# 汽车总线防火墙

# （MCU平台） 文档

## V1.1

# 第1章　汽车防火墙 SDK 介绍

## 1.1　简要介绍

VT_SCAN MCU Engine 是针对汽车信息安全性提供的车载防火墙解决方案接口函数。此方案通过嵌入 VT_SCAN 安全防护层和配置防火墙特征来监控 CAN 总线，并发现总线流量异常。VT_SCAN 通过模块化的设计，强调软件的易移植性和通用性。 以达到对目标设备资源占用的最小化和方便移植，帮助客户实现在目标设备的移植，完成对 CAN 总线的安全防护与监控。

## 1.2　开发运行环境和平台支持

- 平台支持
  MCU和LINUX两种环境
- 支持两种检测方式
  - 实时阻断
    实时检测并阻断恶意指令，上报云端
  - 非实时阻断，仅向云端报警
    检测后并不阻断，仅向云端报警

## 1.3　组成说明

VT_SCAN分为用户API调用库和engine，特征库。 VT_SCAN core 核心库包括两个部分：

a. VT_SCAN 防火墙匹配引擎 Matching Engine
b. VT_SCAN 特征库 Signature
c. APIs

## 1.4 资源需求

MCU平台：256kb memory, one core, 2-CAN

LINUX平台： 2MB memory

# 第2章　模块构成

## 2.1　模块构成说明

VT_SCAN 主要由防火墙引擎模块、接口模块和特征库模块三部分组成，具体产品结构如下图所示：

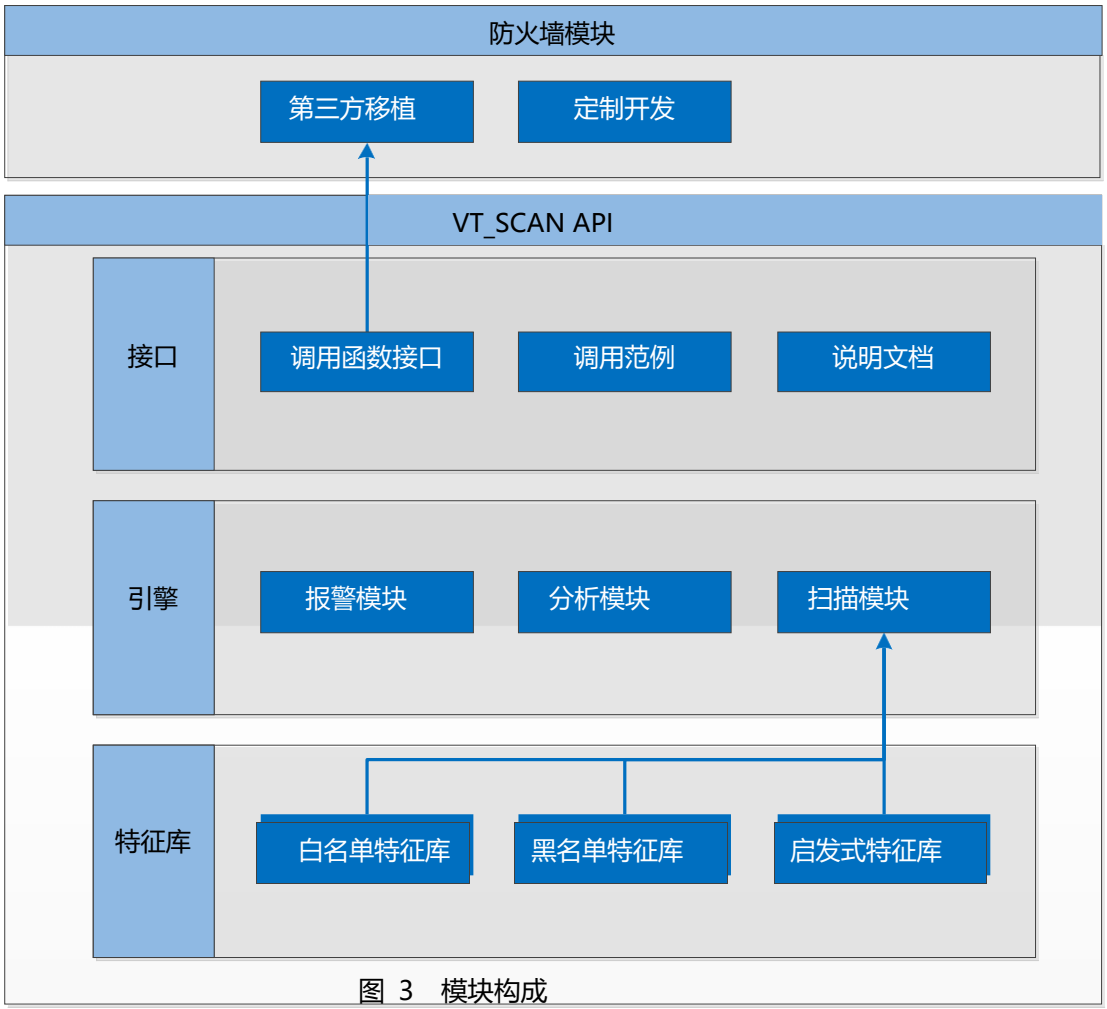



图 3　模块构成

| | |
|---|---|
| 引擎 | 防火墙引擎，分析数据流量，扫描匹配并返回检测结果。 |
| 接口 | 提供函数调用接口，编写例子代码调用引擎，将分析结果传递给调用接口。因此开发者可以按照使用手册， 通过函数调用为产品扩展汽车总线防火墙防御模块。 |
| 特征库 | 多种特征库检测异常，攻击，故障等场景。 |

## 2.2 VT_SCAN安全方案移植集成步骤

本节介绍将VT_SCAN移植到目标设备的基本流程和步骤。 VT_SCAN安全防护方案的移植需要客户提供：

a. 汽车网关或者 TBOX 设备和相关 CAN 设备驱动用户手册，以及集成环境的资源限制，如内存等。
b. 目标平台的编译环境，以生成适合目标设备的 VT_SCAN 类库
c. 网络通讯接口函数。VT_SCAN 将通过网络通讯接口上传报警日志和相关数据到云端汽车防火墙实时监控平台。

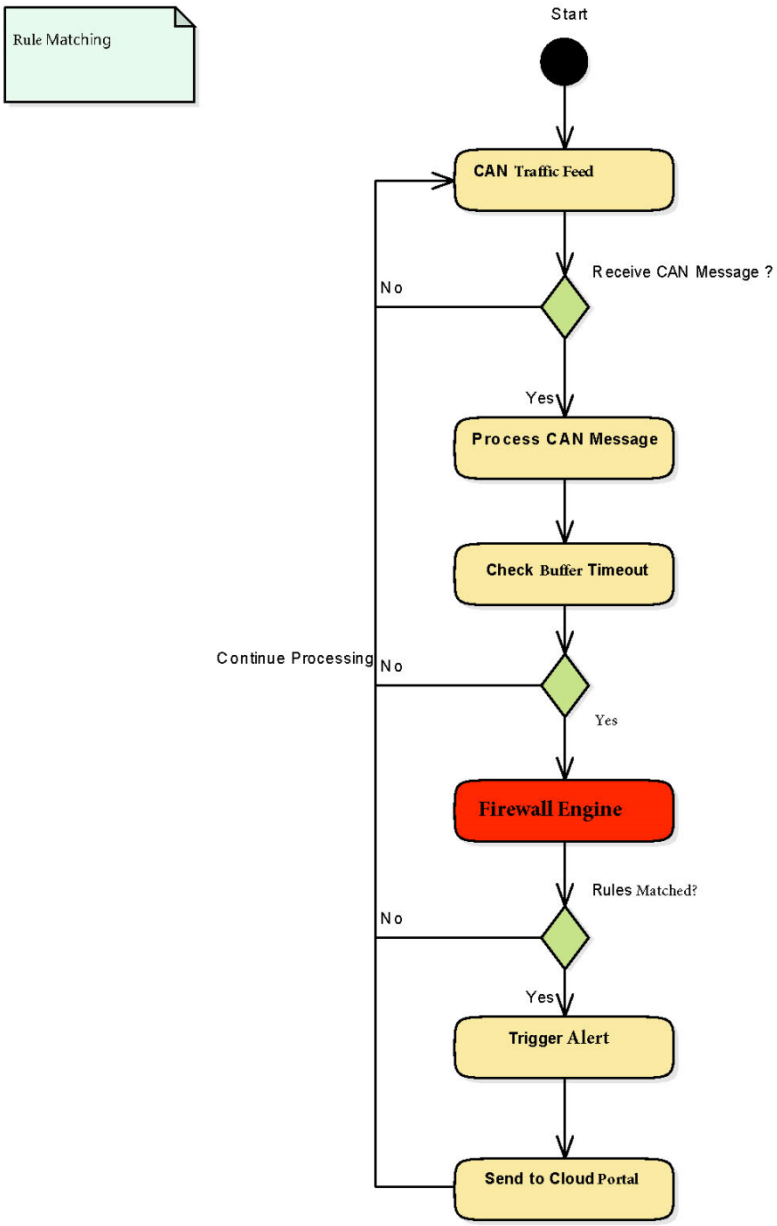
移植集成步骤：

a. 根据目标设备的 CAN 通信驱动接口，修改、调整 VT_SCAN 相应的驱动 。
b. 使用目标平台的编译环境和相应的目标平台系统类库，重新编译生成 VT_SCAN 类库 。
c. 修改 VT_SCAN 例子代码，在目标平台运行启动引擎，进行测试调试。
d. 在目标平台上完成 VT_SCAN 安全防护层集成。

# 第3章　参数配置

## 3.1　引擎扫描开关

引擎检测方式分为在线inline和离线offline方式。在线方式可以集成在网关自身CAN包转发模块里，用来阻断对汽车攻击的指令片段。离线方式用于并行处理，不影响网关正常工作。发现异常后进行报警。



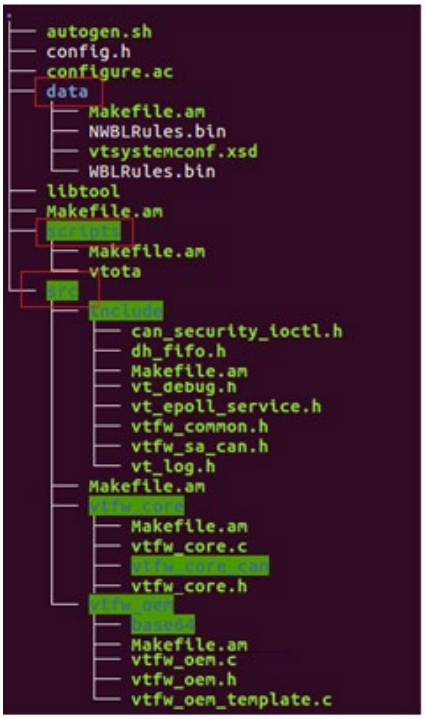

引擎可以设置不同的扫描开关，满足不同场景需求：

| Engine model | 扫描开关 | 应用场景 |
|---|---|---|
| FW ALL | Default normal 缺省（全部开关打开）<br><br>黑名单扫描开关<br>攻击扫描开关<br>启发式扫描开关<br>特征扫描开关 | 以高检测率，低误报率为第一优先，扫描速度和内存CPU资源占用其次<br><br>分析扫描统计结果 |
| FW Quick | 快速扫描<br><br>特征扫描开关 | 以扫描速度和资源占用小为优先，扫描开销小 |
| FW Quick + Heuristic | 启发式扫描开关<br>特征扫描开关<br><br>通过启发式判断再次确认 | 以速度+低误报率优先，牺牲部分速度和内存CPU，提高扫描检测性能 |

三种扫描开关，分别是快速扫描，启发式扫描和深度完全扫描。具体特点见上表。

## 3.2 引擎参数配置

The firewall engine opens the parameter configuration for customization. For example, the engine may have autogen.sh, configure.ac, Makefile.am, data folder, scripts folder and src folder like picture below:




Firewall folder structure

- autogen.sh is for preparing a build for compilation, verifying functionality, and overcoming common build preparation issues.
- configure.ac is an executable script to aid in developing a program to be run on a wide number of different platforms. It matches the libraries on the user's computer, with those required by the program before compiling it from its source code.
- Data folder contains all data for the firewall application such as WBLRules.bin and NWBLRules.bin. They will be installed to "/usr/data" folder on target platform.
- The OEM folder structure includes vtfw_oem.c, vtfw_oem_template.c, vtfw_oem.h and Makefile.am.

## 3.3 OEM接口

Firewall SDK has also implemented the OEM APIs. These OEM API implementations can help third parties customize and integrate codes smoothly.

### Security Agent Message Type

This type is used to communicate between OEM layer and SA Core module.

| **Name** | **vt_sa_msg_t** | |
|---|---|---|
| **Type** | Structure | |
| **Elements** | msg | CAN Data Frame |
| | can_bus_channel_id | CAN Bus Channel ID |
| | can_bus_msg_way | CAN Bus Connection (TX/RX or ANY) |
| | matched_internal_rule_id | Internal rule ID |
| | msg_timestamp | TimeStamp of CAN Message |
| **Description** | This is common structure. | |

### OEM Initialization Function

| **Name** | **vtfw_oem_init** | |
|---|---|---|
| **Syntax** | int vtfw_oem_init(void) | |
| **Parameter [in]** | None | N.A |
| **Parameter [out]** | None | N.A |
| **Return** | FW_OK | Initialize successful |
| | FW_NOT_OK | Initialize failed |
| **Description** | This API will initialize OEM code. | |

## OEM WBL_Matched_Result Function

This function is a callback function for white black list rules. It can be implemented by third parties. When the firewall system is matching WBL, system will call this callback function to run OEM custom implementations.

| **Name** | **vtfw_oem_wbl_matched_result** | |
|---|---|---|
| **Syntax** | int vtfw_oem_wbl_matched_result(vt_sa_msg_t *msg_t) | |
| **Parameter [in]** | *msg_t | Pointer to security agent structure vt_sa_msg_t. |
| **Parameter [out]** | None | N.A |
| **Return** | FW_OK | Invoke trigger callback successful |
| | FW_NOT_OK | Invoke trigger callback failure |
| **Description** | This is type definition of white black rules callback function. OEM will define detail process for this API such as report event for Firewall Server, create report log etc. | |

## OEM NWBL_Matched_Result Function

This function is a callback function for none white black list rules. It will be implemented by third parties. When the firewall system matches NWBL, system will call this callback function to run OEM custom processing.

| **Name** | **vtfw_oem_nwbl_matched_result** | |
|---|---|---|
| **Syntax** | int vtfw_oem_nwbl_matched_result(uint32_t rule_id,char* LucReportContent) | |
| **Parameter [in]** | rule_id | Rule Index based on NWBL list rules |
| | LucReportContent | Pointer to description detail of matching rule |
| **Parameter [out]** | None | N.A |
| **Return** | FW_OK | Invoke trigger callback successful |
| | FW_NOT_OK | Invoke trigger callback failure |
| **Description** | This is type definition of none white black rules callback function. OEM will define detail process for this API such as report event for Firewall Server, create report log etc. | |

## OEM Deinitialization Function

This function will be called to closes or releases memory, buffer, thread that they are created by third parties.

| **Name** | **vtfw_oem_close** |
|---|---|
| **Syntax** | void vtfw_oem_close(void) |

| | | |
|---|---|---|
| **Parameter [in]** | None | N.A |
| **Parameter [out]** | None | N.A |
| **Return** | None | N.A |
| **Description** | This API will de-initialize OEM code. | |

## 3.4 OEM参数配置

The OEM configuration parameters include period timer for trigger alert matching, current policy rules for white black list and specific rules, CAN interface etc. Depending on configuration of OEM, Firewall can set limit number of security agent rules at run-time.

Here is an example of a configuration file:

```
<xsd:complexType name="VTDDetailConfig">
    <xsd:sequence>
        <xsd:element name="vt_wbl_file" type="xsd:string" minOccurs="0"/>
        <xsd:element name="vt_nwbl_file" type="xsd:string" minOccurs="0"/>
        <xsd:element name="vt_wbl_last_file" type="xsd:string" minOccurs="0"/>
        <xsd:element name="vt_nwbl_last_file" type="xsd:string" minOccurs="0"/>
        <xsd:element name="vt_baud_rate" type="xsd:string" minOccurs="0"/>
        <xsd:element name="vt_phone_numer" type="xsd:string" minOccurs="0"/>
        <xsd:element name="vt_wbl_report_delay" type="xsd:string" minOccurs="0"/>
        <xsd:element name="vt_nwbl_report_delay" type="xsd:string" minOccurs="0"/>
        <xsd:element name="vt_can_name" type="xsd:string" minOccurs="0"/>
        <xsd:element name="vt_bridge_can_name" type="xsd:string" minOccurs="0"/>
        <xsd:element name="vt_enable_bridge" type="xsd:string" minOccurs="0"/>
        <xsd:element name="vt_max_sa_rules" type="xsd:string" minOccurs="0"/>
        <xsd:element name="vt_time_period" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="VTSystemConfig">
    <xsd:sequence>
        <xsd:element name="config" type="VTDDetailConfig" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
```

Configuration definition

Firewall Low Driver Module requests OEM add some Application Program Interface (API) such as: initialize OEM configuration, define callback function for trigger matching rule etc.

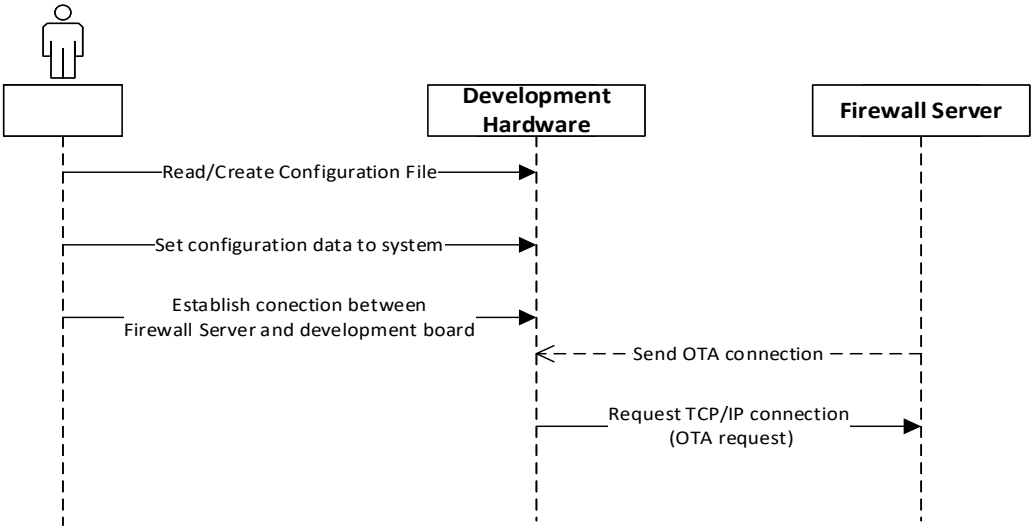
### OEM initialization

In OEM Initialization, OEM will invoke API to read configuration file, set some configuration after read configuration file. The figure below describes some steps which request user initialize before run Firewall system.

### Configuration Parameter Definition

| Parameter | Description | Example |
|---|---|---|
| vt_wbl_file | This parameter describes current name of white black list rules | WBLRules |

| | | |
|---|---|---|
| vt_nwbl_file | This parameter describes current name of none white black list rules | NWBLRules |




OEM Initialization

# 第4章　攻击点检测

## 4.1　检测方法

### 车型白名单扫描

| 目的: |
| --- |
| 车辆正常情况 下的监控。 |
| 指标: |
| 误报率 |
| 测试方法 |
| 测试车型不同状态下的流量，特征库判断是否属于正常情况，并返回结果。 |
| 结果 |
| 是否异常 |

### 其他车型白名单扫描

| 目的: |
| --- |
| 误报测试一部分内容，防火墙对每款车型定制，测试其他车辆正常流量，可以提高误报率指标。 |
| 指标: |
| 2不同车型的误报率指标。 |
| 测试方法 |
| 测试不同车型不同状态下的流量，特征库判断是否属于正常情况，并返回结果。 |
| 结果 |
| 是否异常 |

### 攻击流量扫描

| 目的: |
| --- |
| 模拟多种左右不同场景的总线攻击，例如扫描，暴力破解，DoS, Spoof等，检测攻击。 |
| 指标: |
| 能够检测到每一种攻击。 |

| 测试方法 |
| --- |
| 重放攻击流量文件，特征库引擎扫描并返回结果。 |
| 结果 |
| 是否异常 |

**精确攻击流量**

| 目的： |
| --- |
| 针对该车型的的漏洞隐患而生成对应的精确攻击，这些攻击危害更大。 |
| 指标： |
| 能够检测并实时阻断 |
| 测试方法 |
| 重放相关流量或指令，特征库引擎扫描并返回结果。 |
| 结果 |
| 是否异常 |

## 4.2 检测的攻击点包括：

- replay attack重放一段正常流量或者其他车辆流量，例如，重放充放电流量数据。

重放单个正常包
把收集的板子正常包重放一个，看板子能否检测出异常

重放多个包
重放一系列数据包，看板子能否检测出异常

- Fuzzy 模糊测试：

随机CAN ID和payload

- Denial of Service测试：

低ID DoS 测试
发送CAN ID=0的优先级最高的包。看板子是否能正常相应普通流量请求。
高ID DoS 测试
发送CAN ID=0x7ff的优先级低的包。看板子是否能正常相应普通流量请求。

- CAN proof attack

发送某CAN ID的数据包，伪装该CAN ECU通信。

- overflow 溢出Attack：

模拟一些特殊数值，例如空值，越界数字等，看程序是否处理数字溢出
发送一些payload长度大于8位的包，看板子能否处理 payload长度溢出

- CAN Bus System Probing(UDS/XCP)
- ECU scan
- ECU Security scan
- Privilege Elevation
- ECU parameter tempering
- ECU Data modification

  and more.

## 4.3 实时检测

防火墙检测功能包括：

- 持续监控分析
- 攻击预测
- 检测
- 响应： 攻击回溯， 调查取证， 策略调整，攻击隔离

扫描结果以文本方式输出。以后可以通过网口传到云端在监控平台显示。防火墙匹配某条特征时会触发一个报警事件，这个报警事件包括该特征的代号和事件描述。当报警事件送到云端时，云端会在云端报警数据库中查询该代号对应的信息，显示在云端监控界面。

对实时检测的性能需求可以通过配置相应时间执行。例如，报警时间可以定制化从秒级到分钟级别。然而防火墙硬件平台的性能对实时检测有制约性：快速时间相应需要ECU模块提供更高处理能力，包括内存。

改变实时检测的频率需要对防火墙特征库进行参数调整，适应由此带来的影响。参数调整时间通常为1-2周，并需要进行误报测试。防火墙模块自身提供的开放接口可以进行相关的调节。

## 4.4 内存要求

基于不同的内存要求可以进行特征库裁剪达到不同内存要求。

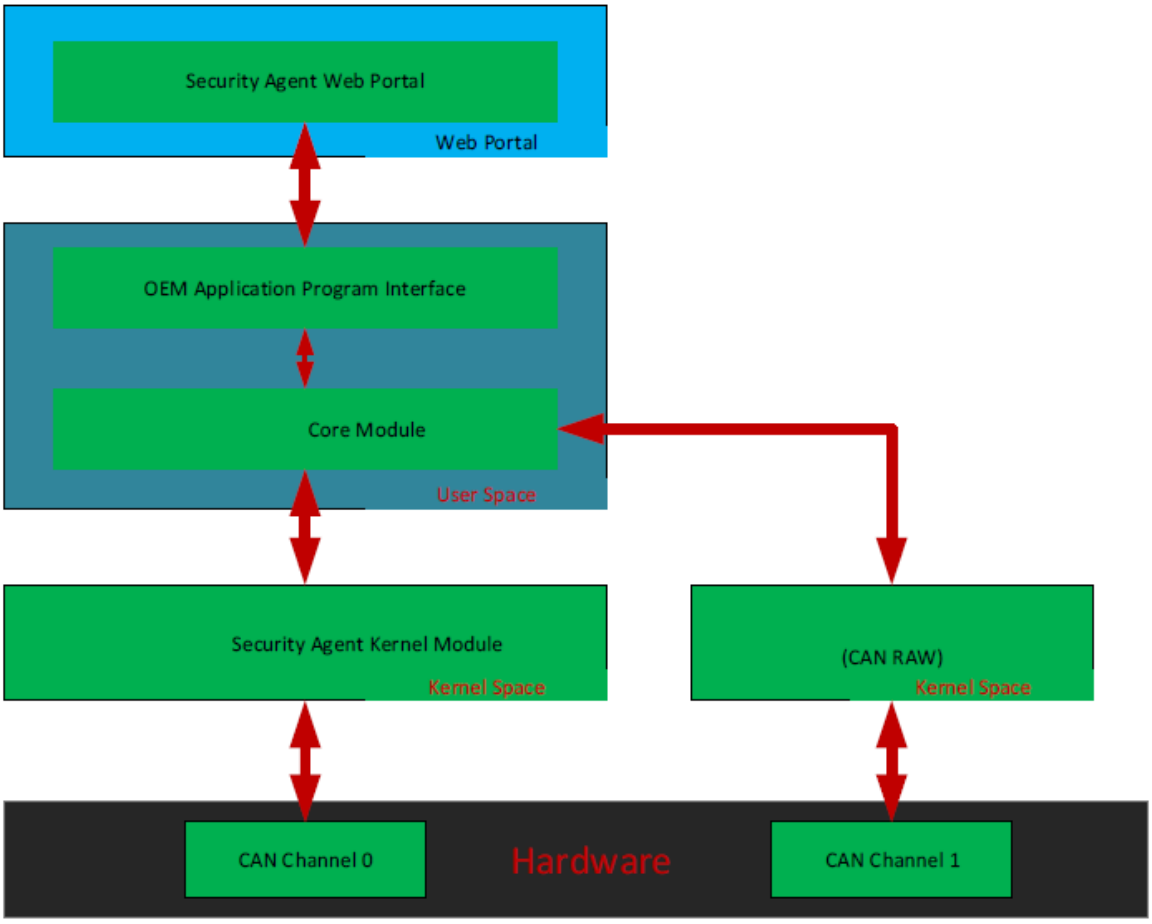
| 内存要求 | 支持ID数目 | 协议分析能力 | 实时检测频率 |
|---|---|---|---|
| <200KB | 30-50左右 | 初步 | 30秒左右 |
| 200-500KB | 100左右 | 中级 | 30秒左右 |
| >500KB-1MB | 140以上 | 高级 | 10-30秒 |

ID数目对内存使用影响最大。

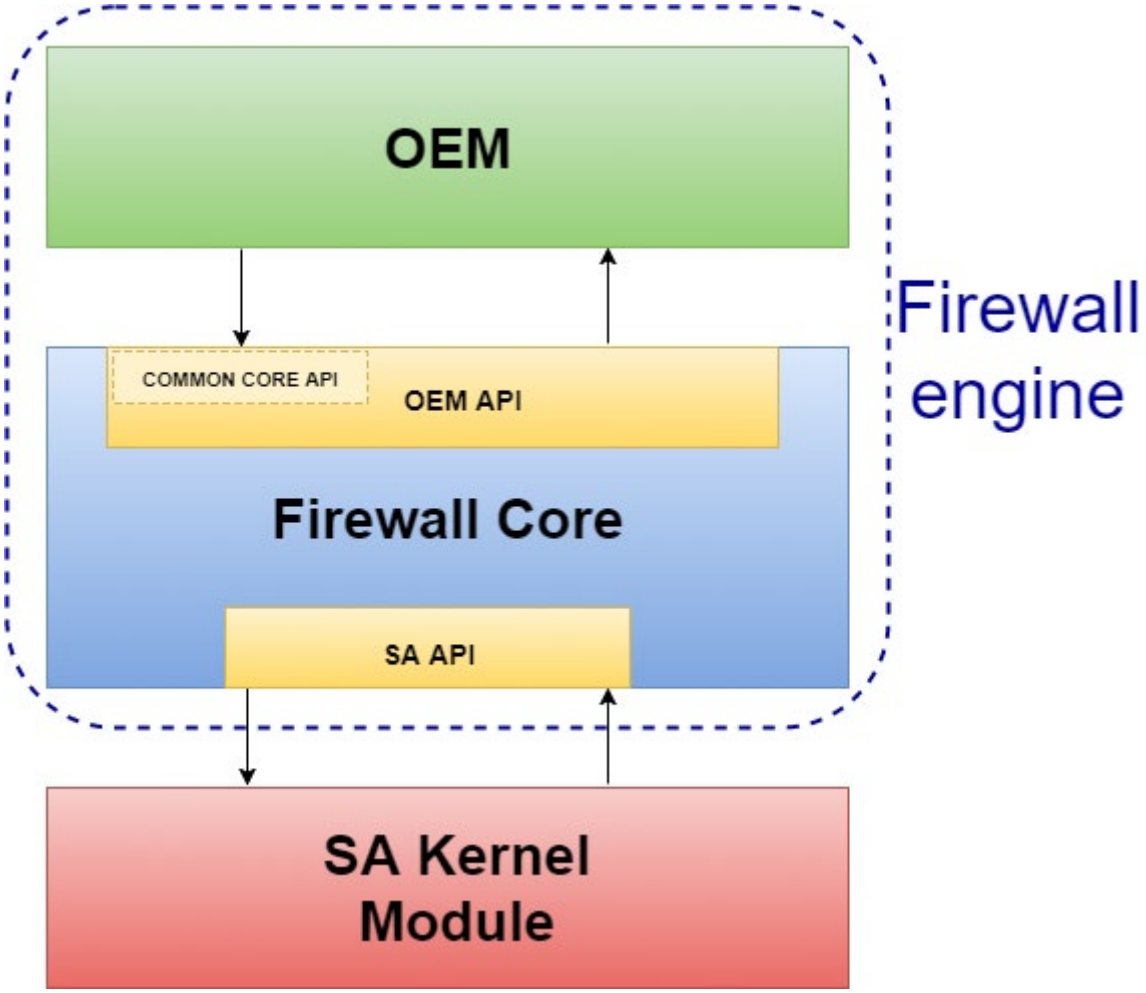
# 第5章　集成接口

## 5.1　硬件接口

防火墙硬件接口包括ECU的接口，这些ECU可以是中央网关，OBD后面网关和关键ECU部件。




Firewall SDK consists of two components: FireWall Engine(FWE) and Security Agent module(SA). The SA kernel module works in on-line mode. It may block malicious CAN messages specified in WBL(WhiteBlackList) rules at low driver layer. While Firewall engine works in off-line mode. It detects attacks based on NWBL(NonWhiteBlackList) rules. The Firewall engine provides APIs for hardware (gateway or ECUs) to custom SA and FWE. If VT Firewall engine detects any attacks, it will call the OEM API call-back functions to report the attack.  The figure below describes overall Firewall architecture:




Hardware Interface

Developers can leverage SA APIs to porting firewall onto different hardware platforms, such as STM32/NXP/TI. Since each vendor provides its own Developer Library Interfaces, developers just changes the corresponding codes to make FWE running on their hardware.