Дежавю

Вопросы по дисциплине «Защита программ и данных»

1. Обоснование необходимости анализа программ.

Актуальность задачи анализа программных реализаций обусловлена следующими факторами:

1. Компьютеризация всех областей национальной экономики России.
2. Многообразие программных средств обработки информации и используемых в них средств защиты.
3. Большой разброс в уровне подготовленности разработчиков.
4. Широкое распространение программных средств зарубежного производства.
5. Регулярное появление новых версий программных продуктов, которые могут отличаться средствами защиты.

2. Метод экспериментов с «черным ящиком».

Программа рассматривается как «черный ящик», преобразующий входные данные. Аналитик проводит эксперименты, манипулируя входными данными и анализируя результаты.

Особенности:

1. Эффективен для простых алгоритмов.
2. Не зависит от программной реализации.

Трудности:

1. Нет гарантий соответствия угаданного алгоритма действительности.
2. Проблемы с анализом сложных форматов данных.

Примеры экспериментов:

1. Анализ формата заголовков бинарных файлов.
2. Проверка наличия марканта (случайно генерируемое число или строка, используемое в криптографических системах для затруднения выявления одинаковых фрагментов в потоке зашифрованных данных) в криптосистеме.
3. Определение типа криптографического преобразования (поточный/блочный шифр).

3. Статический метод.

Статический метод основан на анализе исполняемых файлов программы без её запуска. Основной инструмент — **дизассемблеры**, которые преобразуют машинный код в ассемблерный листинг.
Достоинства:

1. Позволяет полностью восстановить алгоритмы защиты (при отсутствии защиты от дизассемблирования).
2. Не зависит от сложности алгоритмов.

Недостатки:

1. Проблема восстановления символических имен.
2. Сложность различения команд и данных.
3. Ошибки в определении границ машинных команд.

Классификация дизассемблеров:

- «Глупые» (dumb): Быстрые, но неточные (встроенные в отладчики).
- «Умные» (smart): Точные, но медленные (IDA Pro).

4. Программные отладочные средства динамического метода.

Основной инструмент — отладчики (OllyDbg, WinDbg), использующие:

1. **Флаг трассировки (TF)**: Пошаговое выполнение.
2. **Точки останова**: Остановка на заданных командах.
3. **Отладочные регистры**: Аппаратные точки останова.

Возможности отладчиков:

1. Пошаговый проход программы.
2. Установка условных точек останова.
3. Просмотр и изменение регистров, памяти.

Ограничения:

1. Невозможность установки точек останова в ПЗУ.
2. Риск зацикливания.
3. Защита программ от отладки.

5. Методика изучения программ динамическим методом.

Анализ программы динамическим методом включает три этапа:

1. Поиск подходов к интересующим функциям

1. **Метод маяков**: Установка точек останова на системные вызовы (например, обращения к файловой системе).
2. **Метод Step-Trace**: Пошаговый проход программы с чередованием режимов Step (без входа в функции) и Trace (с входом).

2. Анализ потоков данных

- **Метод аппаратной точки останова**: Отслеживание изменений в буферах памяти.
- **Метод Step-Trace второго этапа**: Поиск функций, изменяющих интересующие данные.

3. Анализ функций

1. Изучение дизассемблированных листингов.
2. Проверка гипотез о работе алгоритмов через тестовые программы.

Преимущества динамического метода:

1. Целенаправленный анализ.
2. Возможность наблюдения состояния памяти и регистров.
3. Комбинация с другими методами (статическим и «черным ящиком»).

6. Особенности анализа оверлейных программ.

Оверлейные программы загружают в оперативную память только те фрагменты кода, которые выполняются в данный момент, освобождая память от неиспользуемых частей. Это усложняет анализ, так как точки останова, установленные в удаленных фрагментах, теряются. Для решения проблемы рекомендуется устанавливать точку останова в диспетчере оверлеев, который никогда не выгружается из памяти. Это позволяет отладчику восстанавливать потерянные точки останова при каждом изменении расположения кода.

7. Особенности анализа графических программ Windows.

Графические программы Windows возвращают управление операционной системе после инициализации и получают его только при обработке сообщений. Для анализа таких программ модифицируют метод Step-Trace:

1. Определяют адрес оконной функции с помощью утилиты Spy++.

2. Устанавливают точку останова в оконную функцию, но не в начало, чтобы избежать срабатывания на нерелевантные сообщения (например, WM_COMMAND).
3. Трассировка начинается с этой точки останова.

Для диалоговых окон используют альтернативные подходы, например, установку точек останова на функции создания диалогов (например, DialogBoxParamW).

8. Анализ параллельного кода.

Параллельные программы могут выполнять один и тот же код в нескольких потоках, что приводит к хаотичным остановкам при трассировке. Рекомендации:

1. Удалять все точки останова перед началом трассировки, чтобы контролировать только один поток.
2. Использовать команду Disable для временного отключения точек останова.
3. Избегать установки точек останова на функции, используемые несколькими потоками.

Некоторые отладчики (например, WinDbg) поддерживают точки останова для конкретных потоков, но их работа может быть некорректной.

9. Особенности анализа кода в режиме ядра Windows.

Анализ кода ядра требует специальных отладчиков, таких как Syser или WinDbg. Особенности:

1. **Syser** перехватывает функции ядра и приостанавливает систему при работе.
2. Рекомендуется запускать Syser в виртуальной машине для удобства перезагрузки.
3. Условные точки останова могут работать некорректно.
4. Код ядра выполняется в контексте разных процессов, что усложняет анализ.
5. Для драйверов точки останова следует устанавливать после их загрузки, но до выполнения.

Анализ требует осторожности, так как ошибки могут привести к невозможности загрузки системы.

10. Вспомогательные инструменты анализа программ.

1. **ProcMon**: Мониторит обращения к дискам, создание процессов, загрузку библиотек и использование ресурсов. Позволяет фильтровать события и анализировать стеки вызовов.

2. **Process Explorer**: Показывает детальную информацию о процессах, включая открытые объекты, потоки и стек вызовов. Позволяет приостанавливать процессы, управлять приоритетами и закрывать хэндлы.
3. **Spy++**: Определяет адреса оконных функций и свойства окон.
4. **Антивирусные мониторы**: Используются для выявления вредоносного поведения.

Эти инструменты помогают быстро получать информацию о работе программы и выявлять подозрительные активности.

11. Классификация систем защиты программного обеспечения. Системы, устанавливаемые на скомп илированные модули ПО;

Системы, устанавливаемые на скомпилированные модули ПО наиболее удобны для производителя ПО, так как легко можно защитить уже полностью готовое и оттестированное ПО, а потому и наиболее популярны. В то же время стойкость этих систем достаточно низка (в зависимости от принципа действия СЗ), так как для обхода защиты достаточно определить точку завершения работы "конверта" защиты и передачи управления защищенной программе, а затем принудительно ее сохранить в незащищенном виде

12. Классификация систем защиты программного обеспечения. Системы, встраиваемые в исходный код ПО до компиляции;

Системы, встраиваемые в исходный код ПО до компиляции неудобны для производителя ПО, так как возникает необходимость обучать персонал работе с программным интерфейсом (API) системы защиты с вытекающими отсюда денежными и временными затратами. Кроме того, усложняется процесс тестирования ПО и снижается его надежность, так как кроме самого ПО ошибки может содержать API системы защиты или процедуры, его использующие. Но такие системы являются более стойкими к атакам, потому что здесь исчезает четкая граница между системой защиты и как таковым ПО.

13. Классификация систем защиты программного обеспечения. Комбинированные (Алгоритмы запутывания)

Самыми стойкими к атакам являются комбинированные системы.

Алгоритмы запутывания - используются хаотические переходы в разные части кода, внедрение ложных процедур - "пустышек", холостые циклы, искажение количества

реальных параметров процедур ПО, разброс участков кода по разным областям ОЗУ и т. п.

Рассматриваемый алгоритм может быть реализован в виде полиморфных преобразований кода, когда преобразование модифицируемого кода не является взаимно однозначным, т. е. после упаковки и последующей распаковки кода получается код, не идентичный оригиналу, но выполняющий те же самые действия. Перечислим некоторые наиболее простые полиморфные преобразования:

1. **Вставка "пустышек"**: добавление бесполезных команд (например, `nop`).
2. **Ложные условные переходы**: вставка команд `jnz` с заведомо ложными условиями.
3. **Замена команд синонимами**: например, `mov eax, ebx` заменяется на эквивалентную последовательность `push ebx; pop eax`.
4. Замена регистров и (или) локальных переменных, используемых командами.

Эффект: усложняет статический анализ и дизассемблирование.

14. Классификация систем защиты программного обеспечения. Комбинированные (Алгоритмы мутации)

Алгоритмы мутации - создаются таблицы соответствия операндов-синонимов и замена их друг на друга при каждом запуске программы по определенной схеме или случайным образом, случайные изменения структуры программы.

Алгоритм мутации может быть реализован с использованием искусственного усложнения структуры программы:

1. **Косвенные вызовы функций**: вызов через указатели (`call [pFunction]`).
2. вызов функции в отдельном потоке;
3. вызов функции через пул потоков;
4. вызов функции через передачу некоторому окну нестандартного сообщения;
5. вызов функции по таймеру (предполагается, что текущий поток программы не создает никаких окон);
6. вызов функции через асинхронный ввод-вывод;
7. использование нестандартных способов сравнения данных, например, вместо стандартных машинных команд сравнения (cmp, test) можно использовать арифметические и логические команды (add, sub, and, or и т.д.).

15. Классификация систем защиты программного обеспечения. Комбинированные (Алгоритмы компрессии данных)

Алгоритмы компрессии данных - программа упаковывается, а затем распаковывается по мере выполнения. Реализуется за счет использования динамического изменения кода программы.

Метод заключается в хранении основного кода программы в искаженном виде внутри исполняемого файла, а его восстановлении в нормальный вид только в оперативной памяти во время выполнения. При дизассемблировании корректно распознаётся лишь распаковщик, а остальной код воспринимается как данные.

Постоянное динамическое изменение кода делает программу оверлейной, скрывая большую часть кода от отладчиков и обеспечивая защиту как от статического, так и от динамического анализа. Защита усиливается, если используются несколько алгоритмов модификации кода с случайным выбором при упаковке, а также случайное размещение распакованных фрагментов в памяти.

16. Классификация систем защиты программного обеспечения. Комбинированные (Алгоритмы шифрования данных)

Алгоритмы шифрования данных - программа шифруется, а затем расшифровывается по мере выполнения.
В более сложных защитах динамическое архивирование кода дополняется шифрованием.
Ключ шифрования может быть жестко фиксированным либо поступать из какого-то внешнего источника.

17. Классификация систем защиты программного обеспечения. Комбинированные (Методы затруднения отладки)

Методы затруднения отладки - используются различные приемы, направленные на усложнение отладки программы.

Программа может обнаружить тот факт, что она выполняется под отладчиком, с использованием следующих методов:

1. **Использование штатной функции IsDebuggerPresent из библиотеки kernel32.dll** - эта функция возвращает TRUE, если программа выполняется под отладчиком использующим штатный интерфейс отладки, поддерживаемый Windows, и FALSE в противном случае.
2. **Проверка контрольных сумм участков кода, которые не должны изменяться в ходе обычного выполнения программы** - когда отладчик размещает в проверяемом участке кода программную точку останова, контрольная сумма изменяется.

3. **Отслеживание длительности выполнения тех или иных участков кода программы** - под отладчиком код выполняется медленнее, что можно выявить.
4. **Навязывание отладчику ложных точек останова** - намеренно вызывают остановку программы, мешая анализу.
5. **Засорение консоли отладчика многократными вызовами системной функции OutputDebugString**, выдающей в консоль отладчика длинные строки, например, оскорбительного характера
6. **Выявление конкретных отладчиков по косвенным признакам** - например, отладчик SoftICE обычно создает устройства с именами NTICE и SIWVID, большинство отладчиков пользовательского режима имеют характерные заголовки окон и т.п.
7. **Если программный продукт включает в себя более одного процесса, один из этих процессов может выступать в роли отладчика для другого процесса** - если программа использует несколько процессов, один может отлаживать другой, блокируя сторонние отладчики.
8. **Использование программных ошибок конкретных отладчиков** - если отлаживаемая программа искусственно создает ситуацию, в которой некоторый популярный отладчик работает некорректно, данную программу будет трудно отлаживать с помощью данного отладчика.

18. Схема противостояния технических методов и средств защиты

19. Достоинства и недостатки основных систем защиты.

Основные системы защиты и их характеристики:

1. **Упаковщики/шифраторы**:
   Достоинства: Высокая защита от анализа, увеличение стойкости других СЗ.
   Недостатки: Замедление работы, сложности при обновлении, уязвимость к распаковке.
2. **Системы защиты от копирования**:
   Достоинства: Затруднение нелегального копирования.
   Недостатки: Трудоёмкость реализации, замедление продаж, несовместимость с аппаратурой.
3. **Системы защиты от НСД**:
   - *Парольные защиты*: Просты, но уязвимы к взлому.

- *Системы "привязки" ПО*: Не требуют аппаратных средств, но подвержены ложным срабатываниям.
- *Программно-аппаратные ключи*: Высокая стойкость, но дороги и неудобны для пользователей.

20. классификация СЗ от НСД по принципу функционирования системы Парольной защиты

Парольные системы защиты ПО - самые распространённые. Их принцип - идентификация и аутентификация пользователя через ввод пароля, регистрационного кода или дополнительных данных (например, имени или фирмы). Запрос данных может происходить при запуске, установке, регистрации или по окончании пробного периода.

Проверка пароля обычно сводится к сравнению введённого значения с правильным, хранящимся в коде или вычисляемым из данных (например, через хэш-функции). Хранение хэшей повышает стойкость защиты. Однако анализ кода позволяет найти пароль или обойти проверку, особенно если защита не использует шифрование.

Шифрующие системы надёжнее, но при простых или ошибочных алгоритмах возможна дешифровка. Угроза перехвата пароля при вводе остаётся, а однократная проверка при установке или регистрации часто делает защиту уязвимой к несанкционированному копированию ПО.

Положительные стороны:

1. Надежная защита от злоумышленника-непрофессионала.
2. Минимальные неудобства для пользователя.
3. Возможность передачи пароля/кода по сети.
4. Отсутствие конфликтов с системным и прикладным ПО и АО.
5. Простота реализации и применения.
6. Низкая стоимость.

Отрицательные стороны:

1. Пользователю необходимо запоминать пароль/код

21. классификация СЗ от НСД по принципу функционирования Системы "привязки" ПО

При установке ПО на ПК система защиты ищет уникальные признаки компьютера или устанавливает их самостоятельно. Модуль защиты настраивается на их идентификацию для определения авторизованного использования. Используются параметры процессора, материнской платы, устройств, ОС, энергонезависимой памяти, скрытых файлов и др.

Слабость таких систем в том, что ПО всегда запускается на ПК, что позволяет сохранить программу, исследовать защиту и выявить данные для аутентификации.

Положительные факторы:

1. Не требуется добавочных аппаратных средств для работы защиты.
2. Затруднение несанкционированного доступа к скопированному ПО.
3. Простота применения.
4. "Невидимость" СЗПО для пользователя.

Отрицательные факторы:

1. Ложные срабатывания СЗПО при любых изменениях в параметрах ПК.
2. Низкая стойкость при доступе злоумышленника к ПК пользователя.
3. Возможность конфликтов с системным ПО.

22. классификация СЗ от НСД по принципу функционирования Программно-аппаратные средства защиты ПО с электронными ключами

Этот класс средств защиты ПО становится всё более популярным среди разработчиков. Электронный ключ - это аппаратное устройство с памятью и иногда микропроцессором, размещённое в корпусе и подключаемое к ПК через стандартные порты (USB, COM, LPT, PCMCIA) или слоты расширения. Также в этой роли могут использоваться смарт-карты.

Существует два типа электронных ключей:

1. **Ключи с памятью (без микропроцессора)** - наименее стойкие, хранят критическую информацию (например, ключ дешифрации) в памяти. Такие системы менее надёжны, так как злоумышленник, имея аппаратную часть, может перехватить данные, например, через перехват обмена между ПО и ключом.
2. **Ключи с микропроцессором** - наиболее стойкие, содержат не только память, но и аппаратные блоки шифрования/дешифрования. При работе защиты данные передаются в зашифрованном виде, а обратно - в расшифрованном. Это значительно усложняет перехват ключа, поскольку все операции выполняются внутри устройства.

Несмотря на высокую стойкость таких систем, остаётся риск принудительного сохранения защищённого ПО в открытом виде после завершения работы защиты. Кроме того, к этим системам применимы методы криптоанализа.

Положительные факторы:

1. Значительное затруднение нелегального распространения и использования ПО.
2. Избавление производителя ПО от разработки собственной системы защиты.
3. Высокая автоматизация процесса защиты ПО.
4. Наличие API системы для более глубокой защиты.
5. Возможность легкого создания демо-версий.
6. Достаточно большой выбор таких систем на рынке.

Отрицательные факторы:

1. Затруднение разработки и отладки ПО из-за ограничений со стороны СЗ.
2. Дополнительные затраты на приобретение СЗ и обучение персонала.
3. Повышение системных требований из-за защиты (совместимость, драйверы).
4. Снижение отказоустойчивости ПО.
5. Несовместимость систем защиты и системного или прикладного ПО пользователя.
6. Несовместимость защиты и аппаратуры пользователя.
7. Ограничения из-за несовместимости электронных ключей различных фирм.
8. Снижение расширяемости компьютерной системы.
9. Затруднения или невозможность использования защищенного ПО в переносных и блокнотных ПК.
10. Наличие у аппаратной части размеров и веса (для COM/LPT = 5×3×2 см ~ 50 г).
11. Угроза кражи аппаратного ключа.

23. Показатели эффективности систем защиты.

1. **Защита как таковая** – затруднение нелегального копирования и доступа, защита от мониторинга, отсутствие логических брешей и ошибок в реализации системы.
2. **Стойкость к исследованию/взлому** – применение стандартных механизмов, новые/нестандартные механизмы.
3. **Отказоустойчивость (надежность)** – вероятность отказа защиты (НСД), время наработки на отказ, вероятность отказа программы защиты (крах), время наработки на отказ, частота ложных срабатываний.
4. **Независимость от конкретных реализаций ОС** – использование недокументированных возможностей, "вирусных" технологий и "дыр" ОС.
5. **Совместимость** – отсутствие конфликтов с системным и прикладным ПО, отсутствие конфликтов с существующим АО, максимальная совместимость с разрабатываемым АО и ПО.
6. **Неудобства для конечного пользователя ПО** – необходимость и сложность дополнительной настройки системы защиты, доступность документации, доступность информации об обновлении модулей системы защиты из-за ошибок/несовместимости/нестойкости, доступность сервисных пакетов, безопасность

сетевой передачи пароля/ключа, задержка из-за физической передачи пароля/ключа, нарушения прав потребителя.
7. **Побочные эффекты** – перегрузка трафика, отказ в обслуживании, замедление работы защищаемого ПО и ОС, захват системных ресурсов, перегрузка ОЗУ, нарушение стабильности ОС.
8. **Стоимость** – стоимость/эффективность, стоимость/цена защищаемого ПО, стоимость/ликвидированные убытки.
9. **Доброкачественность** – доступность результатов независимой экспертизы, доступность информации о побочных эффектах, полная информация о СЗ для конечного пользователя.

24. Описание субъектно-ориентированной модели компьютерной

Субъектно-ориентированная модель компьютерной системы, предложенная А.Ю. Щербаковым, рассматривает систему как конечное множество объектов, среди которых выделяются субъекты. Субъекты обладают свойством активности, например, процессы, в отличие от пассивных объектов, таких как файлы. Основные элементы модели:

1. **Объекты (O)**: Пассивные элементы системы (файлы, устройства).
2. **Субъекты (S)**: Активные элементы (процессы, пользователи).
3. **Поток информации (Stream)**: Операция, при которой субъект изменяет состояние объектов, используя информацию из других объектов.
4. **Ассоциирование (S(O))**: Связь между субъектом и объектом, означающая, что субъект использует информацию объекта.
5. **Порождение субъекта (Create)**: Создание нового субъекта из объекта.
6. **Доступ**: Чтение или запись объекта субъектом, включая создание и удаление объектов.

Модель делит доступы на санкционированные (L) и несанкционированные (N), где монитор безопасности (M) контролирует выполнение операций Stream.

25. Описание НСД на основе субъектно-ориентированной модели

Несанкционированный доступ (НСД) в рамках модели возникает по двум причинам:

1. **Некорректное функционирование монитора безопасности (M)**: Ошибки в реализации M, позволяющие выполнить запрещённые операции.
2. **Некорректное описание множества L**: Несоответствие формальных правил доступа реальным требованиям безопасности.

Программные закладки реализуют опосредованный НСД:

1. Автономно изменяют объекты или порождают новые субъекты.
2. Маскируют своё присутствие, используя легальные политики безопасности.

Пример: Расширение множества L через модификацию M, что позволяет включать ранее запрещённые доступы.

26. Модели взаимодействия программной закладки с атакуемой системой. Модель «наблюдатель»

Программная закладка встраивается в атакуемую систему в качестве отдельного субъекта доступа S, реагирующего на активизирующие события.

Как правило, программные закладки данного класса поддерживают следующие функции удаленного контроля:

1. управление файловыми системами;
2. управление конфигурацией системы;
3. управление процессами.

По некоторому активизирующему событию (сигналу) закладка инициирует нетипичный информационный поток, либо моделирует сбойную ситуацию. На практике применяется для скрытного удаленного контроля операционных систем.
Обычно состоит из двух частей:

1. серверной (устанавливаемой на атакуемом компьютере)
2. клиентской (управляющей сервером удаленно).

Клиент посылает команды серверу, сервер выполняет их и возвращает результаты. Часто сервер содержит средства скрытия своего присутствия.

В отличие от обычных средств удаленного администрирования, программные закладки позволяют несанкционированный контроль.

Известные представители: Back Orifice и NetBus, которые имеют различия в платформенной поддержке и размере.

Недостатки современных средств скрытного удаленного контроля:

1. уязвимость к антивирусам;
2. наличие бесполезных функций;
3. отсутствие взаимодействия с подсистемой безопасности Windows;
4. большой размер серверной части.

Дополнительные задачи программных закладок модели «наблюдатель»: первоначальный доступ к системе, внедрение и управление другими закладками, передача перехваченной информации, противодействие обнаружению.

27. Модели взаимодействия программной закладки с атакуемой системой. Модель «перехват»

Программная закладка встраивается в программное обеспечение субъекта, обслуживающего информационные потоки определенного класса. Информационные потоки, проходящие через программную закладку, полностью или частично дублируются.

Типы перехватчиков:
1. Клавиатурные фильтры:

1. **1-й род**: Имитация приглашения входа (сохранение пароля). Имитируют приглашение для ввода пароля. Сохраняют введённые данные и завершают сеанс, заставляя пользователя повторить ввод. Пример: поддельное окно входа в систему.
2. **2-й род**: Внедряются в обработчики клавиатурного ввода. Перехватывают все нажатия клавиш или фильтруют пароли. Пример: модификация драйвера клавиатуры.
3. **3-й род**: Внедрение в подсистему аутентификации. Перехватывают только данные, связанные с аутентификацией. Пример: подмена библиотеки `PAM` (Pluggable Authentication Modules).

2. Мониторы файловых систем: Перехватывают операции чтения/записи критичных файлов (Файлы пользователей, журналы аудита, конфигурации безопасности).
Проблема: Высокий объём данных требует избирательного мониторинга.

3. Мониторы сети: Перехват сетевых пакетов (аутентификация, ключевые слова).

Монитор сети может быть основан на одном из следующих принципов:

1. перехват информационных потоков, проходящих через сетевое программное обеспечение операционной системы;
2. перехват информации, передаваемой по общей шине сегмента локальной сети (прослушивание сегмента).

28. Модели взаимодействия программной закладки с атакуемой системой. Модель «искажение» Несанкционированное использование средств динамического изменения полномочий

> **Вопросы 28-30**
>
> Программная закладка встраивается в программное обеспечение, обслуживающее информационные потоки определенного класса, чаще всего в код системных программных модулей, используемое всеми субъектами доступа.

Такие закладки модифицирует информационные потоки или политики безопасности

Пример: Подмена М на модифицированную версию, расширяющую права нарушителя.

В ОС иногда требуется временное повышение прав пользователя (например, для смены пароля). Для этого используются механизмы динамического изменения полномочий, но их уязвимости могут привести к несанкционированному повышению прав.

UNIX:

1. Полномочия процесса определяются EUID и EGID.
2. Если в атрибутах исполняемого файла установлены биты SUID/SGID, процесс запускается от имени владельца файла.
3. Злоумышленник может создать файл с SUID/SGID, владельцем которого является пользователь с повышенными правами (например, командный интерпретатор).
4. Для защиты требуется либо админский доступ, либо эксплуатация уязвимости в ПО.

Windows:

1. Используется механизм *олицетворения (impersonation)*: серверный процесс временно понижает права потока до уровня клиента.
2. Без этого серверу пришлось бы вручную проверять права клиента для каждого объекта, что повышает риск ошибок.

Чаще всего такие методы применяются для эксплуатации уязвимостей без внедрения вредоносного кода.

29.
Модели взаимодействия программной закладки с атакуемой системой. Модель «искажение» Порождение дочернего процесса системным процессом
и Модификация машинного кода монитора безопасности объектов.

Порождение дочернего процесса системным процессом:
Системные процессы в ОС имеют почти неограниченные полномочия, которые наследуют их дочерние процессы. Злоумышленник может создать дочерний процесс от имени системного, запустив командный интерпретатор или другую программу, чтобы

получить доступ к объектам ОС, создавать новые процессы и выполнять другие действия.

Модификация машинного кода монитора безопасности объектов:
Программная закладка может изменять код монитора безопасности, добавляя функции, которые дают злоумышленнику несанкционированные права. Фактически, это подмена оригинального монитора на модифицированный.
Также модель «искажение» может использоваться для скрытого взаимодействия: закладка изменяет ПО так, что определённые данные обрабатываются как управляющие команды.

30.
Модели взаимодействия программной закладки с атакуемой системой. Модель «искажение» Стелс-технологии

Стелс-технологии применяются в модели «искажение» для скрытия программных закладок от штатных средств обнаружения. Такие закладки выявляются лишь из-за ошибок (несовместимость с обновлениями ОС) или аномалий (высокая загрузка ЦП, сетевой трафик).

Для полноценного скрытия требуется работа в режиме ядра. Закладка либо сама является драйвером, либо содержит упакованный **стелс-драйвер**, который:

1. **Скрывает файлы** – перехватывает функции поиска (например, `NtQueryDirectoryFile` в Windows, `getdents` в Linux), удаляя записи о своих файлах из вывода.
2. **Скрывает записи реестра** – аналогично файлам, но перехватывает функции работы с реестром.
3. **Скрывает процессы** – модифицирует результаты вызовов `NtQuerySystemInformation`, удаляя данные о своих процессах.
4. **Скрывает сетевые порты** – перехватывает функции драйвера TCP/IP.
5. **Скрывает сервисы** – изменяет данные диспетчера сервисов (`services.exe`) и связанные ключи реестра (может работать без драйвера).

31. Предпосылки к внедрению программных закладок.

Программные закладки могут быть внедрены в компьютерную систему, если:

1. Множество несанкционированных доступов NN определено так, что существует совокупность информационных потоков PP, приводящая к внедрению закладки, и P∩N=∅P∩N=∅.

2. Монитор безопасности MM не обеспечивает корректное блокирование информационных потоков из множества NN.

На практике абсолютная защита невозможна из-за:

- Ошибок администраторов.
- Низкой надежности программного обеспечения, включая средства защиты.

32. уязвимости ПО: переполнения буферов;

Программа выделяет буфер недостаточной длины или не проверяет длину при записи данных.

1. **Пример**: Пользователь вводит строку, превышающую размер буфера, что приводит к затиранию данных в памяти, включая адрес возврата.
2. **Результат**: Возможность выполнения произвольного кода, аварийное завершение программы.
3. **Известные случаи**: Переполнение буфера в Internet Explorer, JPEG-обработке в Microsoft, RPC в Windows XP (MSBlast).

Методы защиты:

- Системы обнаружения вторжений (СОВ),
- Защита стека (контроль целостности перед возвратом из функции),
- Использование безопасных библиотек и абстрактных типов данных,
- DEP (Data Execution Prevention), ASLR (Address Space Layout Randomization).

33. уязвимости ПО: отсутствие необходимых проверок входных данных;

Любые данные, поступающие на вход программы, должны проверяться на корректность. Если эта проверка выполняется недостаточно тщательно или не выполняется вообще, нарушитель может передать в программу заведомо некорректные данные, обработка которых программой может нарушить ее функциональность и привести к нарушению политики безопасности.

Примеры:

1. **GetAdmin**: Ядро Windows NT не проверяло адреса, передаваемые в функцию `NtAddAtom`, что позволяло записывать данные в системную память.
2. **%00 (нулевой байт)**: В Internet Explorer v 5.5, строка `prog.exe%00some.html` интерпретировалась как `prog.exe`, что позволяло запускать исполняемые файлы.

34. уязвимости ПО: Зиспользование некорректного контекста безопасности в ходе выполнения тех или

иных функций;

Средства динамического изменения полномочий пользователей традиционно являются уязвимыми в операционных системах. В Windows динамическое изменение реализовано через динамическое понижение полномочий, что надежнее динамического повышения, принятого в UNIX. Механизм олицетворения пользователя в Windows устойчивее к ошибкам, чем SUID/SGID в UNIX, но все же содержит уязвимости.

Примеры уязвимостей:

1. **AdminTrap (1997)**: эксплойт создает сервер NPFS, маскирующийся под стандартный, и при подключении пользователя с административными правами получает такие же полномочия, разрывая соединение.
2. **Атака PipeBomb**: программа в бесконечном цикле открывает новые экземпляры системного канала и записывает в них бессмысленные данные, что вызывает создание множества потоков и потребление памяти процессом-сервером. Это приводит к 100% загрузке процессора, исчерпанию памяти и зависанию системы.
3. **Уязвимости системных окон**: в Windows окна не являются объектами доступа ОС, минимальным объектом доступа является рабочий стол. Приложение может посылать сообщения другим окнам на том же рабочем столе, что позволяет вызвать некорректное поведение системных приложений и повысить права пользователя. Такие уязвимости известны для Norton Antivirus, Agnitum Outpost Firewall и других. В Windows Vista введен мандатный контроль целостности, который ограничил передачу сообщений из пользовательских окон в системные, снизив риск.

35. уязвимости ПО: использование устаревших функций системы, ранее бывших безобидными, но ставших опасными в новых условиях

При обновлении программного продукта часть кода переносится без изменений, но некоторые участки, безопасные в старой версии, могут стать уязвимыми из-за изменений в окружении.

Примеры уязвимостей:

1. **NetDDE Exploit (2002)**: В Windows NT 4.0 сервер NetDDE запускался с правами локального пользователя, что не представляло угрозы. В Windows 2000 он стал сетевым сервисом с правами SYSTEM и автоматически стартует с системой. Сервер поддерживает запуск произвольных программ по команде клиента через сообщение WM_COPYDATA. В Windows 2000 это позволяет любому пользователю запускать программы с правами SYSTEM, что ведет к несанкционированному повышению полномочий.

2. **WMF Exploit (MS06-001, 2006)**: Формат графических файлов WMF поддерживает встроенный машинный код, который выполняется при ошибках отображения файла. Это позволило вирусу Sober запускать вредоносный код при просмотре файла. Уязвимость возникла из-за устаревшего формата, в котором не предполагалась возможность исполнения кода. Эксплойты на основе WMF позволяют повышать права пользователя до SYSTEM.

36. Методы внедрения программных закладок: маскировка под прикладноеПО

> **Вопросы 36-39**
>
> Первоначальное внедрение программной закладки - самый сложный и рискованный этап её жизненного цикла, так как именно на этом шаге большинство закладок обнаруживаются. После внедрения их выявить значительно сложнее.
>
> С точки зрения субъектно-ориентированной модели, внедрение закладки сводится либо к созданию нового субъекта в системе (модель «наблюдатель»), либо к изменению реакции существующих субъектов на информационные потоки. В первом случае создаётся объект - источник нового субъекта, во втором - изменяются объекты, связанные с уже существующими субъектами.

Внедрение программной закладки в атакуемую систему сводится к инсталляции новой прикладной программы, что является тривиальной процедурой, не требующей от разработчика закладки каких-либо специальных приемов программирования.

Существует модификация данного метода, когда закладка внедряется в атакуемую систему под видом не программы, а файла или буфера с данными, содержащих внутри себя исполняемый код — машинный или интерпретируемый.

Примеры данных, внутри которых может внедряться программная закладка:

1. документы Microsoft Office, содержащие макросы, активизирующие закладку;
2. документы в формате HTML (в том числе и письма электронной почты), содержащие скрипты, активизирующие закладку;
3. любые данные, обрабатываемые программой, имеющей уязвимости, позволяющие активизировать исполняемый код, несанкционированно внедренный в данные.

37. Методы внедрения программных закладок: маскировка под системное ПО

Этот метод отличается тем, что закладка маскируется под системную программу (сервис, драйвер, библиотеку и т.д.), которая обычно обладает широкими правами в

ОС. Поэтому такие закладки имеют больше возможностей, чем закладки, замаскированные под прикладные программы.

Системная программа (СПЗ) - новый субъект ОС, который может влиять на информационные потоки, если активируется субъектом, инициирующим операцию. Однако без специальных мер СПЗ не может самостоятельно перехватывать или искажать потоки, что ограничивает эффективность метода для моделей «перехват» и «искажение».

Для внедрения новой системной программы нужны высокие права (обычно администратора), что является недостатком метода. Кроме того, закладку может обнаружить администратор при проверке системных программ, хотя это можно усложнить с помощью стелс-технологий.

В Windows этот метод часто реализуется через несанкционированное создание сервиса: закладка сначала запускается как обычная программа, затем устанавливает и запускает себя как сервис, после чего завершает работу. В Windows 2003 и ранее для этого достаточно административных прав, а в Windows Vista и новее требуется выполнение с повышенным уровнем целостности.

38. Методы внедрения программных закладок подмена системного ПО

В атакуемой ОС выбирается исполняемый файл, который заменяется программной закладкой, полностью повторяющей функции оригинала. При выборе модуля учитываются:

1. Возможности воздействия закладки на систему;
2. Степень документированности функций модуля (чем лучше, тем проще подмена);
3. Сложность и время подмены (перезагрузка и др.);
4. Маскировка факта подмены (аудит, сбои и т.п.).

Основная проблема - невозможно гарантировать полную реализацию всех функций большого или плохо документированного модуля. Кроме того, закладка может быть удалена при обновлении системы.

В многозадачных ОС подмена работающего модуля затруднена из-за контроля совместного доступа к файлам. Для замены нужно завершить работу модуля и перезапустить систему с закладкой или использовать уязвимости ОС.

В многопользовательских ОС с разграничением доступа запись в системные файлы разрешена только администраторам, поэтому для подмены требуются административные права.

В Windows начиная с 2000 года подмена системных модулей практически невозможна из-за встроенной системы контроля целостности WFP.

39. Методы внедрения программных закладок: прямое и косвенное ассоциирование

Прямое ассоциирование:
Метод заключается в присоединении программной закладки к файлу по вирусному алгоритму, частично изменяя существующий файл. Это частный случай подмены программного модуля, поэтому имеет схожие плюсы и минусы.

Особенности метода:

1. Не требуется реализовывать все функции модуля, что упрощает разработку;
2. Оставляет заметные следы, которые могут обнаружить эвристические антивирусы.

В целом, достоинства и недостатки прямого ассоциирования примерно компенсируют друг друга.

Косвенное ассоциирование:
Закладка внедряется в оперативную память после загрузки программы, не изменяя исполняемый файл на диске. Это затрудняет обнаружение традиционными средствами, так как на диске следов нет.

Недостаток - после каждой перезагрузки закладку нужно внедрять заново.

В многозадачных ОС с изолированием адресных пространств косвенное ассоциирование возможно только при доступе закладки к памяти целевой программы.

40. Требования, предъявляемые к вирусам

1. **Скрытность**: избегание обнаружения антивирусами.
2. **Живучесть**: автоматический запуск после перезагрузки.
3. **Распространение**: заражение других файлов/систем.
4. **Функциональность**: выполнение вредоносных действий (кража данных, DDoS и т. д.).

41. Методы защиты от программ закладок (общие принципы защиты)

Все методы защиты делятся на 2 группы:

1. Методы, основанные на применении штатных средств
2. Методы, основанные на применении специализированных средств для выявления, предупреждения и пресечения разрушительных воздействий программ закладок.

Политика безопасности комп. систем в отношении встраиваемых закладок базирующихся на 2 основных принципах:

1. Минимизация используемого ПО
2. Минимизация полномочий/привилегий пользователя - заключается в использовании изолированной программной среды, чем сложнее правила доступа, тем сложнее их использовать

42. Требования к средствам защиты

Не смотря на существующие адекватные политики безопасности, поддерживать их в течение длительного времени неукоснительным соблюдением достаточно сложно (почти невозможно), по этой причине необходимо и используется доп. средства защиты

Основные требования к средствам защиты:
1. Общие требования:

1. эффективность;
2. надёжность;
3. простота реализации;
4. простота администрирования

2. Требования к средствам защиты:

1. Сохранность эксплуатационных качеств системы - защита системы не должна вносить изменения в свойства эксплуатационной системы
2. Противодействие активным воздействия программ закладок
3. Эшелонированность - свойства защитной системы должны сохраняться при выходе её элементов из строй (сохранность свойств, при значительных изменениях состояния элементов)

43. сканирование системы на предмет наличия известных программ закладок;

При сканировании систем используется 2 метода:

1. Сигнатурный – поиск вредоносных программ по известным шаблонам (сигнатурам) в коде или структуре файлов. Недостаток - надо пополнять постоянно пополнять базу сигнатур. Варианты: проверка всей системы или проверка на лету.
2. Эвристический – анализ поведения и характеристик программ для выявления подозрительной активности, даже если точная сигнатура неизвестна. (Для работы программ закладок, как правило исп. специфические виды написания кодов, и их можно эвристически вычислить)

44. контроль целостности программного обеспечения;

Данный метод заключается в следующем на моменте создания ПО осуществляется подсчёт КС и определяется размер файла, он кодируется защищённым кодом, подписывается и хранится в спец. файле. Периодически происходит проверка сумм.

В связи с тем, что изменения программ и обновления, нам приходится часто осуществлять проверки КС. Сканируем либо всё целиком либо, только используемый ПП.

45. контроль целостности конфигурации защищаемой системы;

Данный метод основан на следующем: для всех переменных конфигурации защищаемой системы, которые могут быть изменены с помощью программ закладок, создаются эталонные копии и периодически эти эталонные копии сравниваются с элементами конфигурации.

Любые виды изменений в системе ведёт к изменении конфигурации => необходимо создание новых эталонных копий, кроме того для всех элементов конфигурации нужно проверять не появились ли новые элементы конфигурации системы.

46. антивирусный мониторинг информационных потоков;

Данный метод основан на том, что любая программа закладка так или иначе вмешивается в информационные потоки. Эти вмешательства могут быть обнаружены с помощью соответствующих мониторов.

Наиболее эффективен мониторинг в тех инф. потоках, которые наиболее часто связаны с применением программ закладок:

1. Потоки, связанные с паролями пользователей
2. Потоки, связанные с обращениями к файловой системе
3. Потоки, связанные с обращениями к сетевым ресурсам