Ministry of Education and Research in the context of the Information and Technology 2020 (IKT2020) research program. The twelve partners from the German automotive industry are Alcatel-Lucent Deutschland AG, Audi AG, Audi Electronics Venture GmbH, BMW AG, BMW Forschung und Technik GmbH, Continental Automotive GmbH, Daimler AG, EADS Deutschland GmbH, Elektrobit Automotive GmbH, Infineon Technologies AG, Robert Bosch GmbH and Volkswagen AG, along with six laboratories, namely Technical University of Chemnitz, University of Erlangen-Nuremberg, Technical University of Munich, University of Karlsruhe, Fraunhofer Institute for Communication Systems (ESK) and Fraunhofer Institute for Secure Information Technology (SIT). BMW Group Research and Technology leads the system/software part and coordinates the SEIS Project.

The goal of the SEIS Project is to develop an integrated security IP-based solution for both internal and external vehicle communication. The deployment of IP as the standard communication protocol between ECUs, as well as communication with the outside world, allows for a completely homogeneous system and simplifies the current embedded system model. Removing gateways between the different vehicle buses and networks reduces translation overhead and management costs. Furthermore, IP will allow the vehicle to be integrated with any other devices in a global network, such as the driver's smartphone.

However, deploying IP into an embedded system brings all the security issues associated to this protocol, with increased hazardous potential due to the existence of life-threatening components, such as the vehicle braking system. Having that in mind, the SEIS Project seeks also a secure deployment of the new automotive architecture.

The current situation of the SEIS Project includes an experimental vehicle with a full IP-based system on it. Several standard components from PC or embedded systems were installed to allow testing and measurements. Likewise, vehicle controllers and multimedia content providers were integrated to

this experimental environment.

1.2 CE-Devices Integration

In the past three decades, Consumer Electronic Devices (CE-Devices) have become more popular and ubiquitous in everyday life. They have evolved from simple devices to complex systems capable of calling, playing audio and video, recording and my other functions. CE-Devices have a major role in the everyday life and become more integrated with systems that before were completely separated. In the near future, CE-Devices can be used to open a vehicle in a keyless fashion. Even stronger integration can be found in devices which are connected to the vehicle and exchange data with it, for example the Apple iPod Out, an interface developed by Apple Inc. [2] and fully supported in BMW vehicles to allow integration between Apple devices and the infotainment system of the car [12].

However, integrating CE-Devices and vehicles is not an easy task, because with these devices come several restrictions, such as battery life, data exchange limit and information security. Since devices have a shorter life span than vehicles, a user may need to integrate several devices to the same vehicle. Furthermore, one single vehicle may be associated at a given point in time to several different drivers and their own personal devices. Therefore, the vehicle could hold data belonging to different users and, among each of those, different devices, which form a scenario of potential information leakage.

Allowing any device to integrate the vehicle may lead to serious security breaches, some of which might violate driver's privacy and some which might threaten the life of the occupants of the vehicle. For that reason, vehicle manufacturers must do an analysis of potential risks before integrating these devices to vehicles they produce, in order to safely and securely allow the exchange between these devices and the vehicle. By carefully designing secured systems, it is possible to protect the user's and the vehicle manufacturer's privacy and comfort without compromising the vehicle safety and performance.

1.3 Challenges

The integration between CE-Devices and vehicles produces several advantages, such as entertainment (audio, video, games...) integration and customization, since usually a CE-Device belongs to one individual, whilst the

vehicle might be shared by several individuals, for example in a family car or vehicles belonging to a fleet. However, vehicles must be prepared against attacks that before were mainly applied to CE-Devices. While a malfunctioning application might crash a mobile device internal system, the same application might cause a fatal accident when connected to or executed inside a vehicle.

Current vehicles have several compartmentalized networks and their communication is made possible by gateways that translate the communication from one network to the next. Although this optimizes the purpose of each network, it does not integrate well with external networks. Vehicles designed by the SEIS Project, with IP-based systems, need no translation between networks. Since all components have an IP identification, it is easier to integrate all the networks, including the external devices. However, information and system security must be present to avoid CE-Devices attacks. Even though many attacks exist, we give focus to those related to information security, such as:

Impersonation By impersonating an user in a CE-Device, the attacker may gain unauthorized access to the vehicle internal network. With this access, the attacker might execute a malicious code and modify the vehicle's behavior, greatly endangering the passengers' life.

Information security The vehicle holds sensitive data related to its driver, the manufacturer and the vehicle itself. If this information is leaked by means of a malicious code or listening to traffic between the CE-Device and the vehicle, this affects both the driver's privacy and the manufacturer's reputation.

While data theft and impersonation can be reduced by means of some mechanisms, like two-way SSL/TLS authentication, it does not prevent data from being forwarded once its access is granted. The vehicle embedded system is distributed and each component has different security requirements. In some cases, a particular set of data belongs to more than one user or entity who has access to the car. These entities might not trust each other, posing the challenge of how to access and deliver this data. Even though two different drivers might trust the vehicle to hold and manipulate their common data, they might not want the other driver to retrieve it from the vehicle to his own device. Therefore, the main challenges associated to this thesis consist of integrating a full IP-based automotive distributed system with external devices regarding the information exchange and storage and privacy requirements from both communicating parties.

1.4 Contribution

Having in mind the security aspects of integrating vehicles and CE-Devices, this thesis has the purpose of designing a security solution for this integration, focusing on information security. Drivers willing to customize their driving experience by exchanging data or storing private data in the vehicle need a secure way of doing so. We intend to tackle this problem by controlling the information flow, in order to avoid protected data from being released or propagating to unauthorized parties. In our vehicle environment, data concerning the driver or the vehicle should remain private to those, allowing access from authorized devices and rejecting access from unauthorized ones.

For that purpose, we designed a use case in which information flow control is important and extracted the security requirements. At the end of the project, these requirements need to be fulfilled in a fully operational prototype to provide evaluation and assessment.

1.4.1 Driving Log

The Driving Log use case consists of a storage component inside the vehicle and its controlling software. Its main functions are to control storage, accesses, modification and removal of data to/from the storage component, based on its information flow control policies.

Storage Whenever a driver is making use of the vehicle, several ECUs are constantly sending information to the controlling software through the vehicle's internal network, which saves it in the storage component. When stored, this information keeps metadata in order to control its origin and accessing rights. The information is associated to the current driver and the ECU that produced the data.

Access Accessing the stored data is a critical action, as it depends on who is accessing it and what data is being requested. For that reason, whenever a CE-Device, external to the vehicle network, requests data, the request is considered as untrusted and therefore must go through security checks. The Proxy component, presented more carefully in Section 4.1.2, separates the internal and the external network and is capable of monitoring and filtering the packets going in or out of the vehicle. It can determine and propagate the security level of the external connection to the request, which helps The Driving Log controlling software to decide if the requested data can be retrieved by the application or device requesting it. If this is the case, it states the minimum

security requirements for releasing this data to the external network and the Proxy determines if the response is forwarded or not.

Modification Modifying data can be seen as an application who accesses data, performs one or more operations with it and modifies its properties. In order to retrieve the data, the controlling software will determine if the requester has access to this data. After manipulating the data, releasing it to the external network needs further security evaluation. According to its own policies, the controlling component can determine whether this information can be released and, if so, with which security requirements.

Removal Deleting data from the storage component requires accessing verification and deleting or modifying rights. The controlling component can determine this by the information about who requested the removal of this data.

To illustrate this use case, we can think of a Car Rental Company, whose vehicles are equipped with the Driving Log system. It offers access to CE-Devices owned by the company and applications running on each driver's CE-Device. When a driver makes use of the vehicle, his device connects to the vehicle, setting the current driver. Whilst the vehicle is driven by this driver, all data collected from the ECUs will be associated to him and stored in the storage component. To access the stored data, it is necessary to identify which information is sensitive to the driver's privacy and may not be released to the company device. For example, while the company should have access to oil and water level of its vehicles, it may not request information related to the places where the driver has been, i.e. GPS data. It is important that the controlling software is aware of this difference and is able to identify who is requesting the data. In case the driver wants to retrieve his own data, the Driving Log system should return only data related to this driver and not private to the vehicle or the company. Likewise, in case the company wants to retrieve information about the vehicle, it may not receive private information related to the driver.

To illustrate data modification, we can assume the driver has driven through several kinds of road types and the company would like to offer a better price for the rental in case the vehicle was often on highways and not so often on unpaved roads. For the price evaluation, the company can set up a price calculator that takes as input the odometer and road types (or GPS data) and returns the final price. However, since road type (or GPS data) is sensitive to the driver's privacy, this information may not be released to the company. Likewise, the price value for each road type might be private

to the company and should not be released to the driver. This characterizes a scenario of mutual distrust and, for that reason, an application must be placed inside the vehicle to be able to retrieve both data sets, manipulate it and transform it enough to make it possible to release both to the company and the driver. In our example, an application would calculate the final price based on the driver's road types and the company's price policy. The final price is a compilation of private data and may only be released to either company or the concerned driver.

Finally, in case the company wants to reset the vehicle to its original state, it may be allowed to erase stored data, without being able to read it. That way, the driver's privacy is not violated. On the other hand, a driver cannot erase its own data and prevent the company from calculating the final price.

This thesis focuses primarily on the confidentiality requirement, preventing sensitive information to flow from authorized parties to unauthorized parties by controlling its access rights. Furthermore, data integrity is also partially covered by this thesis, preventing an unauthorized entity from modifying data.

1.5 Outline

The rest of the document is structured as follows: in Chapter 2, the related work concerning Information Flow Control (IFC) and Decentralized IFC are presented; in Chapter 3, the concepts relevant to this thesis are defined and explained and they are important for the implementation of the system in Chapter 4, which brings also an introduction to the vehicle architecture and the Etch middleware used in this thesis; Chapter 5 contains the details of the security and performance evaluation tests, along with their results; finally, conclusion and future work are discussed in Chapter 6.

Information Flow Control

In this chapter, we perform a bibliography review on the Information Flow Control topic.

2.1 Introduction

Protection for privacy of data is nowadays a major research topic, as more and more data is transmitted over networks and manipulated by untrusted entities. Securing all this data requires computing systems to handle several different methods to limit the information disclosure. Methods like firewalls and cryptography prevent information to be released to unauthorized parties, but do not provide guarantees about the propagation of this information once it is released. For example, cryptography provides a way to exchange data privately through a non-secure channel, but does not guarantee the confidentiality of the data once it is decrypted on the receiving side.

This problem becomes even more complex when the system produces an output based on two or more sensitive inputs. As an illustration to this problem, we can think of a trader application, which relies on its clients' private data and its own trading policies. The inputs to the system are composed of both the clients' and the trading company's private data. The application computes a trade between two or more clients and produces an output about this trade, which needs to be released to all clients involved in the trade and the trading company itself. However, releasing the result of a trade might have significant business meanings to competing companies and even contain details of the trading policies belonging to the trading company. Any leak of private data might result in major reputation or financial issues, both to clients and the trading company. Note that traditional access control methods or cryptography does not help in this case.

Information Flow Control (IFC) tackles the problem by analyzing the flow of information throughout the system, assigning security levels to data and entities who manipulate it. The basic model consists of two levels, L (low) and H (high), representing, respectively, publicly available information and

secret information. Confidentiality and integrity are ensured by controlling the flow of information between these levels. In the case of confidentiality, information is not allowed to flow from H to L, or in other words, secret information is not allowed to become publicly available by computation. On the other hand, integrity is ensured by restricting flows from L to H. The general idea is that publicly available information is allowed to flow through low levels of confidentiality or up to higher levels, while high level of confidentiality cannot flow down to lower levels. More generally, the security levels can be viewed as a lattice with information flowing only upwards in the lattice [7].

Nonetheless, if information always becomes more and more restricted throughout the system, it will rarely output useful information that can be read by other entities. For that reason, privileges can be exercised in order to change the security level of the data or an entity. Privileges can be defined in several ways [16, 18], but follow the idea of trusted entities which can *declassify* or *endorse* data to, respectively, decrease the confidentiality level or increase the integrity level.

Information flow control may enforce information flow policies by means of static or dynamic methods. Furthermore, it can be applied to a system with a single host or a network. Regarding the policies, they can depend on one single central authority or scattered among all processes or hosts involved through which the data flows. In the next few sections, we present the different kinds of IFC and its advantages and disadvantages.

2.2 Static and Dynamic IFC

Static IFC is responsible for determining and analyzing all possible information flows during the program compilation. This requires special programming languages or extended annotations to traditional programming languages, as well as a special compiler, capable of determining all possible information flows. With the data flow map, the compiler can then enforce previously defined data flow policies and produce a feedback to the programmer. Depending on the violation, the compiler might be able to fix it, ignore it or abort the compilation.

Languages designed specifically for static information flow control, such as JFlow [17], face the problem of determining implicit data flows. Usually, this kind of flow happens when the computation of a variable depends on another variable. Figure 2.1, extracted from [17], shows a simple example where it is possible to know the value of the secret variable b by looking at the public variable x, even though x has only been assigned constant values.

```
int{public} x;
boolean{secret} b;
...
int x = 0;
if (b) {
        x = 1;
}
```

Figure 2.1: Implicit flow example.

The advantages of this technique include the ability to reject invalid programs even before they are executed, as well as avoiding extra overheads during execution time. Once the compiler determines a program does not violate any data flow policies, it can be executed without further dynamic checks. However, defining all data flows statically is not always feasible and, therefore, it is difficult to guarantee information security based only on the compiler. If a program has a variable that needs dynamic label assignment, the compiler might not be ready to cope with this assignment and either reject the program or, in the worst case, accept it, which could lead to dynamic security breaches. Furthermore, code annotations greatly impacts the code development phase and sometimes source codes are not available, preventing static analysis to take place.

On the other hand, Dynamic IFC happens during execution time, labeling data structures and input/output channels depending on which data flows through them. It has three different approaches: at application level, operating system level or a mix of both.

The dynamic IFC at the application level labels variables dynamically in the programming language or in the memory and enforces each new assignment or variable manipulation on real time [9]. However, this introduces an execution overhead that can affect performance-oriented systems. Furthermore, it must rely on the system underneath, because it depends on the operating system to write and read from the memory and, therefore, a compromised operating system would be able to affect and mislead the IFC system.

The second dynamic IFC approach concerns the dynamic enforcement under the operating system, in order to avoid the need to trust the OS. When IFC is enforced at the operating system level, the OS itself is modified to enforce the information flow policies. Resources are tainted according to

the policies and access to them restrained by security checks. For instance, if an application receives data from the network, there is a security check to determine if the application is allowed to receive untrusted data. Likewise, when an application sends data to the network output, there is a security check to determine whether sending this data out violates the information security policies. However, tainting operating system resources does not allow for fine-grained data flow enforcement. Examples of operating systems modified to support IFC are Asbestos [8], HiStar [25] and Flume [14].

Since both approaches can happen at the same time, the last possible approach is situated between the application and the operating system level and it tries to merge the best of the two other approaches by labeling both the operating system resources and the fine-grained data. An example of this is Laminar [21], which dynamically labels data structures in the Java Virtual Machine (JVM).

Dynamic IFC has the advantage of considering runtime environments and dynamic variables, unlike static IFC. For that reason, it is capable of protecting the privacy of data regardless of how the application was implemented or how the data is handled. This allows for applications to be written by external developers, completely unaware of the information flow control system. Thus, they do not need to worry about inserting annotations into the code or using a special compiler. Furthermore, in comparison to static IFC, developers need not release the source code in order to have the program's data flow analyzed by the IFC system. However, the disadvantages of dynamic IFC include an increased overhead during execution time. In the case of dynamic IFC at the operating system level, modifications to the operating system kernel are required and this can also be complex and costly.

2.3 Single-host and Distributed IFC

An information flow control based on a single host means that the flow policies must be enforced when different processes or users are exchanging data in the same machine. This is useful when systems have several environments, some of which are manipulated by untrusted users or execute untrusted applications. In TaintDroid [9], we can see an example of a dynamic IFC system enforcing secure information flows through the communication between different applications in an Android environment.

Although single-host systems are still numerous nowadays, they do not account for all data exchange systems, as more and more distributed systems become available. In a distributed system, several entities need to exchange data through a common channel, most frequently an unsecured network chan-

nel. As mentioned in the previous sections, there are several ways to protect the exchange of data through the network, but an IFC system is capable of protecting the data after it has arrived to the receiving side. Therefore, a distributed IFC system follows a set of data flow policies to control the flows between several entities, like in DStar [26].

Note that, like static and dynamic IFC, they are also not mutually excludent and systems can enforce both kinds at the same time. The set of data flow policies can determine flows within a single host and when exchanging data through the network. The system can cope with this by controlling the operating system's processes communication and network calls, for example.

2.4 Centralized and Decentralized IFC

Centralized and decentralized IFC depend only where the data flow policies are defined. In a centralized IFC system, policies are defined by a single entity and the whole system needs to recognize and trust the authority and correctness of this entity in order to enforce them. This does not concern the number of hosts in the system, since a single-host IFC system can have a centralized policy definition, in the same host or in another host, or a decentralized one, scattered throughout many authority entities.

The centralized IFC model provides a way to define all data flow policies in one single place and, therefore, make it easier to maintain or modify them. The policies need to be available for all systems enforcing an information flow control. However, some systems fall into cases where there is a mutual distrust between entities and a common authoritative entity holding all the data flow policies might not be feasible. For cases like this, the decentralized IFC model fits better. Although more complex, it allows each entity to define its own set of policies and enforcement and the whole system is composed of several smaller IFC systems. Back to the trader's example, the trading application may be reticent to store its trading policies in a centralized authority along with its users, so each entity can have its own policies and enforces them separately, with no need to share them among the other entities.

In the Decentralized IFC (DIFC) model, each entity enforces its own policy while sharing data. For that reason, processes sharing data in a single host or hosts communicating in a network will have their own set of policies and will enforce independently of the others. It is important to note, however, that even though each entity enforces its own different set of policies, the system as a whole must agree on the information flow control mechanisms and labels definition to avoid compatibility issues.

Several different approaches have been reviewed. The previously pre-

sented Asbestos [8], TaintDroid [9] and JFlow [17] apply the centralized IFC model. For the DIFC model, Jif [19] accounts for a programming level and static approach, where Meyers and Liskov adapt the decentralized model to the Java programming language. HiStar [25] applies the the DIFC model at the operating system level and DStar [26] extends the DIFC model to a fully connected network, implementing a web server as a proof of concept. Finally, DEFCon [16] implements a mixed static and dynamic DIFC system in an event processing system, with focus on information security and performance in a financial data processing scenario. Furthermore, Laminar [21] and Flume [14], mentioned in the previous section, also tackles the decentralized IFC issues.

Concepts

In this chapter, we define the concepts that are relevant to this thesis, based on the bibliography review presented in the previous chapter. The thesis consists of the implementation of a dynamic decentralized IFC system at the application level in a simulated automotive environment composed by several hosts. Our trusting base includes the operating system, the middleware above it and the part of the application which enforces the IFC. The goal is to protect the flow of information throughout the system by labeling both the data and the entities who access and manipulate it.

The approach closest to ours is DEFCon [16], in which flows of events are controlled in a high performance event processing system. Each event has a set of labels, which is evaluated throughout the system whenever events are processed. Even though the ideal deployment of this thesis would also occur in a performance-driven environment, i.e. the automotive environment, the focus of this thesis is on the information flow control aspects rather than improving performance.

To achieve that, entities and messages are tainted with security labels that designate the security level of that entity or message. Likewise, data is also contaminated with labels, which characterize how this data can be accessed by entities or encapsulated into messages. Labels are previously defined and all labels in the system must follow the standard definition to avoid compatibility issues. Furthermore, the evaluation of these labels according to the information flow policies occur in specific entities of the system and these entities are trusted by all the other entities who do not enforce the information flow.

In the next two sections, security labels are detailed, as well as how they can be evaluated.

3.1 Security Labels

In the decentralized information flow control model, data is monitored and enforced through the use of security labels. When labels are applied to a

Label (L_1)	
Confidentiality Set (S)	Integrity Set (I)
car, user	*ecu*

Figure 3.2: An example of a label.

Label (L_2)	
Confidentiality Set (S)	Integrity Set (I)
car.private, user.public	*ecu*

Figure 3.3: Remodeled label example.

piece of data, they compose the set of policies that restrict the flow of this data. Labels are composed of a pair of confidentiality set S and integrity set I, similar to labels defined in [16]. Each set is in turn composed of tags, which represent an indivisible policy concerning confidentiality, placed in S, or integrity, placed in I.

We can use the label $L_1 = \{(car, user); (ecu)\}$ depicted in Figure 3.2 to exemplify the labeling system. A piece of data associated to this label has confidentiality set $S = (car, user)$ and integrity set $I = (ecu)$. While confidentiality tags determine who is allowed to access this data, integrity tags define who produced that data or modified it for the last time. In the case of the label L_1, entities *car* and *user* can access the data associated to it and entity *ecu* produced this data.

Tags reflect directly the policies of the DIFC system and they must agree in all systems that enforce these policies. For example, if a piece of data is labeled with L_1, all entities enforcing policy rules must be able to understand what tags *car*, *user* and *ecu* mean to avoid compatibility issues in the system.

In our system, tags can also have an ending, which defines the public availability of the data. For example, the previous label L_1 can have more specific tags by adding *.private* or *.public* to the end of a tag. The redefined label L_2, depicted in Figure 3.3, when attached to a piece of data, shows that this data is private to the entity *car* and public to the entity *user*. This difference is important when the IFC system enforces its policies. Note that when a label contains a tag with no ending specifying the public availability, like *ecu* in Figure 3.3, it is considered by the system that it contains both *ecu.public* and *ecu.private*.

3.2 Information Flow Enforcement

Labels form a lattice, allowing for comparison and enforcement of how and where the labeled data flow. Data labeled with confidentiality set S_{data} can flow to a component with confidentiality set S_{comp} if and only if $S_{data} \subseteq S_{comp}$, where $\subseteq$ denotes the relation "can flow to" [26]. Therefore, the confidentiality set of the component needs to be a superset of confidentiality set of the data. This means that data labeled with a higher confidentiality set will not be able to flow into an entity with a lower confidentiality set, characterizing the "no read-up" property. In a practical example, if a file belonging to a general is classified as top secret (high confidentiality level), a lieutenant with clearance for confidential files only (low confidentiality level), but no top secret files, cannot see the file.

Integrity, however, works the other way around. The integrity set of the data needs to be a superset of the integrity set of the component, or $I_{data} \supseteq I_{comp}$. In other words, this indicates that the integrity level of the data needs to be higher or equal to the integrity level of the component. This property is called "no write-down", which states that an entity at a given security level must not write to any data at a lower integrity level.

Both properties "no read-up" and "no write-down" together form the Bell-LaPadula model for enforcing access control and information flow control [3]. If both relations hold, then this data can flow to this component. Generalizing the flowing relationship to labels, we have:

$$L_{data} \preceq L_{comp}$$

if and only if

$$S_{data} \subseteq S_{comp} \text{ and } I_{data} \supseteq I_{comp},$$

$$\text{where } L_{data} = \{S_{data}, I_{data}\} \text{ and } L_{comp} = \{S_{comp}, I_{comp}\}$$

With this lattice, once a confidentiality tag has been inserted in a label, data protected by that label cannot flow to components that do not have *at least* the same tag, unless privilege is exercised. That means that confidentiality tags are "sticky" and, once they are inserted and no privilege is exercised, they will be carried out throughout the system. Integrity tags are "fragile" and will be destroyed by manipulation over the labeled data.

To illustrate the enforcement of the information flow, consider a computation over two sets of data labeled $L_{d1} = \{S_{d1}, I_{d1}\}$ and $L_{d2} = \{S_{d2}, I_{d2}\}$, respectively, where $S_{d1} = (car)$, $I_{d1} = (ecu)$ and $S_{d2} = (user)$, $I_{d2} = (ecu)$. Both are given as input to a computing unit labeled $L_{comp} = \{S_{comp}, I_{comp}\}$,

where $S_{comp} = (car, user)$ and $I_{comp} = ()$, as depicted in Figure 3.1. When these two sets of data are given to the computing unit, there is an information flow policy enforcement. Since

$$S_{d1} = (car) \subseteq (car, user) = S_{comp}, \text{ and}$$

$$I_{d1} = (ecu) \supseteq () = I_{comp}$$

the first data set can flow to the component. Analogically, the comparisons

$$S_{d2} = (user) \subseteq (car, user) = S_{comp}, \text{ and}$$

$$I_{d2} = (ecu) \supseteq () = I_{comp}$$

allow the second data set also to flow to the component.

Note that the computing component is allowed to access both data sets, but the owners of each data set, namely *car* and *user*, would not be able to access each other's data. This scenario shows a mutual distrust between the owners of each data set.

After the computation, the output will have all confidentiality tags from the inputs, while the integrity set will be composed only of the integrity tag of the component. Hence, $L_{out} = \{(car, user); ()\}$. The output data has no integrity tags, because it was manipulated by an untrusted unit. However, it still holds both confidentiality tags, since it was computed from confidential data. Further computations will only increase the confidentiality level of the data and it will only be accessed by super privileged entities. In order to release this data in a lower confidentiality level so other units can access it, the computing unit might be granted a *declassification* privilege.

3.3 Privileges

From the previous example, we were able to see that output data sets will always have a higher confidentiality level than those of the input data sets. This, however, is not a realistic model, because data becomes increasingly restrictive to the point where no entity is able to access it anymore. To fix this problem, privileges are introduced in order to *declassify* or *endorse* data.

Declassification is the ability of reducing the confidentiality level of the labeled data. An entity with declassification privileges is able to release the data to entities that were previously not able to access it. Still referring to the previous example, if the computing unit is granted the privilege of declassification by the *car* entity, it will be able to change the output data

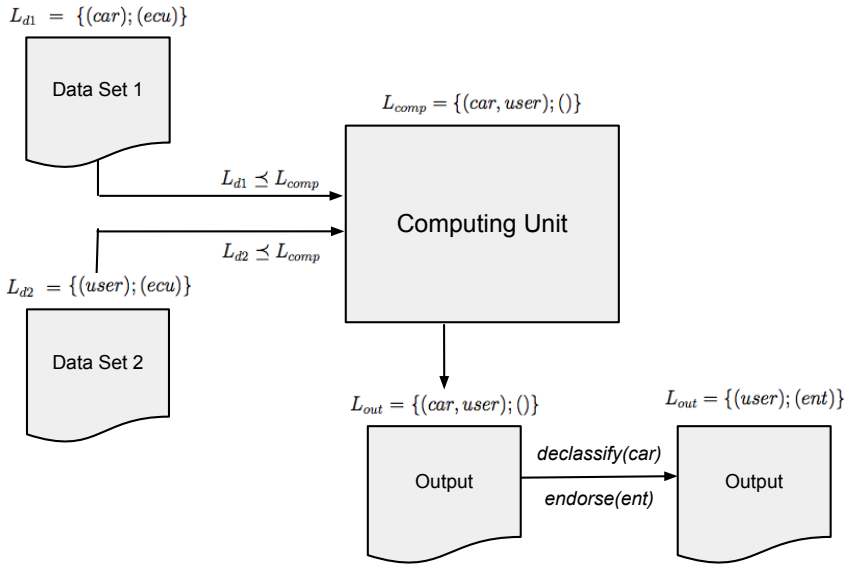

Figure 3.1: Information Flow example. Two data sets with different labels are used as input to a computing unit. Information flow control is applied to propagate and enforce labels through computation.

label to $L_{out} = \{(user); ()\}$, that is, it can remove the tag *car* from the output data and, therefore, declassify it to a lower confidentiality level. By doing this, the entity *user* is henceforth able to access the output data set.

Endorsement is the ability to raise the integrity level of the labeled data by trusting the integrity of it. In our previous example, if an entity *ent* completely trusts the data that is produced by the computing unit, it can give the endorsement privilege to it. By doing this, it allows the computing unit to include the tag *ent* in the output data label, which becomes $L_{out} = \{(car, user); (ent)\}$. This increases the security level of this data, because an entity who accepts data only trusted by the entity *ent* can now receive the output data.

These two privileges are used when it is necessary to decrease the confidentiality level or increase the confidentiality level of a certain labeled data. A concrete use case for this need is when data has been modified at the point

where it is not secret anymore, i.e. the computation over it does not allow to retrieve its original secret. In that case, an entity can give the computation unit the privileges to *declassify* its data after computation and, furthermore, it can *endorse* the data that is produced by the computation unit. Our previous example in Figure 3.1 shows a sketch of this use case in the last step performed by the computing unit, which applies the *declassification* privilege given by the *car* entity and removed the *car* tag. Furthermore, it also applied the *endorsement* privilege given by the *ent* entity and the output data has its integrity now trusted by *ent*.

The privileges are included as part of the information flow policies and define which entities are allowed to declassify or endorse which data, in a way that the system has a correct and functional flow.

Implementation

In this chapter, we will present the prototype that was developed and implementation details concerning the decentralized information flow control at application level.

4.1 Prototype Overview

The prototype architecture is illustrated in Figure 4.1. It is composed of a section representing the internal network of the vehicle and a section formed by every entity external to it. The internal network contains the vehicle electronic units (Electronic Controlling Units - ECUs), which provide some of the functionalities found in current vehicles, such as GPS and gas monitor. The external network represent the devices used by either the driver or the passengers and exchange data with the vehicle. Both sections are interconnected by a proxy unit, which mediate and control data that flow from one section to the other. It is important to highlight that the Proxy is the only channel through which data can flow from the internal to the external network and vice-versa.

In our prototype, the internal network contains four main components, which are the Driving Log Manager, the Database, the ECU and the Virtual Machine. On the other hand, the CE-Device is located in the external network and is completely untrusted. It exchanges data with the Driving Log Manager through the Proxy.

DL-Manager The Driving Log Manager component acts as a server and receives requests from all other entities. It is the only component which has access to the Database, where labeled data is stored. Its main functions are to receive requests, perform actions in the Database and enforce data flow security.

Database The Database is the only component which stores data in a persistent way. It contains only labeled data and the only entity allowed to connect to it is the DL-Manager.

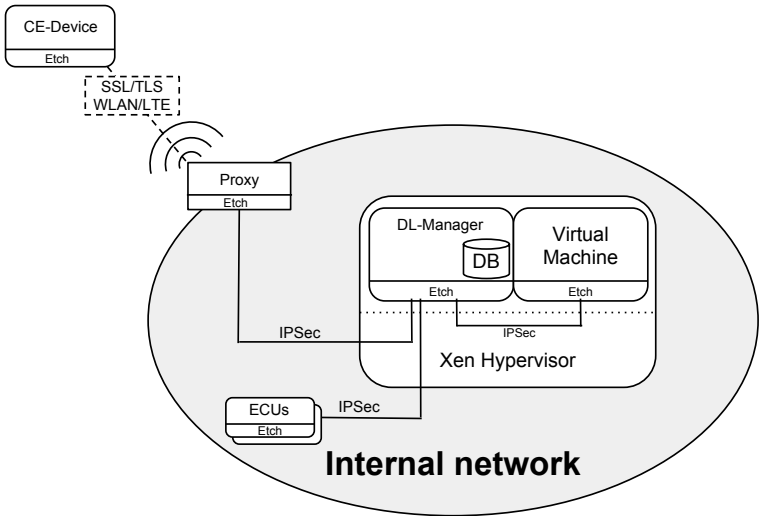

Figure 4.1: Prototype overview.

ECU The ECU component simulates a ECU inside the vehicle internal network and produces different kinds of data, which may be then associated to different entities. It is considered a trusted component, since it is in the internal network and does not exchange any data with untrusted entities.

VM The Virtual Machine executes untrusted applications in a protected environment, as detailed in [15]. This allows for a customization from external entities to run applications and manipulate protected data inside the vehicle. However, its inputs and outputs are monitored and controlled by the DL-Manager regarding the data flow.

CE-Device The mobile device external to the vehicle is considered untrusted and communicates with the internal network through the Proxy. Its requests are also monitored and controlled regarding the data flow.

Every component application, internal or external, runs on top of the Etch middleware. This middleware is responsible for building network services for communication between any pair of applications and carrying out the labels throughout the system. In the following sections, we will present a short introduction to the Etch middleware and the Proxy and how they are used during a regular execution.

4.1.1 Etch Middleware

The Etch middleware is a project of the Apache Foundation and is defined as a cross-platform, language- and transport-independent framework for building and consuming network services [1]. The Etch toolset comes with a network description language and a compiler and it fully supports currently C, C# and Java programming languages and TCP and UDP transport layer protocols. It is an on-going project which aims to simplify the definition of small, focused services.

Etch middleware is used by BMW Forschung und Technik as a communication solution for an IP-based automotive system. It has been intensively used and customized for BMW needs, in order to add automotive-specific and security features [4, 22]. It runs between the Application and the Transport layers of the OSI model and it provides a simple way for an application to communicate with a remote machine by means of previously defined network services. It is responsible for binding to the used programming language, encapsulating the message in packets to the desired transport and network protocols and handle the delivery on the receiving side. By trusting Etch middleware, the developer can focus on the application development and functionalities rather than on network communication.

The network description language defines the service between every client-server pair and contains the methods that are allowed to be called, and therefore the messages to be exchanged, as well as the direction.

In Figure 4.2, we can see the definition of the service *DrivingLog* and methods *requestData*, *requestPrice*, *calculatePrice* and *logIn*. The message direction is limited with the annotation *@Direction(Server||Client||Both)*. When this network service is defined, the Etch compiler produces a basic code in the targeted binding programming language with basic functionalities of message exchanging. The developer then customizes the source code to add the desired features at the application level.

In our prototype, the Etch middleware was used and bound to the Java programming language. All network services were defined and customized to our needs.

4.1.2 Proxy

The Proxy component stands between and separates two domains: internal vehicle network and external network. It was previously developed by colleagues from BMW Group Forschung und Technik in Java and acts as a mediator of the communication between the car and the external world [5].

The Proxy is capable of monitoring every packet going into the vehicle in

```
module com.bmw.proxy.drivinglog

/**
 * Driving Log service version of Driving Log Manager.
 */
service DrivingLog
{
        @Direction( Server )
        Map requestData(string driverName)
            throws OperationException, SecurityException

        @Direction( Server )
        float requestPrice(string driverName)
            throws OperationException

        @Direction( Client ) @AsyncReceiver( FREE )
        float calculatePrice(string driverName)
            throws OperationException

        @Direction( Server )
        void logIn(string username, string password)
            throws OperationException, SecurityException
}
```

Figure 4.2: Etch service sample.

order to determine its origin and security level. It uses a two-way SSL/TLS certificate to identify the origin of the request and also evaluates the security level of the connection based on the connection type, encryption level and other security aspects. With this information, it can filter the packet by either allowing it to go through or rejecting it for some reason, e.g. not enough security level.

Likewise, packets going out of the vehicle are analyzed to find out their destination and required security level. The Proxy can retrieve the desired security level from packets coming from the internal vehicle network and evaluate if the connection to the outside entity is secure enough for the packet to be forwarded.

The main function of the Proxy component is then to filter packets that

go through it. For example, if a connection between a device in the external network and the Proxy does not hold the security level defined by an outgoing packet, the Proxy does not forward it.

In this project, the Proxy was modified by Alexandre Bouard in order to label incoming connections based on the certificate information. Since it runs on top of the Etch middleware, it can propagate the label of the inbound connection or request to the internal network by means of the middleware data structure. The internal components, such as the DL-Manager, are therefore able to retrieve from the middleware the origins of each request. On an outbound connection, the Proxy can retrieve the required security level for the response from the Etch middleware data structure and determine if this response will be forwarded to the external client or not.

4.1.3 Data Flows

In this thesis, we identified three most important data flows between the prototype components. Depending on the type of flow, the Proxy may or not may be present. We present in this section each of the flows and the implemented flows are explained in the next section.

4.1.3.1 Internal Flow

An internal flow is a data exchange between components in the internal domain. For example, the data exchange between ECU and DL-Manager is an internal flow.

Internal Output Flow An internal output flow is a data flow going out of a component. It is composed of the output from the computation of a component. An example of this flow is the output data released by the ECU.

Internal Input Flow An internal input flow is a flow of data going in a component. It characterizes the input of data that will be used by that component. An example of this flow is when the DL-Manager receives an input of data coming from the ECU.

4.1.3.2 External Flow

An external flow is a data exchange between the vehicle and any external device. Since it is a multi-domain communication, the Proxy filters the communication accordingly.

External Output Flow An external output flow is a data flow from the vehicle to an external device. An example of this kind of flow is an external device retrieving some data from the vehicle through the DL-Manager. It is a crucial flow regarding information flow control principles, since sensitive data from the vehicle may be sent to an untrusted entity in the external network.

External Input Flow An external input flow is a data flow from an external device to the vehicle. It works the same way as the output flow, however in the opposite direction. This flow also crosses the internal and the external domains border and is monitored and filtered by the Proxy. An example is an external device storing data in the database controlled by the DL-Manager.

4.2 Driving Log Implementation

The system was completely developed in Java, having as base the code generated by the Etch middleware when compiled with the Java binding. The DL-Manager is a server and receives connections from the CE-Device, the ECU, and the Virtual Machine, following the client-server paradigm.

According to our use case, there are two kinds of CE-Devices, one belonging to the driver/user and another to a company that rents vehicles, referenced to as the rental company. The first one simulates the functionalities a driver would have by associating his device to the vehicle, such as logging in to the vehicle and retrieving his own data. The second one simulates the functionalities a company needs when renting its cars, such as retrieving vehicle data and calculating the final rental price.

The ideal execution of the system follows these steps:

Step 1 The rental company delivers a car to a new driver/user. This car has the DIFC system ready, as well as the Database and the Proxy components.

Step 2 The driver gets into the vehicle in order to use it and associates his device to it, either automatically or manually. In our prototype, this association is fulfilled by the use of the function *logIn*, called by the CE-Device and executed by the DL-Manager. When a driver logs in, the DL-Manager sets this driver as the current driver.

Step 3 While the car is in use, the ECU produces data regularly and sends it to the DL-Manager. The function *storeInformation* is called by the

ECU and executed by the DL-Manager and it stores the freshly produced data into the Database component. Note that this data is associated to the current driver, set in the previous step, and also with the vehicle itself. Some data belongs to the driver only, some to the vehicle only and some to both. If the car is in use and no driver is set as the current driver, the produced data is dropped by the DL-Manager.

Step 4 Still with the car in use, the driver can retrieve his own data, that is, the data that is associated to him in the database. The DL-Manager retrieves this data from the database, enforces the IFC policies and returns the data to the driver's CE-Device, according to the data labels. Note that the driver can only retrieve the data that belongs to him and not to the vehicle. The data flow enforcement is better detailed in section 4.2.2.

Step 5 After the car is returned to the rental company, both the driver and the company can use their own CE-Devices to retrieve data belonging to each of them. While the driver would get personal data, such as location and speed, the company would get data concerning the vehicle, such as gas level and mileage. The DL-Manager also enforces the IFC policies during this step.

Step 6 At this point, the rental company would like to calculate the final price related to the car rental. Since this company charges a different price depending on how the vehicle was used, the final price depends on both personal driver data and vehicle data. This calculation is done by an application inserted in a protected environment inside the vehicle, called Virtual Machine. It executes an untrusted code which takes as input sensitive stored information from both entities and produces as output only the final price based on the company's price policies. This protected environment was projected and implemented by Esko Mattila [15] and is also labeled by the DL-Manager to keep track of the confidentiality and integrity issues associated to this operation.

Step 7 Since the price is calculated and the rental period is over, the last step is optional. The rental company may desire to delete all data associated to the driver. It is allowed to do so, since it owns the car, but it is not allowed to retrieve the data.

There are several alternative execution paths for the system. However, this execution allows for a good evaluation concerning the security and the DIFC system. In the next sections, we will give focus to the database implementation and the information flow policies enforcement.

4.2.1 Database Model

Since all stored data is associated to a label, the database must also be label-aware. Two different models were developed and implemented using MySQL [20] to identify the best option in terms of performance and security. The next subsections present both database models and their advantages and disadvantages, as well as the performance comparison and the final selection.

4.2.1.1 First Model

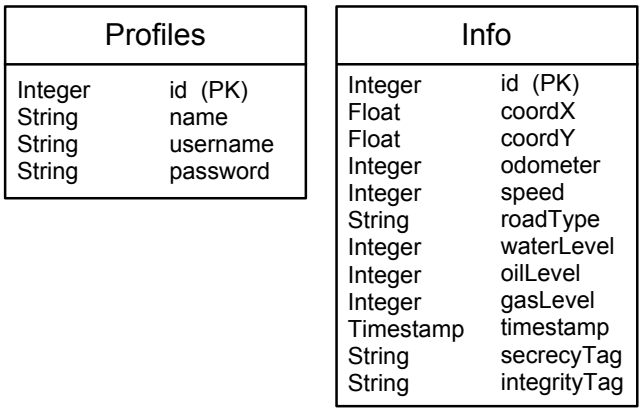

Figure 4.3: First database model.

The first database model, represented in Figure 4.3, consists of two tables: Profiles and Info. The Profile table holds authentication data and its fields are *id*, *name*, *username* and *password*. Its entries contain each profile that connects to DL-Manager, such as every driver, company, VM client and ECU. The Info table, on the other hand, holds every data stored in the database and its fields are *id*, *coordX*, *coordY*, *odometer*, *speed*, *roadType*, *waterLevel*, *oilLevel*, *gasLevel*, *timestamp*, *secrecyTag* and *integrityTag*. Every field is optional, with the exception of the *id* field, which is an autoincrement primary key. Each entry contains some or all values and *secrecyTag* and *integrityTag* compose the label associated to that data. In each of the tag fields, values separated by commas define the entities of the confidentiality and the integrity sets.

The simplified database entry shown in Figure 4.4 represents data that sets values for *odometer*, *roadType* and *timestamp*. As it is possible to see from the *secrecyTag* and *integrityTag* fields, this piece of data is tagged as public data of entities *daniel* and *car* and was produced by the entity *ecu*.

id	coordX	coordY	odometer	speed	roadType
3	NULL	NULL	24030	NULL	Highway

...

timestamp	secrecyTag	integrityTag
2012-07-10 13:50:28	daniel.public, car.public	ecu

Figure 4.4: Database entry sample.

This model is very simple and defines no relationship between the tables. Henceforth, the *secrecyTag* and the *integrityTag* fields in the Info table are not constrained by a foreign key from the Profiles table, leaving those two fields open to any string, which can lead to dangerous data mishandling. However, they can be expanded to as many entities as desired without creating new entries.

4.2.1.2 Second Model

To overcome the unconstrained *secrecyTag* and *integrityTag* fields problem, another model was designed and implemented. This model consists of four tables, namely Profiles, Info, SecrecyTag and IntegrityTag, represented in Figure 4.5. Profiles and Info are very similar to the same tables in the first database model. However, they are now related to each other and the other two tables implement the relationship between them. SecrecyTag and IntegrityTag are junction tables, also called cross-reference tables. Their main function is to avoid the many-to-many relationship between Profiles and Info tables, which happen because one entity in the Profiles table can be associated to several entries in the Info table and one single entry in the Info table can also be associated to different entities from the Profiles table.

The only difference between the Profiles and Info tables from the old model to the new model is that the Info table no longer has the *secrecyTag* and *integrityTag* fields. The association between one entry of the Info table to a confidentiality and integrity label is done through the SecrecyTag and IntegrityTag tables, which are both composed by the fields *id*, *info_id* and *profile*. To understand this association, consider a piece of data with *id* number 4 in the Info table and associated to the profile *daniel*, which has *id* 12 in the Profiles table. The SecrecyTag entry would then include its own *id*, the field *info_id* set to 4 and the field *profile* set to 12.

Although this model is more complicated because of the relationships between the tables, it constrains the possible values of association between

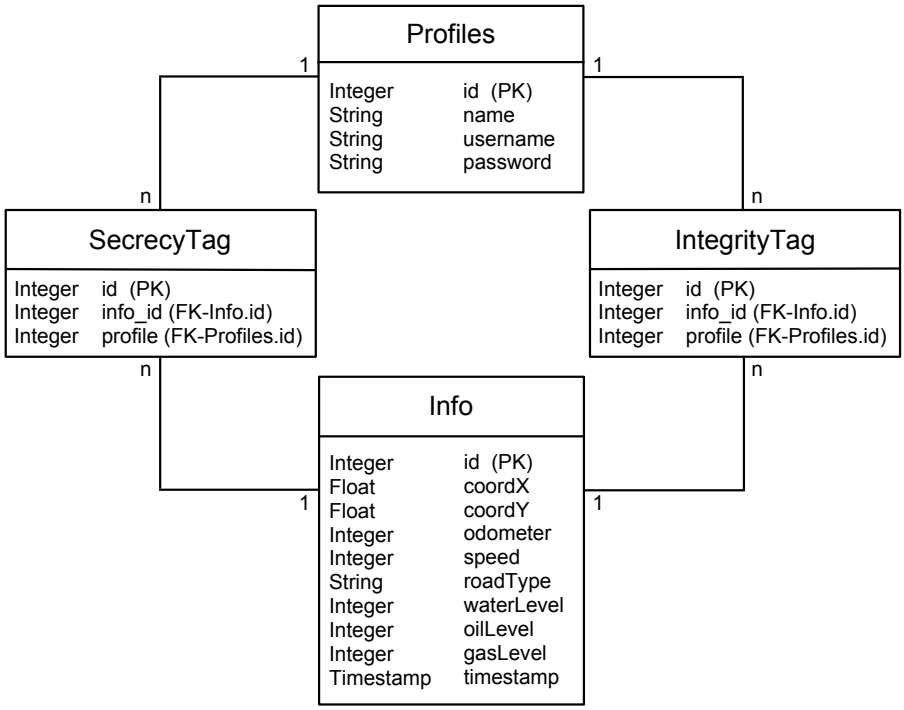

Figure 4.5: Second database model.

an entity and a piece of data. However, in order to associate a piece of a data to a new profile, a new entry in both SecrecyTag and IntegrityTag tables must be created, greatly enlarging the size of these tables.

4.2.1.3 Comparison and Selection

The comparison between the two database models was done by calculating the time it takes to retrieve data belonging to at least two profiles. The company client asked for user data. Since the company is requesting it, it can access both private or public company data. However, only publicly available user data should be returned. Therefore, the DL-Manager needed to return only data with labels *user.public* and *company.public* or *company.private*.

The algorithm to retrieve this data from the first database model is very simple.

Step 1 The DL-Manager builds a caller list L_{caller} composed by the company tags *company.public* and *company.private* and the public data that it is requesting, *user.public*.

Step 2 The DL-Manager searches for every entry that contains *user.public* in the *secrecyTag* field of the Info table. For each entry, the *secrecyTag* is retrieved and composes the data secrecy labels L_{data}.

Step 3 Finally, it checks if all tags in L_{data} are in L_{caller}. If this is the case, this entry is returned. This guarantees that only entries with both *user.public* and either *company.public*, *company.private* or both are retrieved by doing only small lists and string comparisons.

The algorithm to retrieve this data from the second database model is more complicated due to the relationship between Info and Profiles.

Step 1 The DL-Manager retrieves from the Profile table the *id* associated to the user whose data is being requested.

Step 2 The DL-Manager retrieves from the SecrecyTag table all entries from Info that are associated to this user, i.e. all entries in which $SecrecyTag.profile = id$. For each entry, the DL-Manager retrieves the $SecrecyTag.info\_id$ field.

Step 3 For each *info_id* retrieved in Step 2, the DL-Manager searches a second time in the SecrecyTag table for other entries with the same value, meaning that this entry is associated to other profiles. If it is the case where this data is associated to another profile, the DL-Manager looks up in the Profiles table to which entity it is associated. In case it is *company.public* or *company.private*, the entry can be retrieved. If not, this entry may not be returned.

Both algorithms were executed during the tests and their time of execution measured. Each test includes 6 time measurements, each composed of 2000 requests. Since the middleware has a limitation of packets up to 16 Kilobytes, the tests were performed to return from 0 to 16 entries. The full description of the scenario where these tests were performed can be found in section 5.1.2.

The results presented in Figure 4.6 show that the first database model has a better retrieving efficiency. The reason for that can be explained by the simpler algorithm and the low number of entries. A better algorithm for the second model could improve its efficiency, but it was not the scope of this thesis.

Having in mind the difference between the database models, the first model was chosen for its simplicity and performance. The rest of this thesis considers operations performed with the model represented in Figure 4.3. The SQL Code to create the database and populate the Profiles table can be found in the Appendix A.

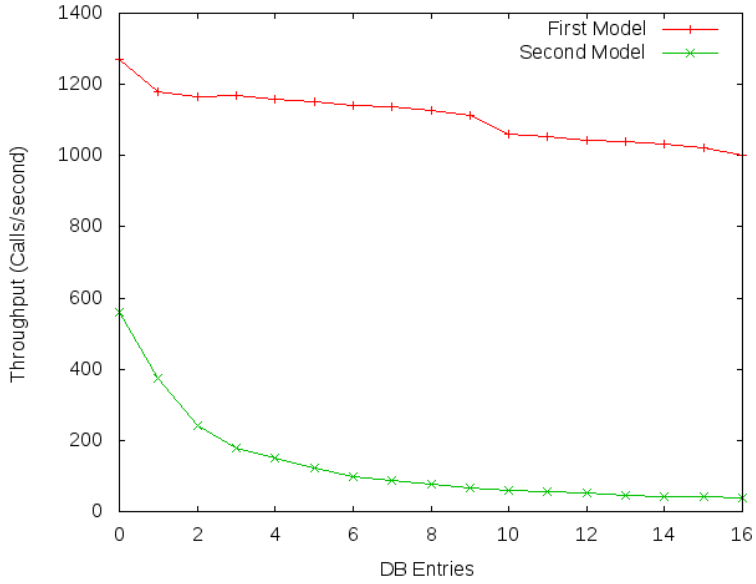

Figure 4.6: Database models comparison.

4.2.2 Security Enforcement at the Application Level

The DL-Manager component is responsible for enforcing the data flow policies. In this section, we explain how this enforcement is done for each of the four functions defined in section 1.4.1, namely storage, access, manipulation and removal of data.

4.2.2.1 Storage

In order to store data, a driver must be using the vehicle, simulated by the *logIn* function. The Driver CE-Device connects to the proxy unit and offers a certificate to prove its identity. The proxy extracts identification information from the certificate and uses the middleware to taint this connection's origin and security level. The request is then forwarded to the DL-Manager, which executes the *logIn* function. The parameters contain the username and the password of the driver who wishes to log in. These parameters are used to authenticate the driver by means of the registered profiles in the database table Profiles. If the profile exists and the password matches, the DL-Manager retrieves from the middleware the origin to make sure the driver is who he claims to be, before setting the active driver. Finally, the DL-Manager sets the middleware label of the response and returns a successful login response.

The Proxy is able to read this tag and determine if the security level of the connection with the driver is strong enough to forward the response.

With the vehicle in use and the active driver set, the ECU periodically produces data. Our ECU produced each time a list of all possible data, namely the coordinates *coordX* and *coordY*, *odometer*, *speed*, *roadType*, *waterLevel*, *oilLevel*, *gasLevel* and the current *timestamp*. According to the policies defined in the DL-Manager, *coordX*, *coordY* and *speed* are private driver data, while *waterLevel*, *oilLevel* and *gasLevel* are private company (vehicle) data. Finally, *odometer* and *roadType* are considered as public data belonging both to the driver and the vehicle.

Whenever DL-Manager receives a set of data from the ECU, it consults its policies and divides the data accordingly. It stores the private driver data with label $L_{driver} = \{activeDriver.private; ecu\}$, where *activeDriver* is the current driver using the vehicle. Likewise, private vehicle data is stored with label $L_{vehicle} = \{company.private; ecu\}$. Note that the label for vehicle data is *company*, because according to the policies, every data belonging to the vehicle belongs to the company as well. Finally, data belonging to both entities are stored with label $L_{both} = \{activeDriver.public, company.public; ecu\}$.The field *timestamp* is repeated in all three storage entries.

4.2.2.2 Access

The DIFC system plays an important role regarding the data protection when data is retrieved from the database. In our system, two kinds of accessing data were considered: from an external device and from the virtual machine. In this section, we explain how accessing data from an external device works. The access from the virtual machine is considered in the next section, during data manipulation.

Like in the *logIn* function explained in the Storage section, when a device connects to the Proxy, it offers a certificate to prove its identity. The Proxy forms a label based on the certificate and the connection security configuration and inserts it into the middleware layer. Upon data request from the device, the Proxy forwards both the request and the label to the DL-Manager component, which can then process the request and retrieve the label from the middleware.

The requested data is retrieved from the database, along with its associated label. For every retrieved entry, the DL-Manager will verify if the data flow policy holds and the data can flow to this client regarding its confidentiality level, that is

$$L_{data} \preceq L_{client}, \text{ or}$$

$$S_{data} \subseteq S_{client}.$$

If it is the case where both expressions holds true, the DL-Manager will add this entry to the returned data set. Finally, this set is returned and the middleware security label is now reset to a new value, so that the Proxy can determine if the result data can flow to the client, having in mind the connected device and the connection security level. Since in our system only the confidentiality policies were carefully defined, the integrity flow verification was not completely done at this stage and left for future work. It is important to remind that the middleware has a 16 Kilobytes packet size limit, which constrains our resulting data to 16 entries. The algorithm is the same used to perform the database evaluation for the first database model in Section 4.2.1.3. The full algorithm implemented in Java can be found in the function *requestData* in the Appendix B.

4.2.2.3 Modification

In order to handle and modify data, there are some extra requirements. Considering an application that modifies driver's sensitive data, we argue that this application must run within the vehicle in the interest of the data protection, that is, the sensitive data must not leave the internal network. However, running a third-party application inside the vehicle can also lead to dangerous consequences, since the developer might not be known or trusted. Therefore, we set up a secure environment using the Xen Hypervisor [24] and each virtual machine running on top of it has an associated label. Both the virtual machine identification and the label associated to it are stored at the DL-Manager at the middleware level.

A virtual machine is created with the purpose of executing a specific untrusted code and, therefore, its labels do not change over time. Whenever a new virtual machine is created and registered at the DL-Manager, the label for this machine is set according to the operations it will perform. Once set, this label will determine which data can flow to this machine and the labels of the data sent as output until the machine is destroyed. Virtual machine reuse by another entity is not allowed to avoid data leakage.

Upon a request from an external device for some kind of data modification, the same first steps mentioned for Storage and Access are followed. The request is tainted by the Proxy and forwarded to the DL-Manager, which in turn retrieves the label from the middleware to find out who is requesting it. With that information, the DL-Manager can find the virtual machine associated to the requesting entity and forward the request to handle data. The virtual machine will then request the actual data from DL-Manager to

modify and the decision of accessing this data is similar to the one explained in the Access section, based on the requested data label and the virtual machine label. After executing the untrusted code, the virtual machine returns an output back to DL-Manager, which labels this data with the same virtual machine labels. The Proxy can retrieve this label and decide if the response will be forwarded to the client or not.

The example use case is PriceApp, an application that calculates the final price to be paid by the driver to the rental company by accessing both driver's and company's data. The PriceApp belongs to the company and therefore is labeled with confidentiality set $S_{vm} = (company, driver.public)$. That means it can access any data labeled with company tag and also public data belonging to the driver. When the company requests the price, the DL-Manager forwards it to the virtual machine. If another external device tries to use this virtual machine, the request is not forwarded, since the request and the virtual machine labels do not match.

The PriceApp then requests data to calculate the price. This data consists of the fields *roadType* and *odometer*, which belong both to the driver and the company. The DL-Manager verifies from which virtual machine this request is coming from and enforces the data flow to make sure the PriceApp is only retrieving data with confidentiality level lower or equal to $S_{vm} = (company, driver.public)$. After receiving the requested data, the PriceApp calculates the final price and returns it to the DL-Manager. By this time, the DL-Manager would set the response label to the same labels of the virtual machine, i.e. $L_{vm} = \{company, driver.public; vm\_priceapp\}$. This means that only an entity with confidentiality set higher or equal to $S_{vm} = (company, driver.public)$ would receive this data. However, the company does not have such confidentiality level. In order for the company to receive back the response, the DL-Manager must *declassify* the data by removing the *driver.public* tag. It can do it because it is a public data and the user gave the vehicle the privilege to release this data, according to the policies. Therefore, the final label set in the middleware for the response is $L_{vm} = \{company; vm\_priceapp\}$ and the proxy will forward this response to the company device.

4.2.2.4 Removal

Removing data works similarly to accessing data. Even though no data is sent back in response to a device, the DL-Manager must check the label of the request and compare to the policies in order to know if that entry can be removed or not. According to the policies defined by us for this system, only the company device can request data removal. Therefore, upon connection

and request, the Proxy will label the connection and forward the request to the DL-Manager. The DL-Manager is able to retrieve this label and determine who is requesting. In case it is an entity allowed to remove data, as the company in our prototype, it will retrieve data from the database. For each data entry, the DL-Manager will decide based on the data label if that entry can be deleted by the requesting entity. If it is the case, the DL-Manager temporarily declassifies the data by removing its confidentiality levels and allowing the company to remove it, since, after declassification, $L_{data} \preceq L_{company}$.

Note that the company is allowed to remove *all* driver's private entries at once, but is not allowed to retrieve it or access it. This would reflect the real scenario where a company would like to reset the vehicle and remove all previously recorded data, after the end of a rental period, for example. For that reason, the company device is able to remove either all entries associated to a driver either its own entries, i.e. the data of the car, such as *odometer*, *gasLevel* and so on.

Evaluation

The system was systematically evaluated and results were drawn to measure the overhead introduced by the DIFC enforcement at the Application level. In this chapter, we present the environment that was set up to evaluate the system and discuss the results from both performance and security perspective.

5.1 Environment and Methodology

Three computers were used for the evaluation, simulating the CE-Device/ECU, the Proxy component and the DL-Manager.

CE-Device/ECU 2x Intel Core 2 Duo E8400 @3.00 GHz with 3 GB memory running Linux/Ubuntu 10.04.1.

Proxy 2x Intel Core 2 6400 @2.13 GHz with 2 GB memory running Linux/Fedora 16.

DL-Manager 2x Intel Core 2 Duo E8400 @3.00 GHz with 6 GB memory running Linux/Fedora 16 and Xen Hypervisor 4.1.2. Virtual machines are also Linux/Fedora 16.

The database was implemented with MySQL 5.1.41 on the Ubuntu computer and accessed via network by the DL-Manager.

The network settings included an wired Gigabit Ethernet between all computers. Above it, the IP protocol runs between CE-Device and Proxy, while Proxy-DL-Manager and DL-Manager-Virtual Machine are IPSec [11] connections. Furthermore, between CE-Device and Proxy, there is also an SSL/TLS [11] layer at the Application level with two-way authentication.

This evaluation scenario reflects the system in a real environment. While the connection between an external entity and the proxy unit happens over IP and SSL/TLS, the vehicle's internal communication doesn't require SSL and was chosen to be IPSec, according to the SEIS Project, for improved security.

5.1.1 Global System Evaluation

The overall system evaluation was performed by means of requests from the CE-Device to the virtual machine. The request flows through the SSL/TLS and IP connection to the Proxy and then is forwarded over the IPSec connection to the DL-Manager. In turn, the DL-Manager finds the suitable virtual machine and sends a request to it also over IPSec, which simply returns any number, i.e. it does not compute any output. The number is returned from the DL-Manager through the Proxy back to the CE-Device. Note that this evaluation does not include any database requests or DIFC enforcement, but simply evaluates the whole system architecture in terms of performance.

The requests were executed in chunks of 2000 and the Java *nanoTime* method, provided by the *System* class, was used to gather timing data. The time for 2000 full calls was measured 6 times and the average was calculated. The reference to compare this result to is composed of plain connections, where no SSL and no IPSec are deployed, and obtained from [15].

5.1.2 DIFC Enforcement at Application Level Evaluation

While the overall system evaluation gives an idea of the full system performance, the DIFC enforcement is not measured. For that reason, we also evaluated the application level enforcement through requests to the database, reading data labels and enforcing the information flow policies. The previous scenario configuration is not suitable for this evaluation, since the Proxy and the SSL connection introduce too much overhead and mask the request timing measurements. Furthermore, to guarantee no variations due to virtual machine accessing, the Xen Hypervisor was also removed from this evaluation.

The scenario consists of direct requests from a client to the DL-Manager over an IP connection. This client is considered a trusted one, for example an ECU internal to the vehicle, which inserts its own label into the middleware. The DL-Manager is capable of accessing this label and retrieve who is requesting the data, i.e. the label of the incoming request. This substitutes the function the Proxy component would do in the ideal scenario. The DL-Manager then accesses the database and retrieves the requested data. It enforces the information flow by comparing the request and the data labels to determine which data is allowed to be returned to the client, before setting up the labels of the response.

This test follows the previously methodology and is executed 6 times in chunks of 2000 requests, giving a total request number of 12000. This is

executed for each number of retrieved database entries, from 0 to 16. The 16 entries limit is due to the 16 Kilobytes limit of the Etch middleware. The mean is calculated for the 6 calculated times of each entry.

5.2 Results

In this section, we present the results of both evaluation scenarios described in the previous section. While the overall system evaluation gives an idea of the system performance, the DIFC Enforcement evaluation shows the overhead introduced by each data enforcement.

5.2.1 Global System Performance

In Table 5.4, it is possible to see the comparison between the system with simple requests to the virtual machine and the system enhanced with network security, including SSL/TLS and IPSec. The later one is over 4 times slower than the former, however most of this overhead comes from the SSL/TLS encryption and connection establishment, as shown in [15].

	Time (*nanosec*)	Throughput (*calls/sec*)
Plain connection	20.262	98.707
SSL/TLS + IPSec	96.569	20.711

Table 5.4: Overall system performance results.

This is an important result, because this big overhead is considered for the evaluation of the IFC enforcement. When SSL/TLS is integrated to the DIFC evaluation, it masks the DIFC overhead and it is not possible to clearly compare it and determine its performance. For this reason, we adapted the evaluation of the IFC enforcement performance to a more suitable one, without the SSL/TLS security layer, which allowed us to see the overhead introduced by the DIFC system.

5.2.2 DIFC Enforcement at Application Level

In this section, we present the results concerning the Decentralized Information Flow Control enforcement at the Application level, considered from the point of view of performance and security requirements.

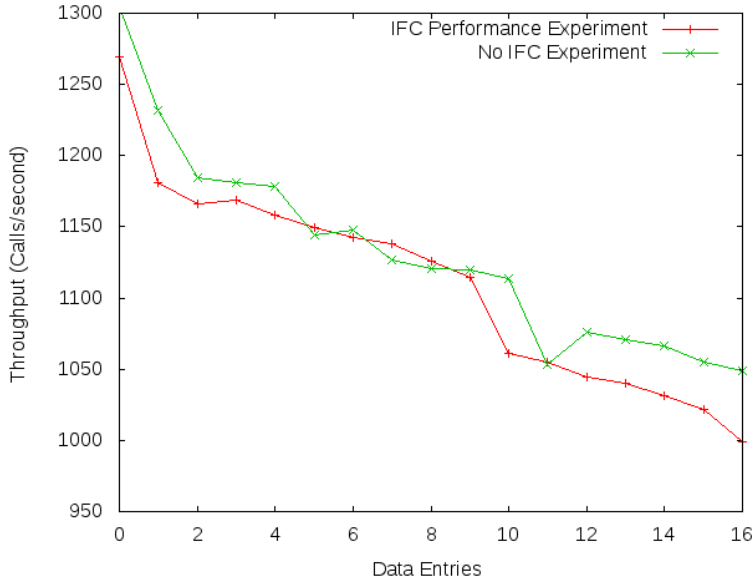

Figure 5.1: DIFC experiment comparison.

5.2.2.1 Performance

The DIFC Enforcement results are shown in Figure 5.1. It shows the average number of calls per second for each number of data entries retrieved from the database. While the first line shows the experiment values having the data flow policies enforced, the second line shows the reference, i.e. when no security enforcement is applied and entries from the database are simply returned.

It is possible to see that the best fit for this experiment is a line in the form of $f(x) = B + Ax$. The graph presented in Figure 5.2 shows the best line fit for both the IFC experiment ($B = 1220.53$, $A = -13.8401$) and the reference values ($B = 1234.93$, $A = -13.0188$).

The first observation drawn from this result is that the experiment has a linear decrease compared to the number of entries. This is an expected result, since each entry retrieved from the database increases the overall execution time and, therefore, decreases the throughput of calls per second.

The second observation relates the experiment and the reference throughput averages. By comparing both line fits in Figure 5.2, we can see that the overhead introduced by the IFC system is between 1.16% and 2.68% for up to 16 entries. However, since both lines are not parallel, this overhead tends to increase when more entries are added to the evaluation, compromising the

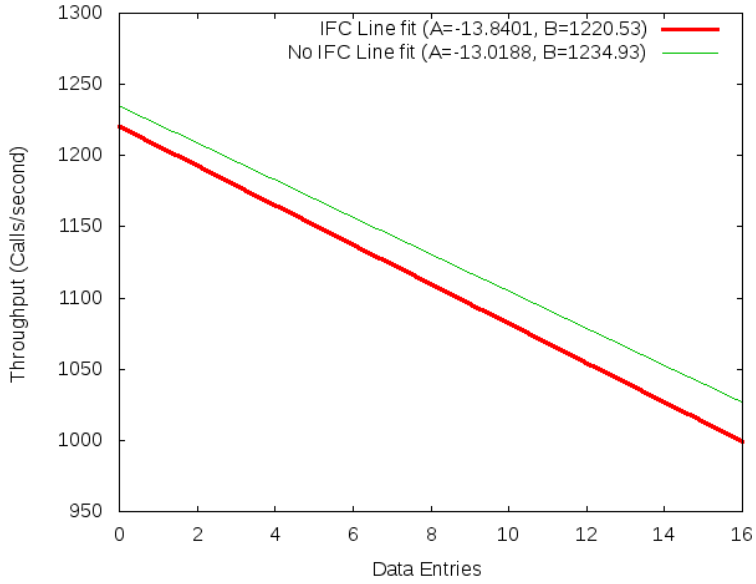

Figure 5.2: DIFC line fit comparison.

scalability property of the system.

This result shows that DIFC enforcement at the application level follows a linear decrease compared to the number of entries, when handling few data entries. Furthermore, it shows that the overhead for low number of entries lies below 3%. If applied to large data sets, it will linearly decrease its performance and might not be suitable for a large automotive system. However, the algorithm can also be further optimized to handle large data sets more efficiently.

5.2.2.2 Security

In terms of security, the prototype successfully fulfilled the confidentiality requirements introduced by the thesis. To illustrate this, consider the following attack cases:

Malicious Driver Device In case a malicious driver connects to the vehicle, but does not have the ability to spoof an SSL/TLS certificate, the Proxy will insert this driver's information into the middleware tag. Whenever this device tries to retrieve data, the DL-Manager will retrieve the tag inserted by the Proxy and retrieve the desired data. However, if it tries to access company data, another driver's data or

any data to which it does not have the right label, the DL-Manager will prevent the data to *flow* back to the malicious driver device.

Malicious Company Device The company device is allowed to perform more actions than the driver device, such as to delete data, but requests from both the company and the driver are treated the same way. Like in the malicious driver device case, the company is also not allowed to retrieve any driver's private data, as long as it can not spoof the SSL/TLS certificate. The DL-Manager would prevent unauthorized data to flow back to the malicious company device.

Malicious Virtual Machine In the case of the Virtual Machine, it always has a purpose and is associated to an entity, which makes use of it. If the VM belongs to the company and its purpose is to manipulate some driver's data, the DL-Manager will not let any other driver's data (public or private) flow to this machine. In the same way, every output will be associated to the company and the driver whose data was manipulated. According to the declassification privileges, it will be accessible by the company, the driver or both. By defining the declassification privileges, it is possible to prevent unauthorized data or manipulated data to flow to one of these parties.

It is important to note the importance of the SSL/TLS certificate for our system to manage external devices properly. If the SSL/TLS certificate is not trustworthy, the information flow might be compromised. Likewise, the Proxy component, which handles the SSL/TLS certificates, must also not be compromised, since its malfunction would lead to a wrong interpretation of the requester. Finally, we assume that the components in the internal network (ECUs) are also trusted and each one has an identification. Compromised components, specially DL-Manager, which enforces the information flow, would greatly impact the designed system.

Conclusion

This thesis aimed to tackle the problem of protecting flows of data in an automotive environment. It made use of the concepts of Decentralized Information Flow Control to implement a prototype, which served as a proof of concept.

The prototype simulated an automotive environment composed of external and internal networks, connected by a Proxy device. The external domain contained a computer simulating the external CE-Device, which connected to the vehicle's internal network through the Proxy in order to request data. The internal network was composed of a computer with the request manager (DL-Manager), a computer simulating an Electronic Controlling Unit (ECU) and a database. The DL-Manager also consisted of a Xen Hypervisor environment to create and execute virtual machines, which executed untrusted third-party applications that manipulated sensitive data.

The goal of this thesis was to control the data flow between every component in the system at the application layer. For this, the stored data was labeled when inserted in the database, as well as the requests and responses for this data. The DL-Manager was responsible for enforcing the policies of data flow and setting up the response labels for the Proxy to be able to determine if the connection to the external device is strong enough to receive the data. Furthermore, the underlying Etch middleware controlled the exchange of messages and abstracted the network programming from the application developers.

6.1 Achievements

Future CE-Device and vehicle integration will demand a way of controlling the exchange of information. For this scenario, DIFC becomes important and this thesis successfully applied its concepts to the vehicle environment. Furthermore, the implemented prototype served as a proof of concept. However, the performance results show that the system has still to be improved to be deployed in a real vehicle.

Personally, during this project I had the chance to know how automotive systems work to this day and how they are going to be in a couple of years. I had the chance to work in a great multicultural work environment and improve social and technical skills, which include database modeling, installation, configuration and manipulation, Java programming, services in the Etch middleware and LaTeX.

6.2 Future Work

The future work related to this thesis are related to system improvement, scaling or deployment.

Integrity Policies The system considers primarily confidentiality issues in its policies, with some important integrity control. However, integrity policies could be improved to be fully considered throughout the information flow and, therefore, increase the system's security level by enforcing writing rights.

Database Algorithms The algorithms to retrieve and store data are not optimal, since the author is not an experienced developer and the focus of the thesis is not on performance issues. A better algorithm or database model would enhance the system performance and safety, allowing a real deployment on a vehicle.

Scaling The system as it is does not scale to a large set of data, because of the Etch restriction of 16 Kilobytes for a packet size. This could be enhanced, either by increasing the size or serializing the data to be sent. Furthermore, the system could also consider larger environments, with several ECUs and devices connected at the same time to the DL-Manager and accessing the database.

Vehicle Deployment To fully comprehend the system and its properties on a real scenario, it could be deployed to a real vehicle and tested with a real use case.

Bibliography

[1] Apache Fountation. Apache Incubator Etch, 2012. `http://incubator.apache.org/etch/` accessed on 19/Jul/2012.

[2] Apple Inc. Apple Inc., 2012. `http://www.apple.com/` acessed on 07/Aug/2012.

[3] Bell, D., and La Padula, L. Secure computer system: Unified exposition and multics interpretation. Tech. rep., DTIC Document, 1976.

[4] Bouard, A. Software development for a security middleware in car. Master's thesis, EURECOM, 2010.

[5] Bouard, A., Schanda, J., Herrscher, D., and Eckert, C. Automotive Proxy-based Security Architecture for CE Device Integration. In *Proceedings of the 5th International Conference on Mobile Wireless Middleware, Operating Systems, and Applications* (2012).

[6] Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F., and Kohno, T. Comprehensive experimental analyses of automotive attack surfaces. *Proceedings of the 2011 Usenix Security* (2011).

[7] Denning, D. A lattice model of secure information flow. *Communications of the ACM 19*, 5 (1976), 236–243.

[8] Efstathopoulos, P., Krohn, M., VanDeBogart, S., Frey, C., Ziegler, D., Kohler, E., Mazieres, D., Kaashoek, F., and Morris, R. Labels and event processes in the asbestos operating system. *ACM SIGOPS Operating Systems Review 39*, 5 (2005), 17–30.

[9] Enck, W., Gilbert, P., Chun, B., Cox, L., Jung, J., McDaniel, P., and Sheth, A. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation* (2010), USENIX Association, pp. 1–6.

[10] eNOVA. SEIS - Sicherheit in Eingebetteten IP-basierten Systemen, 2012. `http://strategiekreis-elektromobilitaet.de/public/projekte/seis/` accessed on 19/Jul/2012.

[11] Gollmann, Dieter. *Computer Security.* John Wiley & Sons, Chichester, 2004. ISBN:0-471-97844-2.

[12] Gryc, Andy. Making Sense of the Smartphone-Vehicle Cacophony, 2011.

[13] Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., et al. Experimental security analysis of a modern automobile. In *Security and Privacy (SP), 2010 IEEE Symposium on* (2010), IEEE, pp. 447–462.

[14] Krohn, M., Yip, A., Brodsky, M., Cliffer, N., Kaashoek, M., Kohler, E., and Morris, R. Information flow control for standard os abstractions. In *ACM SIGOPS Operating Systems Review* (2007), vol. 41, ACM, pp. 321–334.

[15] Mattila, E. Isolation and networking aspects of executing untrusted applications in the automotive environment. Master's thesis, EURECOM, 2012.

[16] Migliavacca, M., Papagiannis, I., Eyers, D., Shand, B., Bacon, J., and Pietzuch, P. Defcon: high-performance event processing with information security. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference* (2010), USENIX Association, pp. 1–1.

[17] Myers, A. Jflow: Practical mostly-static information flow control. In *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (1999), ACM, pp. 228–241.

[18] Myers, A., and Liskov, B. A decentralized model for information flow control. In *ACM SIGOPS Operating Systems Review* (1997), vol. 31, ACM, pp. 129–142.

[19] Myers, A., and Liskov, B. Protecting privacy using the decentralized label model. *ACM Transactions on Software Engineering and Methodology (TOSEM) 9*, 4 (2000), 410–442.

[20] Oracle Corporation. MySQL Relational Database Management System, 2012. `http://www.mysql.com/` accessed on 23/Jul/2012.

[21] Roy, I., Porter, D., Bond, M., McKinley, K., and Witchel, E. *Laminar: practical fine-grained decentralized information flow control*, vol. 44. ACM, 2009.

[22] Weckemann, K., Lim, H., and Herrscher, D. Practical experiences on a communication middleware for ip-based in-car networks. In *Proceedings of the 5th International Conference on Communication System Software and Middleware* (2011), ACM, p. 12.

[23] Wolf, M., Weimerskirch, A., and Paar, C. Security in automotive bus systems. In *Workshop on Embedded IT-Security in Cars* (2004), pp. 11–12.

[24] Xen.org. Xen Hypervisor, 2012. `http://www.xen.org/` accessed on 25/Jul/2012.

[25] Zeldovich, N., Boyd-Wickizer, S., Kohler, E., and Mazières, D. Making information flow explicit in histar. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation* (2006), pp. 19–19.

[26] Zeldovich, N., Boyd-Wickizer, S., and Mazieres, D. Securing distributed systems with information flow control. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation* (2008), USENIX Association, pp. 293–308.

List of Figures

Database SQL Code

The SQL code used to create the database that was used in this thesis.

Database Creation

```
mysql> CREATE DATABASE drivinglog_sec;
mysql> GRANT ALL ON drivinglog_sec.*
       TO dlmanager@'%'
       IDENTIFIED BY 'bmwetch';
mysql> FLUSH privileges;
mysql> USE drivinglog_sec;
mysql> CREATE TABLE Profiles (
       id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
       name VARCHAR(50),
       username VARCHAR(50),
       password VARCHAR(40)
       );
mysql> CREATE TABLE Info(
       id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
       coordX FLOAT,
       coordY FLOAT,
       odometer INT,
       speed INT,
       roadType VARCHAR(50),
       waterLevel INT,
       oilLevel INT,
       gasLevel INT,
       timestamp TIMESTAMP DEFAULT NOW(),
       secrecyTag VARCHAR(100),
       integrityTag VARCHAR(100)
       );
```

Database Profiles

```
mysql> INSERT INTO Profiles(name, username)
       VALUES ('Default', 'default');
mysql> INSERT INTO Profiles(name, username, password)
       VALUES ('Rental Company', 'company', SHA('company'));
mysql> INSERT INTO Profiles(name, username, password)
       VALUES ('Car', 'car', SHA('car'));
mysql> INSERT INTO Profiles(name, username, password)
       VALUES ('Daniel', 'daniel', SHA('daniel'));
```

Driving Log Server

This code is part of the server's code and includes retrieving the label from the Proxy, retrieving data from the database, enforcing the information flow and returning the data.

```
public class ImplDrivingLogServer extends BaseDrivingLogServer
{
  private static String activeDriver = null;
  private static Resources res;
  private SecurityEnvironment secInstance;
  private Connection con;
  private String VM_IP;
  public ImplDrivingLogServer( RemoteDrivingLogClient client,
                               Resources res,
                               Connection con,
                               String vm_ip )
  {
    // Initialization of the variables
    this.client = client;
    this.res = res;
    this.con = con;
    this.VM_IP = vm_ip;
    try {
      // secInstance contains security information from Proxy
      Resources proxyRes =
        (SecurityEnvironment) client.getResources();
      secInstance =
        proxyRes.get(SecurityEnvironment.SECURITY_ENVIRONMENT);
    } catch (Exception e) {
      ...
    }
  }
```

```
private String getCallerSecrecy()
{
  // Retrieve the confidentiality set from the caller
  // from the Proxy
  return secInstance.getCurrentTag().get_difc_secrecy();
}

private String getCallerIntegrity()
{
  // Retrieve the integrity set from the caller
  // from the Proxy
  return secInstance.getCurrentTag().get_difc_integrity();
}

private void prepareProxyLabel(String secrecySet,
                               String integritySet)
{
  // This function prepares the security environment
  // for the Proxy
  String[] IFC = new String[2];
  IFC[0]=secrecySet;
  IFC[1]=integritySet;
  client.getResources().put("SECURITY_ENVIRONMENT.IFC",IFC);
}

@Override
public Map requestData(String driverName)
      throws OperationException, SecurityException
{
  // Retrieve label of who's requesting data
  String callerSecrecy = getCallerSecrecy();
  String callerIntegrity = getCallerIntegrity();
  // Retrieve all tags
  ArrayList callerSecrecyList =
    new ArrayList(Arrays.asList(callerSecrecy.split(",")));
  ArrayList callerIntegrityList =
    new ArrayList(Arrays.asList(callerIntegrity.split(",")));
```

```
String currentLabel = "";
// Apply some modifications to the tags, such as
// replace 'company' by 'car' and extend tags with
// .public and .private
for (int i=0; i<callerSecrecyList.size(); i++) {
  currentLabel = (String) callerSecrecyList.get(i);
  if (currentLabel.matches("(?i)^company.*")) {
    // Applying policy that car data belongs to company
    currentLabel=currentLabel.replaceAll("company", "car");
    callerSecrecyList.set(i, currentLabel);
  }
  if (!(currentLabel.matches("(?i).*\\..*"))) {
    // Extending the tags with .public and .private
    // for the cases where it's not specified
    callerSecrecyList.set(i, currentLabel+".public");
    callerSecrecyList.add(i+1, currentLabel+".private");
  }
}

// Tmp variables init
PreparedStatement pst = null;
Map data = new HashMap();
List ids = new LinkedList();
List coordXs = new LinkedList();
List coordYs = new LinkedList();
List odometers = new LinkedList();
List speeds = new LinkedList();
List roadTypes = new LinkedList();
List waterLevels = new LinkedList();
List oilLevels = new LinkedList();
List gasLevels = new LinkedList();
List timestamps = new LinkedList();
Integer id = 0;
Float coordX = 0.0f;
Float coordY = 0.0f;
Integer odometer = 0;
Integer speed = 0;
String roadType = "";
Integer waterLevel = 0;
```

```
Integer oilLevel = 0;
Integer gasLevel = 0;
java.sql.Timestamp timestamp = null;
String secrecyTag = "";
String integrityTag = "";
ArrayList secrecyTagList = null;
ArrayList integrityTagList = null;

try {
if (driverName.equals("company")) {
  // Requesting company data means requesting car data
  driverName = driverName.replaceAll("company", "car");
}

// Retrieve all data from DB
// associated to driverName in any way
pst = con.prepareStatement(
    "SELECT * FROM Info WHERE secrecyTag LIKE ?;");
pst.setString(1, "%"+driverName+"%");
ResultSet queryResult = pst.executeQuery();

// This counter limits the returned list to 16 entries
// to overcome size limit of Packetizer (Etch)
int count = 0;

while (queryResult.next())
{
  // Break rule
  if (count > 15) break;

  // Retrieve the entry
  id = queryResult.getInt(1);
  coordX = queryResult.getFloat(2);
  coordY = queryResult.getFloat(3);
  odometer = queryResult.getInt(4);
  speed = queryResult.getInt(5);
  roadType = queryResult.getString(6);
  waterLevel = queryResult.getInt(7);
  oilLevel = queryResult.getInt(8);
```

```
      gasLevel = queryResult.getInt(9);
      timestamp = queryResult.getTimestamp(10);
      secrecyTag = queryResult.getString(11);
      integrityTag = queryResult.getString(12);
      secrecyTagList =
        new ArrayList(Arrays.asList(secrecyTag.split(",")));
      integrityTagList =
        new ArrayList(Arrays.asList(integrityTag.split(",")));

      // For each retrieved entry, check if the secrecy labels
      // are 'lower or equal' than the caller's secrecy labels
      if (callerSecrecyList.containsAll(secrecyTagList)
        && integrityTagList.containsAll(callerIntegrityList)){
        // This means that the data secrecy labels are
        // a subset of the caller secrecy labels and that
        // the caller integrity labels are a subset
        // of the data integrity labels.
        // For that reason, data is allowed to flow
        // from the database to the caller

        // Add to list to be retrieved
        ids.add(id);
        coordXs.add(coordX);
        coordYs.add(coordY);
        odometers.add(odometer);
        speeds.add(speed);
        roadTypes.add(roadType);
        waterLevels.add(waterLevel);
        oilLevels.add(oilLevel);
        gasLevels.add(gasLevel);
        timestamps.add(timestamp.toString());
        count++;
      }
```

```
      // After retrieving data and enforcing the flow,
      // format return data to the HashMap structure
      data.put("ids",ids);
      data.put("coordXs", coordXs);
      data.put("coordYs", coordYs);
      data.put("odometers", odometers);
      data.put("speeds", speeds);
      data.put("roadTypes", roadTypes);
      data.put("waterLevels", waterLevels);
      data.put("oilLevels", oilLevels);
      data.put("gasLevels", gasLevels);
      data.put("timestamps", timestamps);
    } catch (SQLException sqle) {
      System.out.println(
        "Error while retrieving data from the database.");
      throw new OperationException(
        "Error with the database on the server side.");
    }

    // Prepare the label for the Proxy
    prepareProxyLabel(callerSecrecy, callerIntegrity);

    return data;
    }
    ...
}
```