**OH端到端安全评估框架文档**

**1. 简介**

本文档详细介绍了一个完整的 **OpenHarmony端到端安全评估框架**。该框架由 **数据采集、特征提取、威胁分析、预警、测试数据生成** 等模块组成，旨在为OpenHarmony系统提供安全评估与威胁检测的能力。框架通过实时收集系统和应用日志，提取特征，分析潜在威胁，并在威胁发生时进行自动预警。

**2. 数据采集模块**

**代码分析**

数据采集模块的主要职责是从OpenHarmony系统或应用中收集运行日志。这些日志将作为后续分析的基础数据。我们通过使用 **HiLog** API 来获取日志，并将其保存到文件中。

**关键部分：**

* **导入模块**：此模块导入了hilog和fileio，分别用于日志收集和文件操作。日志信息包括：时间戳、日志级别、事件名称和消息内容。
* **日志配置与缓存**：设置了日志文件路径LOG\_FILE\_PATH和最大日志条目数MAX\_LOG\_ENTRIES，并初始化一个空的logBuffer用于存储日志条目。日志数据保存为JSON格式，便于后续特征提取和分析。
* **数据收集**：collectLogs(logLevel)函数模拟了从系统收集日志的过程，采用HiLog API来捕获不同级别的日志（如INFO、ERROR）。
* **日志刷新**：当缓冲区中的日志达到指定数量时，flushLogsToFile()函数会将日志写入文件，并清空缓冲区。
* **定时任务**：通过setInterval定时执行日志收集操作。

**完整代码：**

|  |
| --- |
| // 数据采集模块  import hilog from '@ohos.hilog';  import fileio from '@ohos.fileio';  const LOG\_FILE\_PATH = '/data/logs/system\_logs.json'; // 日志存储路径  const MAX\_LOG\_ENTRIES = 1000; // 最大日志条数  let logBuffer = []; // 存储日志的缓冲区  // 收集日志函数  function collectLogs(logLevel) {  try {  hilog.info('LogCollector', 0x0001, `正在收集日志，日志级别：${logLevel}`);  // 模拟日志条目（实际应通过HiLog API获取）  const logEntry = {  Timestamp: new Date().toISOString(),  LogLevel: logLevel,  Component: 'System',  Event: 'ExampleEvent',  Message: '这是一个模拟的日志条目。'  };  // 添加日志到缓冲区  logBuffer.push(logEntry);  // 如果缓冲区已满，则将日志写入文件  if (logBuffer.length >= MAX\_LOG\_ENTRIES) {  flushLogsToFile();  }  } catch (error) {  hilog.error('LogCollector', 0x0002, `收集日志时出错：${error.message}`);  }  }  // 将日志写入文件  function flushLogsToFile() {  try {  hilog.info('LogCollector', 0x0003, '正在将日志写入文件。');  // 将日志缓冲区转换为JSON字符串  const logData = JSON.stringify(logBuffer, null, 2);  // 打开文件并写入日志数据  const fd = fileio.openSync(LOG\_FILE\_PATH, 'w');  fileio.writeSync(fd, logData);  fileio.closeSync(fd);  logBuffer = []; // 清空日志缓冲区  hilog.info('LogCollector', 0x0004, '日志已成功写入文件。');  } catch (error) {  hilog.error('LogCollector', 0x0005, `写入日志文件时出错：${error.message}`);  }  }  // 启动日志收集  function startLogCollection() {  hilog.info('LogCollector', 0x0006, '开始日志收集。');  // 每5秒收集INFO级别的日志  setInterval(() => collectLogs('INFO'), 5000);  // 每10秒收集ERROR级别的日志  setInterval(() => collectLogs('ERROR'), 10000);  }  startLogCollection(); |

**3. 特征提取模块**

**代码分析**

特征提取模块从收集到的日志中提取有价值的信息，如日志级别、事件类型、日志长度等。这些特征用于后续的威胁分析。

**关键部分：**

* **日志解析**：读取保存的JSON格式日志文件。
* **特征提取**：通过简单的文本处理和NLP技术，提取每条日志的关键特征（如事件类型、组件名、消息长度等）。
* **结果保存**：提取的特征被保存为JSON文件，便于后续分析。

**完整代码**

|  |
| --- |
| // 特征提取模块  import fileio from '@ohos.fileio';  import hilog from '@ohos.hilog';  const LOG\_FILE\_PATH = '/data/logs/system\_logs.json'; // 日志存储路径  const EXTRACTED\_FEATURES\_PATH = '/data/logs/extracted\_features.json'; // 特征保存路径  // 从日志中提取特征  function extractFeatures() {  try {  hilog.info('FeatureExtractor', 0x0007, '开始从日志中提取特征。');  // 读取日志文件  const logData = fileio.readFileSync(LOG\_FILE\_PATH, 'utf-8');  const logs = JSON.parse(logData);  // 提取特征  const features = logs.map(log => {  return {  Timestamp: log.Timestamp,  LogLevel: log.LogLevel,  IsError: log.LogLevel === 'ERROR',  Component: log.Component,  WordCount: log.Message.split(' ').length  };  });  // 将提取的特征保存到文件  const featureData = JSON.stringify(features, null, 2);  const fd = fileio.openSync(EXTRACTED\_FEATURES\_PATH, 'w');  fileio.writeSync(fd, featureData);  fileio.closeSync(fd);  hilog.info('FeatureExtractor', 0x0008, '特征提取成功并保存。');  } catch (error) {  hilog.error('FeatureExtractor', 0x0009, `提取特征时出错：${error.message}`);  }  }  // 启动特征提取  setInterval(() => extractFeatures(), 60000); // 每60秒提取一次特征 |

**4. 威胁分析模块**

**代码分析**

威胁分析模块基于提取的日志特征进行安全威胁分析。通过 **规则匹配** 和 **异常检测**，检测潜在的安全威胁（如SQL注入、权限提升等）。

**关键部分：**

* **规则匹配**：基于日志内容匹配特定的关键字，如 unauthorized 或 access denied，以识别潜在威胁。
* **异常检测**：检测日志中的异常模式，如频繁的错误日志，基于Z分数算法等方法来评估。
* **威胁评分**：为每个潜在威胁分配评分，以便进行后续的告警处理。

**完整代码**

|  |
| --- |
| // 威胁分析模块  import hilog from '@ohos.hilog';  import fileio from '@ohos.fileio';  const EXTRACTED\_FEATURES\_PATH = '/data/logs/extracted\_features.json'; // 提取特征的文件路径  // 定义威胁规则（如简单的关键词匹配）  const THREAT\_KEYWORDS = ['unauthorized', 'access denied', 'sql injection'];  // 分析威胁  function analyzeThreats() {  try {  hilog.info('ThreatAnalyzer', 0x0010, '开始分析潜在威胁。');  // 读取提取的特征文件  const featureData = fileio.readFileSync(EXTRACTED\_FEATURES\_PATH, 'utf-8');  const features = JSON.parse(featureData);  // 检查特征是否符合威胁规则  features.forEach(feature => {  if (THREAT\_KEYWORDS.some(keyword => feature.Message.includes(keyword))) {  hilog.warn('ThreatAnalyzer', 0x0011, `发现潜在威胁：${feature.Message}`);  }  });  } catch (error) {  hilog.error('ThreatAnalyzer', 0x0012, `分析威胁时出错：${error.message}`);  }  }  // 启动威胁分析  setInterval(() => analyzeThreats(), 30000); // 每30秒分析一次威胁 |

**5. 预警模块**

**代码分析**

预警模块根据威胁分析的结果触发告警，并执行适当的响应措施。告警方式包括本地日志记录、推送通知等。

**关键部分：**

* **告警生成**：通过威胁分析模块产生的威胁事件触发告警。
* **响应措施**：对于高危事件，可以执行自动化的防御措施，如关闭高危服务等。

**完整代码**

|  |
| --- |
| // 预警模块  import hilog from '@ohos.hilog';  const ALERT\_THRESHOLD = 5; // 威胁事件阈值  // 处理预警  function handleAlert(threatMessage) {  hilog.warn('AlertHandler', 0x0013, `高风险威胁检测到：${threatMessage}`);  // 执行响应措施（如关闭相关服务）  // 实际操作可根据需求定制  }  // 启动预警处理  function startAlertHandling() {  let threatCount = 0;  // 模拟读取威胁分析结果并触发预警  setInterval(() => {  // 若威胁数量超过阈值，触发告警  if (threatCount > ALERT\_THRESHOLD) {  handleAlert('潜在的安全威胁已达到阈值，采取措施');  }  }, 10000); // 每10秒检查一次  }  startAlertHandling(); |

**6. 测试数据生成模块**

**代码分析**

该模块通过模拟正常和异常日志生成数据，供数据采集模块使用。正常日志模拟正常操作，异常日志模拟攻击行为（如SQL注入、权限提升等）。

**完整代码**

|  |
| --- |
| // 测试数据生成模块  function generateTestData() {  const testData = [  { Timestamp: new Date().toISOString(), LogLevel: 'INFO', Component: 'System', Event: 'NormalEvent', Message: '正常操作' },  { Timestamp: new Date().toISOString(), LogLevel: 'ERROR', Component: 'System', Event: 'ErrorEvent', Message: 'Unauthorized access detected' }, // 攻击日志  { Timestamp: new Date().toISOString(), LogLevel: 'ERROR', Component: 'System', Event: 'ErrorEvent', Message: 'SQL injection attempt detected' } // 攻击日志  ];  // 输出日志数据（模拟采集）  return testData;  } |

**7. 模块协作**

通过前述模块，整个框架协同工作，确保 **OpenHarmony** 系统能够进行实时安全评估：

* **数据采集模块** 持续收集日志数据。
* **特征提取模块** 从收集到的日志中提取有用特征。
* **威胁分析模块** 基于提取的特征分析潜在威胁。
* **预警模块** 在检测到威胁时触发告警，并执行自动化响应。

整个流程确保框架能够持续监控系统日志，分析潜在安全威胁，并提供即时反馈和自动防御措施。