AntiDebug от Indy

ReferenceInformation - Development

Author	Last Editor	Created	Last Updated	Version
galenkane wasm.in		24 сент. 2024	24 сент. 2024	1.00

Description

Цели и задачи:

1. Основная цель:
Разработка системы защиты программного обеспечения от отладки и анализа.

2. Ключевые задачи:
- Реализация методов обнаружения отладчиков и средств анализа
- Противодействие отладке и реверс-инжинирингу
- Обеспечение целостности кода и данных защищаемого приложения
- Маскировка критически важных участков кода
- Усложнение статического и динамического анализа

3. Дополнительные цели:
- Обеспечение совместимости с 32- и 64-разрядными системами Windows
- Минимизация влияния на производительность защищаемого приложения
- Возможность гибкой настройки уровня защиты

Описание компонентов:

1. cb.asm - Основной файл проекта

Назначение:
- Инициализация и координация работы всех компонентов защиты
- Реализация ключевых алгоритмов anti-debug
- Управление потоком выполнения защищаемого приложения

Основные функции:
- Setup: инициализация среды выполнения и настройка защитных механизмов
- Entry: точка входа и основной цикл работы системы защиты
- RtGetTicks, SvGetTicks: получение и верификация системного времени
- RtAtom, SvAtom: проверка атомарности критических операций
- RtMsgBox: защищенный вывод сообщений

2. Avl.inc - Реализация АВЛ-дерева

Назначение:
- Эффективное хранение и поиск данных для работы антиотладочных механизмов

Signature

Ключевые структуры:
- RTL_AVL_TABLE: основная структура АВЛ-дерева
- RTL_BALANCED_LINKS: узел дерева

Основные операции:
- AVL_InitializeGenericTable: инициализация дерева
- AVL_InsertElementGenericTable: вставка элемента
- AVL_LookupElementGenericTable: поиск элемента

3. Crc.inc - Реализация алгоритмов CRC

Назначение:
- Вычисление контрольных сумм для проверки целостности кода и данных

Основные функции:
- Crc32: расчет CRC32 для произвольных данных
- Crc32S: расчет CRC32 для строк
- Crc32D: расчет CRC32 для 32-битных значений

4. Gate.asm - Реализация шлюза между 32- и 64-разрядным кодом

Назначение:
- Обеспечение взаимодействия между компонентами в различных режимах работы процессора

Ключевые функции:
- GtCall: вызов функций ядра из 32-разрядного кода
- RtCall: возврат из callback-функций
- NtGateInit: инициализация шлюза
- NtCrcToId: преобразование CRC в идентификаторы системных вызовов

5. Head.inc - Определение ключевых структур и констант

Назначение:
- Централизованное хранение общих определений, используемых в проекте

Основные структуры:
- ENV: глобальное окружение системы защиты
- TLS: локальное хранилище потока
- NTMM: сервисы управления памятью

6. Id.inc - Таблица идентификаторов системных вызовов

Назначение:
- Хранение CRC-хешей имен системных функций для их косвенного вызова

7. log.asm - Модуль логирования

Назначение:
- Запись отладочной информации в файл и вывод через OutputDebugString

Основные функции:
- LgNew: создание нового лог-файла

24 сент. 2024, 06:02

Signature

- LgAdd: добавление записи в лог
- LgPrint: форматированный вывод в лог

8. Mem.asm - Управление памятью

Назначение:
- Реализация низкоуровневых операций с памятью

Ключевые функции:
- MmCrc: расчет CRC для региона памяти
- MmQueryViewSize: определение размера отображенного в память файла
- WsEmpty: очистка рабочего набора процесса
- WsGet: получение информации о рабочем наборе
- MmCheckIp: проверка целостности кода

9. Nt.asm - Взаимодействие с Native API

Назначение:
- Реализация низкоуровневого взаимодействия с ядром Windows

Основные функции:
- NtTree: построение дерева системных вызовов
- NtCrcToId: преобразование CRC в идентификаторы системных вызовов
- NtGateInit: инициализация шлюза Native API
- NtGetUserId: получение идентификатора пользователя

10. Pe.asm - Работа с PE-файлами

Назначение:
- Анализ и модификация исполняемых файлов формата PE

Ключевые функции:
- PeGetNativeBase: получение базового адреса ntdll.dll
- PeEnumExport: перечисление экспортируемых функций
- PeQueryRoutinesFast: быстрый поиск адресов функций по их CRC
- PeMapView: отображение PE-файла в память
- PeRelocate: релокация PE-файла

11. Time.asm - Работа со временем

Назначение:
- Реализация защищенных операций с системным временем

Основные функции:
- TmAccessFile: получение времени доступа к файлу
- TmNewObject: создание объекта с заданным временем
- TmCreateTimer: создание таймера
- TmSetTimer: установка таймера
- TmQueryTimer: опрос состояния таймера
- TmWait: защищенное ожидание

24 сент. 2024, 06:02

Signature

Draft

12. Trace.asm - Отслеживание выполнения

Назначение:
- Реализация механизмов обнаружения трассировки и пошагового выполнения

Ключевые функции:
- TrRpl: проверка уровня привилегий
- Tr86Gs: проверка сегмента GS
- TrCMGs: проверка сегмента GS в режиме совместимости
- Tr86Sexit: проверка SYSENTER
- TrCMSexitFlags: проверка флагов SYSENTER в режиме совместимости

13. Ts.asm - Управление потоками

Назначение:
- Реализация механизмов защиты на уровне потоков

Основные функции:
- TsGetState: получение состояния потока
- TsCleaning: очистка неиспользуемых TLS
- TsAlloc: выделение TLS для нового потока
- TsGet: получение TLS текущего потока

14. Wow.asm - Поддержка WOW64

Назначение:
- Реализация защитных механизмов в среде WOW64

Ключевые функции:
- WwGetSeg: получение 64-битного сегмента кода
- WwCheckGate: проверка шлюза WOW64
- WwIsTurbo: проверка режима Turbo
- WwGetTurboThunk: получение адреса TurboThunk
- WwSetup: настройка окружения WOW64

Аспекты реализации:

1. Использование недокументированных возможностей Windows:
- Прямые вызовы функций Native API
- Доступ к внутренним структурам ядра
- Использование особенностей реализации WOW64

2. Противодействие отладке:
- Обнаружение и нейтрализация популярных отладчиков
- Искажение отладочной информации
- Внедрение ложных точек останова

3. Защита от анализа:
- Динамическое шифрование кода и данных
- Обфускация потока выполнения

- Противодействие дизассемблированию

4. Проверки целостности:
- Расчет контрольных сумм критических участков кода
- Верификация системного окружения
- Отслеживание модификаций защищаемого приложения

5. Противодействие инструментации:
- Обнаружение и блокировка популярных средств анализа
- Противодействие внедрению кода
- Защита от перехвата системных вызовов

6. Устойчивость к виртуальным машинам:
- Обнаружение популярных средств виртуализации
- Противодействие анализу в изолированной среде

7. Самомодификация и полиморфизм:
- Генерация уникального кода при каждом запуске
- Динамическое изменение алгоритмов защиты

8. Использование аппаратных особенностей:
- Применение недокументированных инструкций процессора
- Использование особенностей работы кэшей и предсказания переходов

Заключение:

Представленный проект реализует комплексную систему защиты программного обеспечения от отладки и анализа. Используя широкий спектр техник противодействия исследованию, система значительно усложняет задачу реверс-инжиниринга защищаемого приложения. Модульная архитектура и возможность тонкой настройки позволяют применять данное решение для защиты различных типов программного обеспечения.

24 сент. 2024, 06:02

Signature