Evolutions et dé-évolutions de la sécurité des systèmes multimédia automobiles

François Pollet et Nicolas Massaviol
francois.pollet@thalesgroup.com
nicolas-massaviol@toucan-system.com

Thales Communication and Security, Toucan System

Résumé La sécurité des systèmes embarqués dans les véhicules est, depuis quelques années, un thème de plus en plus abordé lors des conférences en sécurité informatique « généralistes », les constructeurs souhaitant ajouter de plus en plus de fonctionnalités : multimédia natif ou déport des smartphones des utilisateurs, mais aussi *« platooning »*, *« cloud car sharing »* et autres automatisations du véhicule. Nous avons donc souhaité revenir sur les différents travaux effectués chez plusieurs constructeurs au cours des dernières années, en s'appuyant sur l'étude de produits de plusieurs constructeurs, et sur les problématiques de sécurité constatées.

1 Introduction

Nous avons pu, depuis plusieurs années, intervenir auprès de plusieurs constructeurs pour effectuer un certain nombre de missions de conseil ou d'audit de leurs équipements embarqués. Cet accompagnement nous a permis de suivre l'évolution des produits multimédia entre plusieurs générations (BMUL v0, v1 et v2 pour le premier constructeur), mais aussi d'appréhender, auprès d'autres constructeurs, des approches radicalement différentes autour des mêmes problématiques.

Nos travaux nous ont aussi permis d'accompagner certains constructeurs sur les évolutions à venir et sur les challenges que ces modifications apportent (par exemple l'automatisation toujours plus poussée des véhicules).

Les résultats présentés ont été trouvés lors de tests effectués chez les constructeurs respectifs, à leur demande et avec leur appui. Toutes les vulnérabilités présentées sont soit corrigées dans les systèmes en production soit en cours de correction dans les systèmes en développement.

Nous commencerons cet article par une description sommaire des architectures multimédia rencontrées. Dans une seconde partie, nous allons décrire l'approche suivie pour effectuer l'analyse en boîte noire

d'un système multimédia embarqué, en nous appuyant dès que possible sur des exemples de vulnérabilités ou défauts d'architecture rencontrés. Enfin nous reviendrons sur les grandes problématiques rencontrées par les constructeurs et l'évolution de leurs réponses à ces problématiques, avant de conclure sur les évolutions à venir et leurs impacts sur la sécurité des véhicules connectés.

2 Terminologie et principes d'architecture dans l'automobile

2.1 Les ECU

Un système automobile est constitué d'un nombre toujours plus important de calculateurs. Ces calculateurs, ou ECUs (Electronic Control Units), vont assurer le bon fonctionnement des divers systèmes du véhicule (injection, régime moteur, fonctions de sécurité, systèmes de freinage et de suspension, direction, etc.).

2.2 Les réseaux CAN

Les bus CAN, encore très répandus dans le domaine automobile, servent de médium principal de communication entre les multiples calculateurs distribués dans le véhicule. Au contraire d'une approche point à point, l'ensemble des ECUs amenés à discuter entre eux sont reliés à un même câble. En cas de collision (plusieurs ECUs envoyant simultanément une trame), un champ d'arbitrage permet de déterminer la priorité des différents messages. Le lien physique est donc très similaire à ethernet dans son fonctionnement.

Très rudimentaire, et conçu pour être utilisé par des calculateurs aux performances très restreintes, les protocoles reposant sur le CAN ne proposent aucune authentification des ECUs entre eux, ni aucune protection contre un DOS (par envoi continu de messages très prioritaires). Divers projets et travaux académiques ont été publiés ces dernières années [3][6], pour tenter de répondre à ces problématiques.

Les systèmes étant, par essence, propriétaires, chaque constructeur utilise sa propre messagerie (correspondance entre identifiant de trame CAN et signification de la trame, contenu des différentes trames, etc.) et sa propre nomenclature. Nous avons choisi, lors de nos audits, d'appeler :

— CAN-V ou CAN « véhicule » : le CAN principal reliant les ECUs critiques (comme le moteur ou les freins). Comme a pu le démontrer M.Miller, un attaquant pouvant émettre sur ce CAN des trames arbitraires est actuellement en mesure de contrôler complètement les véhicules étudiés.
— CAN-M ou CAN « multimédia » : le CAN reliant les divers équipements multimédias entre eux. On y trouve typiquement les boitiers du système multimédia, mais aussi des options comme la caméra de recul.
— Il existe généralement plusieurs autres CAN dans les véhicules suivant leur gamme. Ces bus peuvent être assimilés au CAN-V car ils hébergent aussi des fonctions critiques. Le reste du document est centré sur les deux précédents CAN.

2.3 Composants du système multimédia

Radio/Multimédia Immédiatement accessible à l'utilisateur, ce boitier inclut une interface, souvent tactile, et un ensemble d'interfaces permettant à l'utilisateur d'y brancher ses périphériques (USB, wifi, Bluetooth, etc.). Pour des raisons commerciales, les constructeurs choisissent généralement de lier à cet équipement un environnement applicatif dédié (GPS, lecteurs multimédia, etc.) mais aussi un magasin applicatif fermé.

De plus en plus, on retrouve sur cet équipement des fonctionnalités de déport d'écran pour les équipements mobiles de l'utilisateur (CarPlay et AndroidAuto principalement).

Boitier télématique Boitier chargé de la communication GSM et Data, on y retrouve souvent, en plus de la connexion avec le SI du constructeur, des fonctionnalités de secours (en cas d'accident, un micro retransmet au centre de secours le son de l'habitacle). Il permet généralement d'offrir une connectivité au boitier multimédia lorsque celui-ci n'est pas appairé avec un smartphone.

Gateway inter-CAN Boitier central assurant la communication et le transfert des trames entre plusieurs CAN.

2.4 Exemples d'architectures

Lors de nos différents audits, nous avons pu observer deux types d'architectures bien distinctes. Sur les systèmes du premier constructeur,

on trouve, pour le moment, une architecture à plat. D'autres constructeurs utilisant eux une architecture segmentée.

La figure 1 présente une architecture à plat. Dans ce type de système, les différents boitiers composant le système multimédia sont directement reliés au CAN multimédia voire au CAN véhicule. Sur chaque boitier, le microcontrôleur chargé d'émettre les trames CAN ne doit autoriser qu'un sous-ensemble défini de trames CAN.

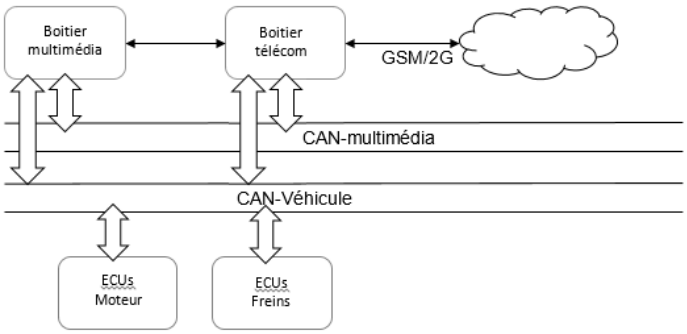

Figure 1. Architecture à plat

Au contraire, d'autres constructeurs ont choisi d'utiliser un boitier central assurant, entre autres, un rôle de proxy inter CAN (présenté par la figure 2). Ce concentrateur, présent en grande partie pour permettre la communication entre des CAN n'utilisant pas la même fréquence, ne doit laisser passer qu'un ensemble de trames légitimes entre les bus multimédia et véhicule afin d'assurer le bon cloisonnement des différents CAN.

3 Méthodologie d'analyse

L'analyse d'un système embarqué dans une automobile se doit de répondre à certaines préoccupations spécifiques au constructeur et à l'environnement. L'objectif des analyses effectuées étant, au-delà de la compromission de l'équipement multimédia, d'obtenir la possibilité d'émettre des trames arbitraires sur le CAN véhicule. À l'exception d'un bruit soudain, au maximum de la puissance sonore et que l'utilisateur ne peut pas arrêter, la compromission de l'équipement multimédia, si elle est contenue, se limite à des effets d'impacts moindres (pertes d'images, de confidentialité

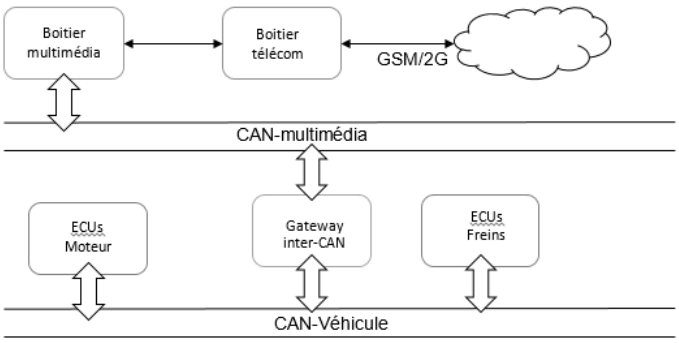

Figure 2. Architecture segmentée

des données personnelles ou de revenus générés par la vente de licences applicatives). Au contraire, la compromission du CAN véhicule - et par conséquent des ECUs reliés à ce CAN - peut mettre en danger l'intégrité du véhicule et de ses passagers.

Il faut aussi noter qu'en règle générale, les constructeurs minimisent la criticité des attaques purement matérielles, ou ne pouvant être effectuées sans accès à l'habitacle du véhicule. En effet, la prise OBD (On Board Diagnostics), présente dans l'habitacle et facilement accessible, permet un accès direct et non filtré aux CAN véhicule et multimédia.

Nos analyses sont effectuées parallèlement sur le boitier multimédia et la gateway entre les différents CAN du véhicule, ainsi que sur le boitier télématique. Elles visent, pour le boitier multimédia :

- À récupérer une image disque du ou des systèmes de fichiers utilisés (via mise à jour ou dump mémoire)
- À obtenir une exécution de code arbitraire sur le boitier. En général les canaux sont :
  - Un défaut dans la chaîne de boot
  - Les fichiers et le processus de mise à jour
  - La présence d'une interface de debug ou de programmation, de console série, de menus cachés
  - La présence de vulnérabilités systèmes ou applicatives
  - L'utilisation d'applications lorsque le boitier est ouvert aux développeurs tiers
- À élever ses privilèges (si nécessaire)

— Émettre des trames CAN arbitraires, soit par défaut architectural soit en flashant le composant de communication avec le CAN
— À détourner les fonctions légitimes.

Pour le boitier de communication inter CAN (gateway) :
— À lister les interfaces disponibles et isoler les interfaces de programmation ou de debug
— À identifier et contourner les règles de « pare-feu » CAN et les messages pouvant circuler entre les différents CAN
— À obtenir une exécution de code arbitraire sur le boitier

3.1 Boitier multimédia – prise d'information

Reconnaissance du hardware

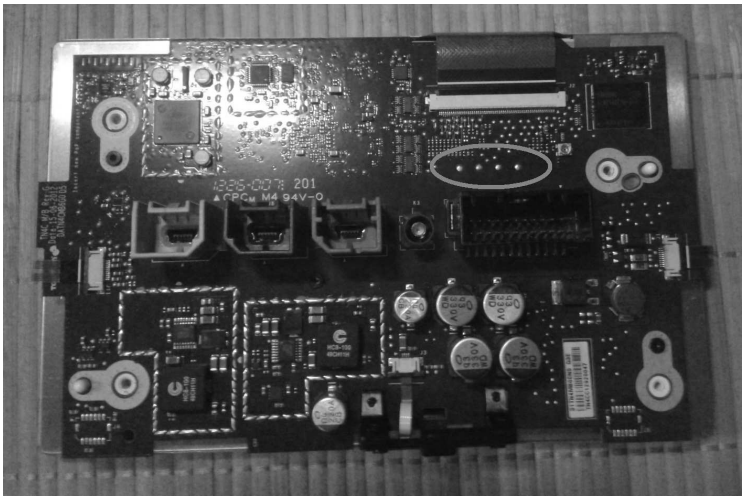

Figure 3. Boitier BMUL-v1

Sur le PCB (figure 3) , on retrouve les interfaces externes suivantes :
— Trois ports USB dont deux sont reliés à la façade (lecteur de carte SD et connecteur USB), le troisième port permet, sur une pièce en configuration de test, d'accéder à un canal ADB.
— Un connecteur coaxial
— Un connecteur 2pins sur lequel est relié l'alimentation, la caméra de recul et les bus CAN *multimédia*

On note aussi la présence très visible de quatre points de test contenant un UART (115200 bauds) utilisé comme console non interactive par l'OS

et un connecteur, qui s'il est mis à la masse, provoque l'ouverture d'une image de diagnostic.

Enfin, on note la présence d'un SOC *Texas Instruments OMAP3630-HS (OMAP)* et d'un microcontrôleur ARM7 *Texas Instrument type TMS470* assurant l'accès au CAN. La mise à jour de son firmware est possible (pour des raisons fonctionnelles) depuis l'OMAP.

UART, JTAG

Directement accessible à l'arrière du boitier, ou via des points de test sur le PCB, les ports séries nous servent de premier point de prise d'information.

BMUL-v1 : Comme mentionné plus haut, sur le PCB du BMUL-v1 est présente une sortie série utilisée comme console non interactive par l'OS. L'analyse de la sortie console selon différents scénarios (chargement nominal, mise à jour, comportement suite à une mise à jour corrompue) nous offre une première image de la chaîne de boot :

- X-loader chargeant et vérifiant U-boot depuis une mémoire flash
- U-boot charge et vérifie différentes images selon l'état du boitier :
  - Image de diagnostic si le pad sys_diag est mis à la masse
  - Image de mise à jour si nécessaire
  - Image de secours en cas de défaut
  - Image contenant le kernel
- Le boot continue ensuite depuis la zImage, via un kernel minimaliste chargé de monter la partition système (on apprend au passage que cette dernière est signée et vérifiée par le driver kernel « signedloop »)
- Enfin on observe le lancement d'un kernel linux, puis du système Android.

L'image de diagnostic permet entre autres de lire et écrire l'EEPROM, de récupérer, via le CAN, les valeurs de divers capteurs du véhicule, de configurer et de tester l'ensemble des systèmes multimédia (accès GPRS, Bluetooth, etc.) ou de lire les différentes mémoires flash. Il est aussi doté d'une fonctionnalité d'autodestruction manuelle, permettant de supprimer définitivement cette image de la pièce.

L'accès, de manière logicielle, en lecture et écriture aux différentes mémoires permet, à un attaquant patient, de s'affranchir de la phase potentiellement destructrice du dump physique de ces composants. L'extraction via la console série est en effet très lente mais n'implique pas d'opération de soudage/dessoudage qui peuvent endommager les composants.

Figure 4. image de diagnostic - écran et sortie console

BMUL-v2 : Sur cette seconde version, on trouve un connecteur de debug de 24 pins, contenant une console série et un JTAG vers l'IMX6 (processeur dédié au système multimédia), ainsi qu'une interface de programmation vers les microcontrôleurs dédiés au CAN (composants Renesas et DIRANA3).

Lors de l'audit le port JTAG était actif sur l'IMX6, et ce que le boitier soit dans une configuration sécurisée ou de debug, permettant un contrôle total sur le processeur.

Second constructeur : Enfin, nous avons découvert sur un autre système, des consoles séries en mode interactif présentant une mire de login. Une telle vulnérabilité appelle bien évidement à se concentrer sur la récupération des fichiers *passwd* ou *shadow*, soit via les fichiers de mise à jour, soit via un dump mémoire.

Hardware – dump

EEPROM

Les mémoires EEPROM (Erasable Programmable Read Only Memory) sont des composants aisément identifiables et très répandus, dont le contenu (souvent de la configuration) est accessible et modifiable sans soudure (ici à l'aide d'une pince SOIC8).

BMUL-v1 : Sur le BMUL-v1 on y trouve un Flattened Device Tree (format de stockages de données) contenant les paramètres de l'écran, un flag secure/unsecure, l'adresse MAC de l'interface Bluetooth ainsi que les device-id USB. On constate d'ailleurs, lors d'une tentative de passer une pièce du mode sécurisé au mode non sécurisé - via modification de l'EEPROM - que ce FDT est signé. Ce dernier est copié à

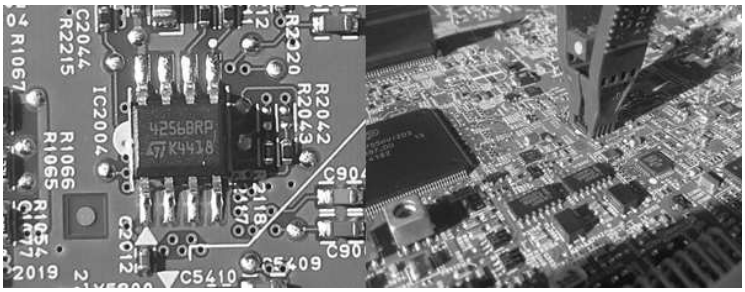

Figure 5. Dump d'une EEPROM

divers emplacements en RAM, au début de la NOR, en fin de NAND et enfin sur une carte SD, lorsque celle-ci est insérée, dans le dossier /device/NUMEROSERIE/factory.dat.

BMUL-v2 : Sur la seconde version de BMUL, on trouve une EEPROM reliée au composant MICOM. On y trouve des hashs SHA1 des différentes images système, un identifiant ECU, le VIN (Vehicle Identification Number) et le HUID. Son contenu n'étant pas signé il est possible à un attaquant de modifier ces données en une petite vingtaine de minutes, démontage des pièces inclus.

eMMC

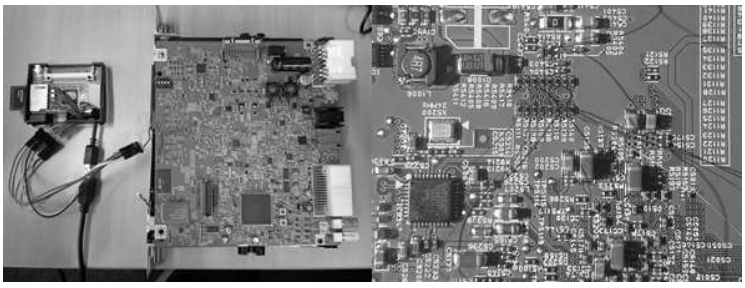

Figure 6. Dump d'une eMMC à l'aide d'un lecteur de carte SD

BMUL-v2 : Le protocole MMC 4.41 (JESD8-A441) est très proche du protocole Secure Digital, ce qui rend possible l'utilisation d'un simple lecteur de carte SD pour accéder au contenu de l'eMMC. La figure 6 présente les modifications apportées au PCB du BMUL-v2 afin d'accéder à l'eMMC principale.

Cette eMMC est séparée en trois espaces : deux partitions de boot de 16MB chacune ainsi qu'un espace linéaire de 32GB contenant les diverses partitions système et Android.

NAND

BMUL-v1 : Sur le boitier du BMUL-v1, la NAND contient une image du kernel et du système de fichiers. Il est possible de dumper la NAND après l'avoir dessoudée en utilisant un simple Arduino ou Teensy. Cette procédure est cependant sensible à de nombreuses corruptions par bit-flip, et nécessite pour certains composants une bonne compréhension des documents de références (plan d'adressage spécifique, insertion de blocs de contrôle en lieu et place de donnée, etc.).

Pour lisser au maximum les corruptions générées, nous choisissons en général d'effectuer une dizaine de dumps, puis de construire une méta-image en choisissant pour chaque octet, l'octet le plus représenté dans le corpus d'images.

Second constructeur : Sur ce boitier, nous avons été confrontés à des NAND non formatées, sur lesquelles était présente une couche de séparation en bloc UBI, puis un système de fichier UBIFS. Les outils disponibles ne permettant pas de charger une image UBI corrompue, il a été nécessaire d'écrire divers outils en Python permettant de reconstruire et d'extraire les fichiers présents dans la NAND.

L'extraction des différentes partitions a permis de récupérer plusieurs versions du fichier *passwd* (le fichier *shadow* n'étant pas utilisé par le système). Ce fichier contenait, lors de l'audit, un compte alternatif pour root, accessible sans mot de passe, ce qui permet une prise de contrôle complète, via la console série précédemment citée, de l'équipement.

Fichiers de mise à jour

BMUL-v1 : Sources d'informations privilégiée, car aisément accessibles, les fichiers de mise à jour sont distribués, pour le système BMUL-v1, sur des cartes SD contenant des fichiers *REDACTED*.00X d'une taille maximale de 2 Go chacun. Ces derniers forment le système de fichiers de mise à jour et de données de cartographie. À noter que des outils permettant d'accéder aux données sont librement téléchargeables.

BMUL-v2 : Sur la seconde version du système BMUL, les images de mise à jour sont contenues dans les fichiers mm2014_upgrade.*XX*u ou mm2014_factory.*XX*u. Ces fichiers, dont le format est propriétaire, sont suffisamment simples pour pouvoir être manipulés, n'étant ni chiffrés ni

offusqués. Pour exemple, la partition système peut être montée directement depuis une image .*XX*u, son offset pouvant être retrouvé grâce à la présence du marqueur « SYST » 24 octets avant le début de la partition.

Ces deux formats n'étant ni chiffrés ni protégés, il a été possible d'utiliser les données extraites des fichiers de mise à jour pour commencer l'audit de l'OS multimédia.

Second constructeur : Sur ce système, les fichiers de mise à jour étaient chiffrés (SMIME RSA2048+AES256-CBC) et signés (RSA2048), obligeant un attaquant à passer par le dump des différentes mémoires (en clair pour des raisons de performance) pour progresser dans son étude.

3.2 Boitier multimédia – exécution de code

Fichiers de mise à jour

BMUL-v1 : Sur les systèmes BMUL de première génération, lors de l'insertion de carte SD, le kernel principal détecte la présence d'une carte de mise à jour, change l'état du flip-flop puis déclenche un reset, ce qui permet de charger l'uImage de mise à jour. Cette dernière est chargée de parcourir les périphériques présents à la recherche d'un fichier ttresc.img, avant de le monter et de lancer la zImage supposée s'y trouver ; la signature de l'image étant vérifiée avant l'appel à kexec. Les fichiers *.img.gz présents à la racine du périphérique étant alors flashés sans vérifications supplémentaires.

BMUL-v2 : Pour la seconde version BMUL-v2, le contenu des fichiers .*XX*u est utilisé par un binaire /bin/recovery qui copie les différents fichiers et partitions à leur emplacement respectif. Toujours sans plus de vérification.

Dans ces deux cas, la sécurité du boitier multimédia repose uniquement sur la robustesse de sa chaîne de boot, chargée de vérifier l'intégrité et la provenance des différents composants. Sur la première version du système BMUL, il est toutefois possible de mettre à jour sans vérification le contrôleur en charge de la communication avec le CAN multimédia, ce composant ne faisant pas partie de la chaîne de boot.

Second constructeur : Lors de l'audit de ce boitier, les mises à jour chiffrées et signées pouvaient :

— Ne pas être chiffrées
— Ne pas être signées, la fonction chargée de vérifier la signature du fichier contenant les hashs des différents composants retournant

une valeur interprétée plus tard comme : « mise à jour signée » lorsque ce fichier n'est pas présent.

Il était donc possible pour un attaquant de déployer un système de fichiers arbitraire (ce dernier n'étant pas signé au contraire des systèmes BMUL), ainsi que des packages applicatifs mais aussi de mettre à jour le composant en charge de la communication avec le CAN.

À noter que pendant la mise à jour, le système permettait la connexion, via le port série, en utilisant le compte root (sans mot de passe).

chaîne de boot

La connaissance des images systèmes (extraites des fichiers de mise à jour ou récupérées via le dump des mémoires présentes) permet, en s'aidant si disponible de la console de debug, de reconstruire les différentes chaînes de boot.

BMUL-v1 : L'analyse des différents composants de la chaîne de boot du système BMUL n'a pu isoler aucune vulnérabilité ou défaut de configuration. Chaque composant vérifiant, via des appels à la TrustZone ARM, la signature (DSA-2048-SHA1) du composant suivant.

Le choix, par U-boot, de l'image lancée (image de diagnostic, image de mise à jour ou image système « normale ») dépend :

— D'une bascule interne déclenchant la mise à jour. Cette bascule pouvant être initialisée par le système nominal lorsqu'un média de mise à jour est inséré.

— De la mise à la masse de la broche sys_diag, déclenchant le chargement de l'image de diagnostic

L'image nominale, un système Android, utilise un patch kernel destiné à assurer l'intégrité de la partition /system. Il utilise un principe similaire à dm_verity intégré dans le kernel Linux depuis la version 3.4. Ce patch impose, pour tout système de fichiers monté via un loop device deux états :

— Un état signé, et par extension non inscriptible.

— Un état inscriptible, mais avec les options de montage nosuid, noexec et nodev (restriction des droits d'exécutions, des flags suid et sgid et de l'accès aux devices).

Les partitions protégées par signedloop sont structurées de la sorte :

La première section contient les données, non chiffrées, alignées sur une taille de bloc de 4k.

La seconde section contient les hash SHA1 de chacun des blocs de données. De même, les blocs de niveau 1 contiennent les hashs des blocs

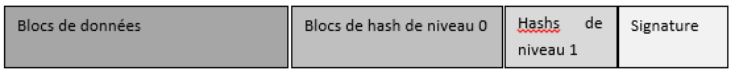

Figure 7. Ségmentation d'une partition signée

de niveau 0. Enfin, le dernier bloc contient la signature DSA-2048+SHA1 de la section de hashs de niveau 1.

Au montage, seule la signature est vérifiée, la vérification des blocs de données étant effectuée lors du premier accès à celui-ci.

BMUL-v2 : L'implémentation, sur les systèmes BMUL-v2, d'une chaîne de boot sécurisée et du patch signedloop dénote une mauvaise compréhension du fournisseur des problématiques associées :

- Sur les premières versions auditées, les eFuses (fusibles reprogrammables permettant de configurer le comportement d'un processeur) gérant les paramètres de secure boot n'avaient pas été protégés contre la réécriture, rendant l'intégralité du dispositif inutile.
- La vérification, par la bootrom (code lancé au démarrage du composant, situé sur une mémoire interne), de la signature d'U-Boot était effectuée sur une plage mémoire incorrecte. Il était donc possible de modifier, sur l'eMMC, le code d'U-boot afin de permettre l'exécution d'une image arbitraire. Il était par exemple possible de patcher la fonction BL_authimg (fonction chargée de vérifier la signature du kernel).

Enfin la vérification, via un portage du patch signedloop, de la partition system s'effectue en deux temps :

- La partition system est montée une première fois sans être vérifiée.
- Le binaire /system/bin/busybox est ensuite utilisé pour remonter la partition /system via un loop device ; et donc pour en vérifier l'intégrité.

```
on fs
    mount ext4 /dev/block/mmcblk0p1 /system wait ro
#[20140523 meewha.yun  add for secure boot ] [Start]
    exec /system/bin/busybox losetup /dev/block/loop0 /dev/block/
        mmcblk0p1
    write /sys/class/block/loop0/loop/signedloop 1
    exec /system/bin/busybox mount -t ext4 /dev/block/loop0 /system
#[20140523 meewha.yun  add for secure boot ] [End]
```

Listing 1. Fragment du script init.XXX.rc

Il est donc tout à fait possible de modifier le binaire busybox, en le remplaçant par exemple par un script, pour empêcher la vérification de la partition /system.

Il faut aussi noter que, contrairement au system BMUL de premère génération, la version BMUL-v2 de signed loop ne force pas les flags *noexec*, *nosuid* et *nodev* sur les partitions non signées. Cette absence de protection supplémentaire induit une perte de sécurité non négligeable.

Second constructeur : Au contraire des systèmes BMUL, ce système n'implémente pas de vérification au démarrage du kernel et des systèmes de fichiers. Le fournisseur ayant pris comme postulat que les systèmes de fichiers ne pouvaient être modifiés en dehors du cycle de mise à jour, ce mécanisme apportait, selon lui, une vérification redondante à la signature des fichiers de mise à jour.

Il est donc possible, pour un attaquant, de modifier le contenu des partitions stockées sur la flash. Typiquement de modifier le fichier *passwd* afin de créer un compte privilégié avec lequel l'attaquant pourra se connecter à la console série. L'absence de vérification des systèmes de fichiers permet aussi à un attaquant ayant obtenu une exécution de code via une vulnérabilité logicielle de créer une backdoor.

Système multimédia

En l'absence de vulnérabilité sur la chaîne de boot, de défaut dans la procédure de mise à jour, ou de console interactive astucieusement dissimulée, il reste à l'auditeur toute la surface d'attaque offerte par le système multimédia. Malheureusement, les systèmes que nous avons pu auditer n'avaient pas, ou très peu, de fonctionnalités accessibles via les interfaces réseaux. Pas de D-BUS ouvert sur internet ici[4], et chez le premier constructeur, peu de services en écoute sur le lien entre boitiers multimédia et télécom.

Plus généralement, l'absence de fonctionnalités implémentées lors des tests diminue considérablement la surface d'attaque potentielle.

Le premier constructeur a choisi, pour ses systèmes BMUL d'utiliser une base Android, 2.2 pour la première génération et 4.0 pour la seconde génération. Toutes deux étaient, lors des audits, obsolètes et sujettes à de nombreuses vulnérabilités.

BMUL-v1 : Début 2013, on trouvait sur Android Froyo : deux vulnérabilités[1] portant sur des défauts de restriction dans Ashmem (mécanisme de partage de mémoire propre à Android), une corruption

BMUL2	CVE-2013-4710	Webviews	OK	
	CVE-2013-4787	MasterKey	OK	
BMUL	CVE-2011-1149	KillingInTheNameOf, psneuter	NOK	Pas de code natif
	CVE-2011-1823	Gingerbreak	NOK	Pas de code natif
	CVE-2011-3874	zergRush	NOK	ADB non ouvert
	CVE-2010-EASY	RageAgainstTheCage/Zimperlich	NOK	Watchdog

Figure 8. Résumé des vulnérabilitées utilisées

mémoire dans Vold, nécessitant l'utilisation de socket PF_NETLINK[9], un défaut de vérification de la valeur de retour de setuid dans ADB[8] et zygote[10] (binaire principal de DALVIK). Tous ces exploits nécessitent au préalable l'exécution de code sur le système cible.

Pour obtenir cette exécution de code, le premier constructeur ayant choisi d'ouvrir son système à des développeurs tiers contrôlés, nous avons soumis une application simple pour validation. Cette dernière allait récupérer sur internet un ensemble de classes Java avant de les charger via l'API DexClassLoader, permettant de passer outre les analyses de code avant publication sur le store interne.

Cependant, le driver signedloop restreint les partitions non signées aux droits nosuid, noexec et nodev. Il n'est donc pas possible pour notre application d'utiliser du code natif, nous restreignant aux vulnérabilités Zergrush et Zimperlich. Zergrush reposant sur ADB, et ce dernier n'étant pas activé, nous avons dû nous rabattre sur Zimperlich.

Ce dernier exploit reposait sur un défaut dans zygote, qui ne vérifiait pas la valeur de retour de l'appel setuid après la création d'un nouveau processus utilisateur. Lorsque la limite de processus pour un utilisateur était atteinte, le nouveau processus continuait à vivre en tant que root. Malheureusement, un watchdog interne reboot le boitier multimédia, pour cause de lag, bien avant que cette limite soit atteinte.

BMUL-v2 : Pour la seconde version de BMUL (Android 4.0), nous avons pu exploiter la vulnérabilité « WebView »[7] découverte entre nos deux audits et permettant l'exécution de code arbitraire à distance.

Une fois l'exécution de code atteinte (bien que de manière très sommaire sur la première version), il est naturel de chercher à réactiver ADB, si possible en le démarrant en tant que root.

BMUL-v1 : Sur la première version du BMUL, on constate, via le shell de notre application (voir figure 9), que le flag ro.secure (flag contrôlant les privilèges de la console ADB) est commodément positionné à 0 (ADB sera donc lancé avec les droits de *root*) et que le binaire ADB est toujours

présent sur le système. Il suffit donc d'envoyer un Intent pour afficher le menu de configuration (autrement inaccessible) et permettre à l'utilisateur d'obtenir un shell ADB en tant que root.

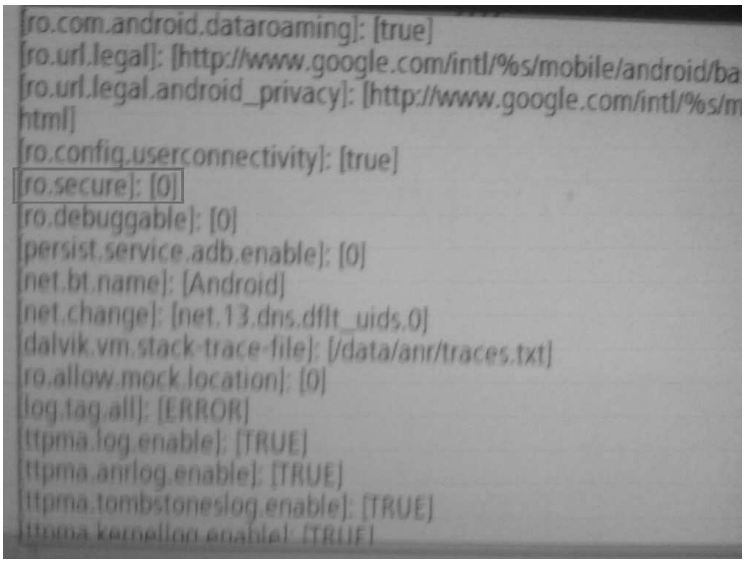

Figure 9. Console logicielle - extrait de la sortie de la commande *getprop*

BMUL-v2 : Pour la seconde version de BMUL, il est possible d'écrire dans le fichier /data/local.prop, situé sur une partition non signée, afin d'écraser la configuration du système, et donc de positionner les flags ro.secure ou ro.kernel.qemu. Au-delà de l'Intent permettant l'accès à la configuration, un menu caché est présent sur le système permettant la réactivation d'ADB. Sur certaines versions de test, l'activation de ce menu nécessitait l'insertion d'une clé USB contenant un fichier spécifique.

3.3 Boitier multimédia – après le root

Une fois en possession d'une exécution de code arbitraire, même non privilégiée, l'attaquant à la possibilité d'adopter un comportement perturbateur ou dangereux pour le véhicule. Au-delà des démonstrations purement multimédia consistant à déclencher, pour un cap de vitesse donnée, une musique ou un son strident au volume maximum sans que le conducteur ait la possibilité de la couper ou de baisser le volume, il est très souvent possible d'interagir avec le véhicule.

BMUL-v1 : Sur le premier système BMUL, divers services Android exportent des interfaces permettant de dialoguer avec le véhicule, l'ensemble des paramètres accessibles ou modifiables via l'interface étant accessible sans limitation. Le véhicule mis à notre disposition étant un modèle XXXXX et le système étant configuré pour un autre modèle, ces diverses fonctionnalités n'ont pu être testées. Il était, par exemple, potentiellement possible de modifier le cycle de charge, de diffuser du parfum via la climatisation...

Le bus CAN multimédia et le boitier télécom étaient eux, accessibles via deux devices dédiés (/dev/ACM0 pour le boitier télécom) permettant d'émettre des trames CAN pour l'un ou commandes AT arbitraires pour l'autre vers ces deux composants.

BMUL-v2 : Sur la seconde génération de système BMUL, le constructeur a choisi de filtrer à l'accès aux services de communication avec le véhicule. Seules peuvent interagir avec le CAN (via le service *iviserver*) les applications dotées d'une permission spécifique, signées et stockées dans la partition système (partition qui elle-même est théoriquement signée).

Le service *iviserver* transfère ensuite les requêtes au daemon *micomd*, ce dernier contrôlant directement la liaison série avec le contrôleur CAN via le device /dev/ttymxc3. La communication entre *iviserver* et *micomd* s'effectuant via une socket TCP en écoute locale, et le filtre d'accès étant fait par *iviserver*, il est possible, pour une application malicieuse de se connecter à *micomd* sans contrôle d'accès. L'accès à *micomd*, s'il permet d'effectuer les actions exportées par *iviserver* (mise en marche de la ventilation, rabattage des sièges...), permet également de reprogrammer le contrôleur CAN, de forcer un reset du processeur applicatif, de surveiller les messages CAN, etc.

Suite à l'audit, la socket TCP a été transformée en socket UNIX, permettant de restreindre son accès, et le contrôleur CAN n'est plus reprogrammable. La possibilité de mettre à jour du contrôleur CAN était notamment l'une des vulnérabilités majeures remontées par Chris Valasek et Charlie Miller sur Jeep[4].

Second constructeur : Sur ce système, qui implémente la norme GENIVI, la cible privilégiée est D-BUS. L'accès à ce bus permettait de mettre à jour les composants de communication avec les CAN (et donc d'émettre des trames arbitraires sur le ou les CAN connectés), de contrôler l'affichage et les applications en cours (par exemple en utilisant le navigateur intégré pour accéder à une page web) et plus généralement d'interagir

avec le véhicule dans les limites des fonctionnalités implémentées par le constructeur.

La vraie limitation post intrusion étant, justement, les limitations du système implémentées. S'il est possible de rabattre les sièges depuis le composant multimédia c'est que cette fonctionnalité a été implémentée, que les trames CAN permettant de contrôler les sièges peuvent circuler, via le boitier inter-CAN, entre CAN multimédia et CAN véhicule. Et comme certains systèmes testés n'avaient, lors des tests, quasiment aucune fonctionnalité, celles-ci devant être ajoutées ultérieurement, les possibilités post-intrusion en étaient très réduites. De nouvelles fonctionnalités devant être ajoutées lors du cycle de vie des équipements multimédia, et la sécurité n'étant que très tardivement prise en compte, la surface d'attaque disponible devrait mécaniquement augmenter.

3.4 Boitier télécom

Les problématiques et l'architecture des boitiers télécom associés aux différentes générations de système BMUL (chez le premier constructeur) ayant peu évolué, ce document n'abordera que le boitier télécom associé à la première génération du système BMUL.

Le boitier télécom du second constructeur, mis à notre disposition lors de l'étude du système multimédia, ne faisait pas partie de notre périmètre. Il ne sera donc pas abordé dans ce document.

Reconnaissance du hardware Sur ce PCB, correspondant au boitier télécom associé au système multimédia BMUL de première génération, on trouve les interfaces externes suivantes :

— Interfaces pour les bus CAN « véhicules » et « multimédia »
— Interface dédiée à la liaison USB vers le système multimédia
— Modem GPRS

On remarque aisément le connecteur JTAG 20 broches, dont le connecteur n'est pas présent sur un circuit de production. Les pistes sont toutefois présentes et alimentées électriquement, et ce, quelle que soit la version du boitier télécom. En plus des interfaces permettant de reprogrammer les différents composants présents, ce connecteur inclut une liaison série vers le SOC ARM principal conçu par Sierra Wireless.

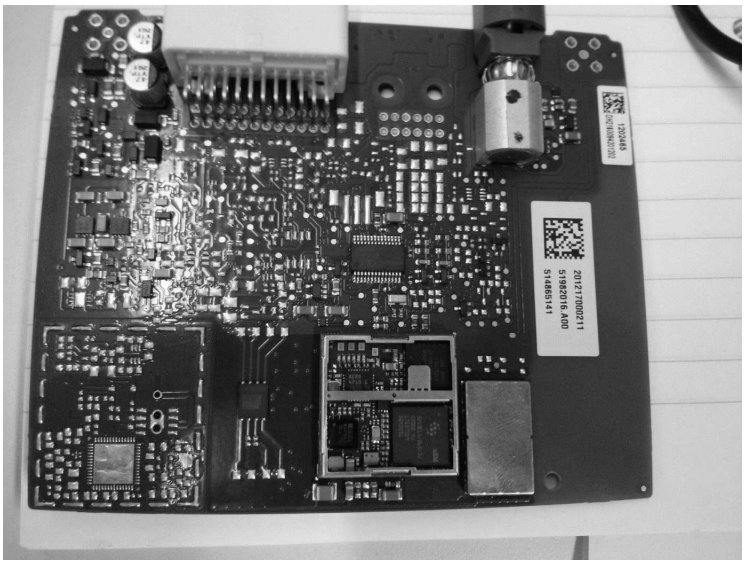

Figure 10. Boitier télécom (BMUL-v1)

Système d'exploitation OpenAT Le SOC principal du boitier télématique est conçu pour exécuter deux applications distinctes :

- Le système d'exploitation OpenAT, développé par Sierra Wireless. Cette application fournit les fonctions liées au GSM et exporte des API permettant de passer des appels (voix) ou de transférer des données.
- L'application dite télématique, qui nous a été fournie, lors de l'audit, sous forme d'un fichier binaire. Cette application peut aussi être récupérée directement depuis la RAM en détournant certaines fonctionnalités d'OpenAT

On peut accéder, en mode interactif, à OpenAT via la console série présente sur le port JTAG mentionné précédemment. Cette console interactive accepte un certain nombre de commandes « AT » répertoriées dans un document de référence disponible sur le site de Sierra Wireless . Une partie de ces commandes sont aussi accessibles depuis la liaison USB, afin d'être utilisables par le boitier multimédia.

On isole rapidement les commandes suivantes :
Mode debug : **AT+WDEBUG=1**

Cette commande permet de passer le système d'exploitation OpenAT en mode debug, il est alors possible de débugger le système d'exploita-

tion, mais aussi l'application télématique installée. Cette commande n'est utilisable que depuis la console série.

Mode développeur : **AT+WDM=1**

Cette commande permet de déverrouiller le mode « développeur », permettant l'utilisation de l'environnement de développement « Developer Studio » mis à disposition par Sierra Wireless. Cette commande est active sur le port série, mais aussi sur le port USB

Le mode développeur permet, via l'utilisation du logiciel Developer Studio l'accès en lecture et écriture au contenu de la RAM. L'accès en lecture permet de récupérer le code du système OpenAT (qui est public), mais aussi le code de l'application télématique et diverses informations privées (comme les certificats permettant d'authentifier la pièce auprès du système d'information distant).

Les fonctionnalités d'écriture en RAM n'ont pu être exploitées lors de l'audit, à cause d'un système de watchdog (système s'assurant de la disponibilité d'un composant et le redémarrant électriquement en cas de dysfonctionnement) particulièrement contraignant.

Mode Download : **AT+WDWL**

Contrairement aux autres modes présentés, ce mode download n'est pas activable directement, cette commande étant sans effet tant que le boitier télécom n'a pas été passé en mode constructeur. Pour passer en mode constructeur, une trame CAN spécifique, pouvant par exemple être extraite des outils de diagnostic utilisés par les garagistes, doit être émise sur le CAN véhicule.

Une fois passée en mode constructeur, cette commande est à nouveau disponible sur le port série et peut être utilisée pour envoyer des données au système d'exploitation (configuration ou autre). Le mode constructeur est d'ailleurs appelé « Unsafe mode » par l'application télématique comme on peut le constater sur la capture d'écran 11.

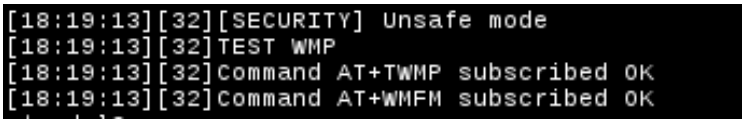

Figure 11. Passage en mode constructeur

Ce mode permet en théorie d'écraser ou de remplacer l'application télématique, cependant, et comme pour les fonctionnalités d'écriture en RAM, écraser l'application télématique va, pendant un court instant la rendre non disponible. Le watchdog va alors immédiatement faire redémarrer le système Sierra Wireless, le laissant dans un état corrompu.

Une attaque plus complète nécessiterait de reprogrammer, via le port JTAG, le composant (un PIC) hébergeant le watchdog. Nous n'avons pas pu, lors de l'audit, mener ces attaques à bien.

Application télématique L'analyse, par retro ingénierie, de l'application télématique nous a appris que cette dernière pouvait répondre aux commandes interactives, via l'espace de nom « TWMP ». L'accès à ces commandes via le port USB ou la console série n'est possible qu'en mode constructeur, et après avoir activé le mode de debug, pour l'application, via la commande AT : **at+twmp=$dbg,1**

Il est ensuite possible d'utiliser n'importe quelle commande de l'application télématique, et ce sans aucune contrainte de sécurité. Parmi ces commandes, les services « Remote diag » et « Probe » permettent d'émettre deux types de trames de diagnostic sur les CAN « véhicule » et « multimédia ».

```
[22:55:47][32]MDIAGDATACONFIG
Command WMP received: AT+TWMP=$ADDREMDIAGDATACONFIG
[22:55:47][32]   --- NEW WHITE ECU LIST ---
[22:55:47][32]Number of ECUs = 17
[22:55:47][32]RDIAG DATA: Data ID (ECU NUM)[0] = 1
[22:55:47][32]RDIAG DATA: CAN Frame ID (NA)[0] = 0x0
[22:55:47][32]RDIAG DATA: CAN Param Mask (NA)[0] = 0x0000000000000000
[22:55:47][32]RDIAG DATA: CAN Read Freq (NA)[0] = 0
[22:55:47][32]RDIAG DATA: Conversion Type[0] = 0
[22:55:47][32]RDIAG DATA: Usage Type[0] = 0
[22:55:47][32]RDIAG DATA: Data List Length[0] = 1
[22:55:47][32]RDIAG DATA: A Param (CAN ID Rx) [0] = 0x763 (1891,0000)
[22:55:47][32]RDIAG DATA: B Param (CAN ID Tx) [0] = 0x743 (1859,0000)
[22:55:47][32]RDIAG DATA: C Param (Function ID)[0] = 0x51 (81,0000)
[22:55:47][32]RDIAG DATA: D Param (CAN Net)[0] = CAN_NET_V (0,0000)
[22:55:47][32]RDIAG DATA: Unavailable value (NA) [0] = 0x00000000
[22:55:47][32]-----------------------------------
[22:55:47][32]RDIAG DATA: Data ID (ECU NUM)[1] = 2
```

Figure 12. configuration du service "remote diag"

On note aussi que l'application télématique était, lors de l'audit, vulnérable à un buffer overflow. Le 52ème caractère d'une commande AT provoquant une exception.

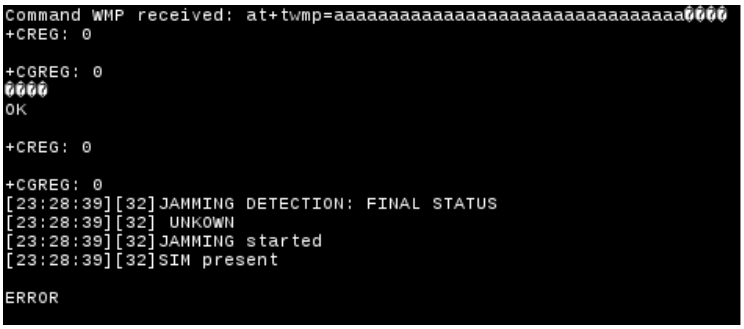

Figure 13. application télématique : buffer overflow

Cette vulnérabilité permet l'exécution de code arbitraire sur le boitier télématique. Lors de nos tentatives d'exploitation le watchdog intervenait toutefois très rapidement.

3.5 Gateway CAN

Les systèmes multimédias que nous avons pu auditer chez le premier constructeur ne possèdent pas de boitier de communication inter-CAN ; les trames pouvant être émises par les divers boitiers étant limitées par des composants situés sur les cartes, dédiés à la communication avec le CAN (par exemple le composant Texas Instrument sur les boitiers multimédia BMUL-v1).

Le système du second constructeur est lui architecturé autour d'un boitier central. Celui-ci a un rôle central pour le véhicule, en plus de servir de boîte à fusibles et de contrôleur pour les fonctionnalités de signalisations, d'accès véhicule et de climatisation, il centralise la grande majorité des bus CAN et LIN du véhicule.

Attaques hardware Ce boitier est composé de deux cartes électroniques distinctes (une carte dédiée à l'alimentation électrique, servant de support aux fusibles, une carte dédiée aux fonctions de contrôle et d'interconnexion), liées entre elles par un connecteur.

La carte dédiée aux fonctions de contrôle et d'interconnexion est particulièrement complexe, et la position des composants et du plan de routage (positionnement des pistes électriques inter-composants) semble

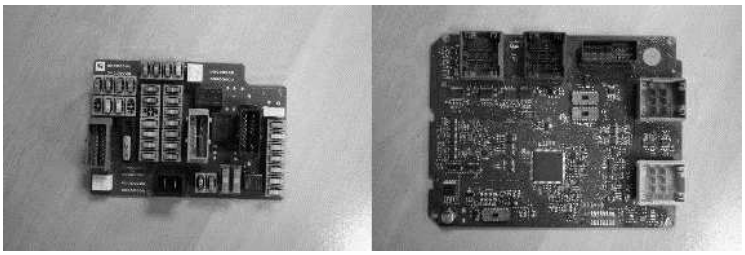

Figure 14. PCB du boitier central - étages de puissance et de contrôle

varier de version en version. Lors de l'audit, au moins deux versions différentes nous ont été transmises.

EEPROM Sur la carte de contrôle, on retrouve une EEPROM, à laquelle on ne peut accéder qu'en désolidarisant (à grands coups de scie à métaux) les deux cartes électroniques. Sur une des versions, une des pattes de l'EEPROM est accessible depuis le verso de la carte, ce qui permettait d'écouter les données en cours de lecture sur l'EEPROM sans avoir à endommager le boitier.

Cette mémoire EEPROM contient de nombreuses données sensibles (dont le code d'anti démarrage) stockées en clair. L'accès à ces données étant tout de même couteux en temps, et destructrice pour le boitier. On retrouve cependant sur internet des dumps mémoire de ce composant.

JTAG Un connecteur JTAG est présent, de manière accessible, au verso de la carte de contrôle. Afin d'utiliser ce connecteur, un mécanisme de protection additionnel doit être désactivé en reliant deux broches du connecteur entre elles.

La possibilité de débugger le microcontrôleur via cette interface semble toutefois être très aléatoire, une partie des pièces auxquelles nous avons pu accéder pouvant être débuggées, d'autre non. Les conditions exactes permettant de différencier pièces « débuggables » et pièces non « débuggables » n'ont pu être déterminées lors de notre audit.

Fonctions de protections inter-CAN La table de routage inter CAN a pu être obtenue en envoyant sur un BUS des trames CAN, et en écoutant sur un autre bus les messages sortants. Ce processus assez long nous a permis d'isoler la table de routage inter CAN.

À partir de cette table, on note que :

— L'ensemble des trames en 6XX et 7XX, provenant ou à destination du CAN de diagnostic, sont des trames légitimes de diagnostics.
— À l'exception de ces trames, un ensemble très restreint de messages peut être transmis entre deux CAN.
— En particulier aucune trame ne pouvait être transmise, lors de l'audit, depuis le CAN multimédia. Une compromission complète de ce système ne devrait donc pas compromettre l'intégrité du véhicule.

Attaques via le port de diagnostic Une étude par retro-ingénierie du firmware tournant sur la carte de contrôle a permis d'isoler plusieurs vulnérabilités et faiblesses cryptographiques. Ces dernières, remontées et prises en compte par le constructeur, n'ont pu être abordées dans ce document.

4 Problématiques récurrentes et défis à venir ?

4.1 Backdoors

La présence de backdoor dans les systèmes audités est d'une constance désarmante. Dans certains cas, le constructeur lui-même est parfaitement au courant, dans d'autres c'est complètement à son insu. Elles s'expliquent par les besoins de validation et de recettes mais ne sont absolument pas maîtrisées. La présence d'ADB sur les systèmes Android a par exemple pu être justifiée par le besoin de prendre des screenshots, alors qu'il permet l'exécution de code arbitraire.

De manière plus générale, les fournisseurs et les constructeurs n'ont pas encore intégré la notion de livraison de développement/test/production/.... Le développement d'un ECU et du code associé suit le même circuit et donc les outils de debug et de développement nécessaires lors des phases initiales se retrouvent en production dans les voitures vendues aux utilisateurs.

Cette tendance lourde est à l'origine de la plupart des problèmes de sécurité relevés lors de nos audits et les tentatives d'inversion de cette tendance sont pour l'instant assez timides.

4.2 Intégrité des systèmes

La vérification de l'intégrité des systèmes est un problème facile à comprendre. Il est par contre très complexe à résoudre et dépend des

capacités matérielles. Les plateformes physiques des systèmes multimédia sont maintenant assez développées pour avoir le support matériel nécessaire, mais ce n'est pas encore le cas pour le reste des calculateurs. Les constructeurs sont donc confrontés à la problématique classique du coût supplémentaire élevé pour un gain en sécurité invisible pour le produit. Le résultat est donc connu d'avance. Lorsque les besoins fonctionnels justifient la montée en gamme des capacités matérielles, alors les constructeurs ont la volonté d'en tirer un maximum, et notamment d'implémenter du secure boot.

Cependant, même lorsque la plateforme matérielle le permet, l'implémentation est très délicate et peut être contournée dans la plupart des cas rencontrés. Le manque de maîtrise de ces technologies par les fournisseurs automobiles traditionnels est flagrant. La pression des coûts étant particulièrement forte dans ce secteur de l'industrie, elle n'incite pas les fournisseurs à investir dans la sécurité de leurs produits tant qu'ils ne sont pas tenus pour responsables.

La couverture de cette vérification est aussi soumise à interprétation selon les fournisseurs et les constructeurs. Elle peut s'arrêter au bootloader, au kernel, au système, aux applications, au code tiers etc. Aucune communication n'étant faite et aucune information publique n'étant diffusée, il est impossible pour un utilisateur de savoir quelle technologie est mise en œuvre dans ces ECUs.

4.3 Intégration de la sécurité

Les problématiques de sécurité informatique pour les véhicules sont une notion encore très récente pour les constructeurs. Elle est poussée par les nouveaux usages, notamment connectés, qui sont de plus en plus répandus. Ce changement fondamental est encore en cours d'intégration dans les processus de développement, ce qui explique pourquoi les audits sont soit réalisés lorsque le système n'est pas terminé (donc non représentatif du produit final), soit réalisés lorsque le système est déjà vendu et que les possibilités de correction sont très restreintes.

La diffusion des mises à jour de sécurité pose aussi de nombreux problèmes :

— Les utilisateurs ne sont pas encore habitués à faire eux-mêmes la mise à jour de leur voiture
— La connectivité du véhicule est différente suivant les modèles, les options, l'appairage éventuel d'un équipement de l'utilisateur, le lieu où il se trouve, l'abonnement choisi, etc.

— Il faut que l'ECU reste alimenté pendant la durée de la mise à jour, que le moteur reste allumé, etc.

4.4 Défis à venir

Fonctions futures Malgré les problématiques récurrentes, de nombreuses difficultés supplémentaires vont venir compliquer l'équation déjà difficile de la sécurité des véhicules. La principale étant le véhicule autonome, prévu dans les roadmap autour de 2020.

Le principe est le suivant : une multitude de capteurs internes, externes ainsi que des tiers envoient des informations en temps réel à un calculateur qui décide de la vitesse et de la trajectoire du véhicule. Le calculateur pourra donc décider de freiner brusquement ou d'entamer une procédure d'évitement, y compris à haute vitesse, s'il est persuadé qu'il va entrer en collision. Les signaux peuvent provenir de sources internes (calcul de la vitesse instantanée par exemple), externes (LIDAR, caméras[5]) ou de tiers tels que les véhicules environnants ou encore la cartographie du lieu où le véhicule se trouve qui est directement envoyée depuis le boitier multimédia qui stocke la cartographie globale.

La sécurisation de ces systèmes est un enjeu vital[2] et la maîtrise inégale démontrée par les différents constructeurs est loin d'inspirer confiance comme en attestent les expériences de Jeep et Tesla.

D'autres fonctions moins critiques pour l'intégrité des utilisateurs mais toutes aussi potentiellement catastrophiques pour la sécurité des systèmes sont en cours de développement et de déploiement tels que l'auto partage, la dématérialisation de la clé de contact, le démarrage à distance etc.

Les constructeurs étant conscients des enjeux, ils commencent à s'organiser pour résister aux futures attaques qui ne manqueront pas d'arriver étant donné l'attrait grandissant des hackers pour les véhicules. Les architectures électroniques sont elles en train de changer pour intégrer les concepts de défenses en profondeur.

Réglementations : L'environnement législatif est aussi en cours de mutation. Les différents gouvernements et autorités de régulation réfléchissent aux contraintes à imposer aux constructeurs pour protéger leurs utilisateurs. Les Etats Unis en particulier.

Environnement : De nouveaux acteurs viennent s'intégrer dans cet écosystème, comme les assureurs ou les systèmes après-vente offrant des fonctions connectées aux véhicules qui n'en n'ont pas. Généralement, ces systèmes se branchent sur la prise OBD réglementaire. Cette prise OBD n'était jusque-là utilisée que par les réparateurs agréés avec des outils dédiés. Elle avait donc un accès illimité aux bus CAN. Les nouveaux systèmes peuvent être connectés aux smartphones, au WiFi ou même directement sur internet et peuvent donc être compromis. Comme ces systèmes restent branchés lorsque les véhicules roulent, ces nouveaux usages nécessitent une réponse à la hauteur des enjeux de la part des constructeurs.

5 Remerciement

Les auteurs souhaitent remercier Cédric BUXIN, dit le Sorcier, pour son aide précieuse à travers les méandres du monde électronique. Aux personnes impliqués dans la réalisations des tests chez les différents constructeurs pour nous avoir accueillis avec bonne humeur, accepter avec philosophie la destruction de nombreux boitier et plus généralement pour nous avoir fait confiance. Enfin nos relecteurs, et les membres du comité de sélection du SSTIC pour leurs conseils avisés.

Références

1. . killinginthenameof. 2013.
2. dailymail, Ellie Zolfagharifard . When Tesla's autopilot goes wrong . 2014.
3. Debojyoti Bhattacharya, Sriram Subramanian Neelakantan, Jamshid Shokrollahi, Hans Loehr . Bootstrapping trust in automotive networks with different types of ECUs . 2014.
4. Dr. Charlie Miller, Chris Valasek . Remote Exploitation of an Unaltered Passenger Vehicle . 2015.
5. Jonathan Petit, Bas Stottelaar, Michaelb Feiri, Frank Kargl. Remote Attacks on Automated Vehicles Sensors : Experiments on Camera and LiDAR. 2015.
6. Ryo Kurachi, Yutaka Matsubara, Hiroaki Takada, Naoki Adachi, Yukihiro Miyashita and Satoshi Horihata . CaCAN - Centralized Authentication System in CAN . 2014.
7. Rapid7. Metasploit module : webview. 2013.
8. revolutionary. github - zergrush exploit. 2013.
9. The Android Exploid Crew. Gingerbreak. 2013.
10. The Android Exploid Crew. zimperlich - source. 2013.

6 Glossaire

ADB (canal) Android Debug Bridge, outil en ligne de commande permettant de communiquer, de débugger, et d'exécuter des commandes sur un dispositif android.

Ashmem Mécanisme d'allocation de mémoire partagée.

Bootrom Premier code exécuté au démarrage d'un composant, souvent stocké sur une mémoire non inscriptible.

Busybox Logiciel implémentant un grand nombre de commandes UNIX, très utilisé dans l'embarqué.

CAN (bus) Bus de communication série, sur une ligne partagée, très utilisé dans l'automobile.

chaîne de boot Suite de composants exécutables, allant de la bootrom à l'OS final.

Cloud car sharing Service de partage de clés en ligne, permettant par exemple de gérer les accès à une flotte de véhicules d'entreprises.

Commande AT Commande, principalement orientée télécom, pour le système OpenAT de Sierra Wireless.

D-BUS Logiciel de communication inter-processus sous système Linux.

DALVIK Machine virtuelle JAVA incorporée au système Android.

Dump mémoire Processus permettant de récupérer une image du contenu d'une mémoire.

ECU Electronic Control Unit, boitier électronique contrôlant une série de mécanismes automobiles (moteur, frein, etc.).

eFuse Fusibles électroniques permettant de configurer le comportement d'un composant, peuvent être reprogrammés.

EEPROM Electrically Erasable Programmable Read Only Memory, mémoire read only, pouvant être ré-écrite de manière logicielle.

FDT Flattened Device Tree, structure de sérialisation et de stockage de données.

Intent Mécanisme de communication inter processus sur le système Android.

JTAG Joint Test Action Group, terme, un peu abusif, désignant les ports de contrôle et de test d'une carte électronique.

Kexec Mécanisme permettant de démarrer un kernel allant écraser le kernel existant. Souvent utilisé, dans une bootchain, pour charger le système d'exploitation final.

LIN (bus) Bus Local Interconnect Network, bus série utilisé dans les véhicules automobiles, en complément du bus CAN.

OBD (prise) Prise On Board Diagnostic, connecteur présent dans l'habitacle d'un véhicule et permettant d'interroger les différents ECU.

PCB Printed Circuit Board, carte électronique.

Platooning Asservissement de la vitesse d'un groupe de véhicules afin de conserver une distance constante.

PIC Famille de microcontrôleur.

SoC System On Chip, composant incluant un microcontrôleur et un ensemble de sous-composants (mémoires internes, composants dédiés, ...).

TrustZone ARM Mécanisme de sécurité propre aux SoC ARM, permettant d'exécuter un environnement de code sécurisé.

UART Console série.

UBI (blok) Unsorted Block Images, système de gestion de volumes logiques conçu pour fonctionner sur une mémoire flash non formatée.

UBIFS Système de fichier UBI. Système de fichiers reposant sur la séparation en volumes logiques UBI.

U-boot Bootloader, un programme chargé d'initialiser et de lancer un ou plusieurs systèmes d'exploitation, très utilisé dans l'embarqué.

uImage Image du kernel Linux accompagnée d'informations pour u-Boot.

VIN Vehicle Identification Number, numéro de série propre à un véhicule.

Vold Service de gestion des volumes disque, équivalent de mountd.

Watchdog Système s'assurant du bon fonctionnement d'un composant, le redémarrant en cas de besoin.

X-Loader Bootloader, voir u-boot.

zImage Image du kernel Linux compressée.

Zygote Sur Android, processus racine, qui sera cloné à chaque lancement d'une application.