Фонд оценочных средств для промежуточной аттестации

по дисциплине

Защита программ и данных

Направление подготовки (специальность)
10.05.01 Компьютерная безопасность

Направленность (профиль)
специализация № 1 «Анализ безопасности компьютерных систем»

Присваиваемая квалификация
специалист по защите информации

Форма обучения
очная

Год набора 2026

Челябинск 2026 г.

Содержание

1. ПАСПОРТ ФОНДА ОЦЕНОЧНЫХ СРЕДСТВ

Специальность 10.05.01 Компьютерная безопасность.
Специализация № 1 «Анализ безопасности компьютерных систем».
Дисциплина: **Защита программ и данных**.
Семестр (семестры) изучения: 7 семестр.
Форма (формы) промежуточной аттестации: экзамен (7 семестр).
Используется балльно-рейтинговая система для оценивания результатов.

2. ПЕРЕЧЕНЬ ФОРМИРУЕМЫХ КОМПЕТЕНЦИЙ

2.1. Компетенции, закреплённые за дисциплиной

Изучение дисциплины «Защита программ и данных» направлено на формирование следующих компетенций:

Коды компетенции согласно ФГОС (ОПОП ВО)	Содержание компетенций согласно ФГОС (ОПОП ВО)	Индикаторы достижения компетенции согласно ОПОП	Перечень планируемых результатов обучения по дисциплине
1	2	3	4
ОПК-13	Способен разрабатывать компоненты программных и программно-аппаратных средств защиты информации в компьютерных системах и проводить анализ их безопасности	ОПК-13.1.2 знает основные средства и методы защиты программного обеспечения от анализа; ОПК-13.1.3 знает основные типы уязвимостей программного обеспечения; ОПК-13.2.1 умеет использовать программные средства анализа защиты компьютерных систем; ОПК-13.2.2 умеет разрабатывать компоненты средств защиты информации.	Знать: – особенности программирования шеллкодов; – методы исследования программного обеспечения без исходных кодов. Уметь: – создавать шеллкоды для современных операционных системы под разные аппаратные платформы; – исследовать программное обеспечение без исходных кодов. Владеть: – навыками создания шеллкодов с учетом специфики различных сценариев использования; – навыками использования современных средств исследования программного обеспечения без исходных кодов.
ОПК-16:	Способен проводить мониторинг работоспособности и анализ	ОПК-16.1.1 знает средства мониторинга работоспособности средств защиты информации в компьютерных системах и сетях ОПК-16.2.1 знает методики	Знать: – базовые методы функционирования вредоносного программного обеспечения; – методы защиты программного

	эффективности средств защиты информации в компьютерных системах и сетях;	анализа эффективности средств защиты информации в компьютерных системах и сетях ОПК-16.2.2 умеет использовать средства мониторинга работоспособности средств защиты информации в компьютерных системах и сетях	обеспечения от исследования, копирования, модификации. Уметь: – реализовывать базовые функциональные компоненты вредоносного программного обеспечения; – реализовывать методы защиты программного обеспечения от исследования с учетом специфики операционных систем, аппаратной платформы, используемой архитектуры. Владеть: – навыками исследования вредоносного программного обеспечения с использованием современных инструментов анализа и собственных утилит; – навыками реализации методов защиты программного обеспечения от исследования и обхода этих методов.

3. СОДЕРЖАНИЕ ОЦЕНОЧНЫХ СРЕДСТВ ПО ДИСЦИПЛИНЕ

3.1 Виды оценочных средств

№ п/п	Код компетенции / планируемые результаты обучения	Контролируемые темы/ разделы	Наименование оценочного средства для текущего контроля	Наименование оценочного средства на промежуточной аттестации/№ задания
1.	ОПК-1.1 ОПК-1.2	Раздел 1. Исследование программного обеспечения без исходных кодов	Зачетные задания. Домашние задания. Аудиторные задания.	Вопросы на экзамене. Задания на экзамене.
2.	ОПК-1.1 ОПК-1.2	Раздел 2. Защита кода в Windows	Зачетные задания. Домашние задания. Аудиторные задания.	Вопросы на экзамене. Задания на экзамене.
3.	ОПК-1.1 ОПК-1.2	Раздел 3. Защита кода в Linux	Зачетные задания. Домашние задания. Аудиторные задания.	Вопросы на экзамене. Задания на экзамене.

Типовые задания, критерии и показатели оценивания в рамках текущего контроля представлены в рабочей программе дисциплины (модуля). Полные комплекты оценочных средств и контрольно-измерительных материалов хранятся на кафедре.

3.2 Содержание оценочных средств

3.2.1 Темы для домашних, аудиторных, зачетных заданий и заданий на экзамене

1. Статическое исследование программного кода.
2. Динамическое исследование программного кода.
3. Исполняемые файлы.
4. Вредоносное программное обеспечение.
5. Шеллкоды.

3.2.2 Примеры зачетных заданий

I. С использованием интерфейса Windows Debug API и библиотеки udis86 реализовать отладчик приложений под Windows с возможностью дизассемблирования.

Базовая функциональность (к сдаче не принимаются задания с функциональностью меньше базовой), 10 баллов.

Требования:

1) Интерфейс пользователя должен позволять:

- дизассемблировать инструкции по указанному адресу;
- выставлять программные точки останова по указанному адресу;
- выводить/изменять регистры;
- выводить/изменять память;
- трассировать по шагам (через флаг трассировки).

2) Достаточно поддержки одной из сред: Win32 или Win64.

Полная функциональность, 20 баллов.

Требования (дополнительно к базовым):

1) Интерфейс должен быть графическим.

2) Должна быть поддержка аппаратных точек останова (всех четырёх).

3) Соответствующие сборки отладчика должны уметь отлаживать 32-х и 64-х битные программы (можно написать два отладчика, но логичней будет писать единый код с использованием директив условной компиляции).

Если одна сборка отладчика будет одновременно поддерживать 32-х и 64-х битные программы, самостоятельно определяя разрядность, то +5 баллов.

4) Должна быть возможность указывать в командах (дизассеблирования, точек останова и других) вместо адресов имена, которые доступны в таблицах экспорта загруженных в отлаживаемый процесс модулей. +5 баллов.

II. Реализовать шеллкод с функциональностью бэкдора и руткита пользовательского уровня под Windows.

Базовая функциональность (к сдаче не принимаются задания с функциональностью меньше базовой), 10 баллов.

Требования:

1) Внедряться в существующие процессы (любым способом).

2) Предоставлять по сети возможность исполнения команд:

- загрузка/выгрузка файлов;
- листинг каталогов;
- запуск программ.

Полная функциональность.

Требования (дополнительно к базовым):

1) Внедрение без выделения памяти (5 баллов).

2) Внедрение без порождения потока (5 баллов).

3) Корректное внедрение в порождаемые процессы. Защита от повторного внедрения. (до 10 баллов)

4) Скрытие процессов (5 баллов), поддержка удалённой команды для скрытия процесса по PID'у/имени (+5 баллов).

5) Скрытие файлов (5 баллов), поддержка удалённой команды для скрытия файлов (+5 баллов).

6) Скрытие сетевых соединений (5 баллов), поддержка удалённой команды для скрытия сетевых соединений (+5 баллов).

III. Реализовать файловый вирус, осуществляющий заражение 32-х и/или 64-х разрядных PE-файлов.

Базовая функциональность (к сдаче не принимаются задания с функциональностью меньше базовой), 10 баллов.

Требования:

1) Реализовать шеллкод, осуществляющий внедрение кода в PE-файл (32-х или 64-х битный) без нарушения его работоспособности. При запуске зараженной программы сначала должен отрабатывать внедрённый код, как-то сигнализировать о своем выполнении (эмуляция полезной нагрузки) и передавать управление оригинальному коду.

Полная функциональность.

Требования (дополнительно к базовым):

1) Должен быть предусмотрен механизм самораспространения (т.е. запущенный из заражённой программы внедрённый код должен внедрять

себя в другой исполняемый файл) и защита от повторного заражения (до 10 баллов).

2) Заражение должно быть максимально незаметным (см. указания) (до 10 баллов).

3) Возможность заражения 32-х и 64-х разрядных файлов (после перекомпиляции под нужную разрядность) (5 баллов).

4) Корректно обрабатывать релоки (до 5 баллов).

5) В качестве тестовой полезной нагрузки реализовать обращение по сети к какому-либо узлу (5 баллов).

IV. Реализовать шеллкод с функциональностью бэкдора и руткита пользовательского уровня под Linux.

Базовая функциональность (к сдаче не принимаются задания с функциональностью меньше базовой), 15 баллов.

Требования:

1) Внедряться в существующие процессы.

2) Предоставлять по сети возможность исполнения команд:

- загрузка/выгрузка файлов;
- листинг каталогов;
- запуск программ.

Полная функциональность.

Требования (дополнительно к базовым):

1) Поиск адресов библиотечных функций (сканированием адресного пространства или с помощью анализа файла cat /proc/self/maps найти адрес загрузки libc и проанализировать таблицу экспорта) (10 баллов). Без этого пункта адреса функций для перехвата (для следующих пунктов) можно задавать статически (считаем, что рандомизация отключена: sysctl kernel.randomize_va_space=0).

2) Корректное внедрение в порождаемые процессы (см. указания). Защита от повторного внедрения. (до 10 баллов)

3) Скрытие процессов (5 баллов), поддержка удалённой команды для скрытия процесса по PID'у/имени (+5 баллов).

4) Скрытие файлов (5 баллов), поддержка удалённой команды для скрытия файлов (+5 баллов).

5) Скрытие сетевых соединений (5 баллов), поддержка удалённой команды для скрытия сетевых соединений (+5 баллов).

V. С использованием интерфейса Windows Debug API и библиотеки

udis86 реализовать инструмент динамического исследования бинарного кода. Соответствующие сборки программы должны поддерживать 32-х и 64-х битное окружение.

Некоторые задания предполагают логирование в файл. Для каждого задания необходимо заводить отдельный файл, имя которого должно формироваться из имени программы, идентификатора процесса, идентификатора потока и времени начала логирования.

Возможно, что задания будут мешать друг другу, поэтому необходимо иметь возможность произвольно включать/выключать функциональность отдельных заданий.

Для тестирования найти или написать программы, которые позволят продемонстрировать весь реализованный функционал.

n - номер студента в списке.

a=19

b=11

Базовая функциональность (к сдаче не принимаются задания с функциональностью меньше базовой).

15 баллов.

Требования:

1. Должна быть возможность запускать указанную программу под отладкой и присоединяться к указанному процессу.

2. 32-х битная сборка программы должна отлаживать 32-х разрядные программы. 64-х битная сборка программы должна отлаживать 64-х разрядные программы. Соответственно везде, где это необходимо, должна обеспечиваться (скорее всего с помощью директив условной компиляции) поддержка двух разрядностей: для кода, для регистрового контекста, для PE-формата и т.д.

3. Должна быть поддержка многопоточных программ: вся реализуемая функциональность должна выполняться для всех существующих и вновь порождаемых потоков.

4. С помощью отладочного интерфейса реализовать трассировку (с использованием флага трассировки) исследуемой программы на уровне отдельных инструкций. Необходимо отслеживать отдельные инструкции. Необходимо выбрать из приведенных ниже наборов инструкций один с номером (n*a+b)%18. В документации необходимо найти все инструкции, относящиеся к набору, для всех возможных разрядностей. В потоке всех трассируемых инструкций необходимо находить (придумать как это сделать) все инструкции из набора (остальные инструкции должны

игнорироваться). Для каждой найденной инструкции сохранять в лог-файле запись со следующей информацией: дизассемблированная инструкция, регистровый контекст.

Наборы инструкций:

0) Все инструкции mov.

1) Все строковые инструкции: movs, cmps, scas, lods, stos.

2) Все инструкции call ближнего вызова функции. Все инструкции jmp безусловного перехода.

3) Все инструкции условного перехода.

4) Все инструкции ret, retn возврата из ближнего вызова функции.

5) Все инструкции sub/add.

6) Все инструкции mul/imul.

7) Все инструкции div/idiv.

8) Все инструкции xor/or/and.

9) Все инструкции cmp/test.

10) Все инструкции int/sysenter/syscall.

11) Все инструкции push/pop.

12) Все инструкции pusha/popa/pushf/popf.

13) Все инструкции rcl/rcr.

14) Все инструкции rol/ror.

15) Все инструкции sal/sar.

16) Все инструкции shl/shr.

17) Все инструкции xadd/xchg.

Дополнительные задания.

1. В случае реализации функциональности помимо базовой это дополнительное задание обязательно для исполнения.

Из списка ниже необходимо выбрать один набор с индексом (n*a+b)%9. С помощью программных точек останова необходимо перехватывать все функции из набора. Необходимо выставить точки останова на нужные функции, отлавливать их срабатывание, при срабатывании правильно обрабатывать. Необходимо придумать корректный способ выполнения инструкции, перетёртой точкой останова. Выполнение перетертой инструкции можно эмулировать. Либо можно останавливать все потоки, восстанавливать перетертую инструкцию, делать один шаг трассировки, восстанавливать точку останова, запускать потоки.

Функции для перехвата находятся в системных библиотеках kernel32.dll, ntdll.dll. Эти библиотеки загружены во все процессы по одним адресам. Поэтому адреса функций можно получать в процессе отладчика с

помощью GetProcAddress и эти адреса будут корректны в отлаживаемых процессах.

Необходимо придумать способ получения управления после возврата из функции (для получения возвращаемого значения и выходных аргументов).

Необходимо анализировать (для вывода) аргументы функции: входные при срабатывании точки останова, выходные - после возврата из функции. Анализ аргументов должен быть глубоким (т.е. с разыменованием указателей, разбором структур, учётом типов): например, для указателя на строку должна быть выведена строка, для массива (указателя на массив) - все элементы этого массива, для набора битовых флагов - имена выставленных флагов, для последовательности структур - все структуры в последовательности, для хендлов - имена и т.д. Должен быть настраиваемый предел максимального количества выводимых символов строки и элементов массива.

Если какой-то аргумент (или совокупность аргументов) определяет интерпретацию других аргументов (например, как аргумент FileInformationClass в функции ZwQueryInformationFile), то необходимо в зависимости от значения этого аргумента осуществлять различный разбор зависимых аргументов (в соответствии с описанием в документации - недокументированные коды и структуры не обрабатывать).

Разбор структур должен осуществляться на два уровня вложенности: если аргумент является указателем на структуру, то выводить все поля этой структуры в соответствии с их типом, если в этой структуре есть другая структура (как подструктура или указатель на структуру), то выводить и её поля в соответствии с их типом.

Т.е. необходимо выводить максимально полную доступную информацию о произошедшем вызове. Но для упрощения глубина вложенности ограничена двумя (только это может являться ограничением для получения полной информации).

Выводить в лог-файл следующую информацию: имя функции, аргументы функции (разобранные как сказано выше), возвращаемое значение, адрес возврата. Предложить и реализовать максимально наглядный вариант форматирования этой информации.

Наборы функций:

0) CreateFileA/CreateFileW, OpenFileA/OpenFileW, CreateFileMappingA/CreateFileMappingW, CloseHandle.

1) MapViewOfFile, MapViewOfFileEx, UnmapViewOfFile, ReadFile,

WriteFile.

2) VirtualAlloc, VirtualAllocEx, VirtualFree, VirtualFreeEx, VirtualQuery, VirtualQueryEx, VirtualProtect, VirtualProtectEx.

3) HeapCreate, HeapDestroy, HeapAlloc, HeapFree, HeapSize, HeapReAlloc, HeapQueryInformation.

4) ZwQueryInformationFile

5) ZwQuerySystemInformation

6) ZwQueryDirectoryFile

7) CreateProcessA/CreateProcessW, CreateProcessAsUserA/CreateProcessAsUserW, ExitProcess, TerminateProcess.

8) CreateThread, ExitThread, CreateRemoteThread, OpenThread, GetThreadContext, SetThreadContext, SuspendThread, ResumeThread.

До 10 баллов.

+5 баллов за унифицированный формат хранения информации о типах и функциях и обработку аргументов в одной общей функции без ограничения уровня вложенности.

Из следующих дополнительных заданий необходимо выбрать не более двух заданий (нельзя одновременно выбирать 2 и 3 задания).

2. Статическое определение функций программы. Трассировка на уровне функций (с помощью точек останова).

Придумать алгоритм, который позволит при инициализации с помощью дизассемблирования найти все (по возможности) функции программы (в кодовых секциях PE-файла программы). Поставить точки останова на все функции и все точки выхода из функций. Придумать эвристический алгоритм поиска инструкции вызова функции при срабатывании точки останова в её начале. При заходе в функцию выводить в лог-файл следующую информацию: инструкцию вызова и её адрес, адрес функции. При выходе из функции выводить в лог-файл следующую информацию: возвращаемое значение, адрес возврата.

До 10 баллов.

3. Трассировка (с помощью флага трассировки) с динамическим определением функций программы.

Функции находить эвристически: трассировать все инструкции, по инструкции call находить начало функции (в PE-файле программы), по инструкции ret - конец. Придумать эвристический алгоритм, позволяющий определить количество аргументов функции (предполагать нотации cdecl, stdcall, fastcall для x64). При заходе в функцию выводить в лог-файл следующую информацию: инструкцию вызова и её адрес, адрес функции,

аргументы функции. При выходе из функции выводить в лог-файл следующую информацию: возвращаемое значение, адрес возврата.

До 10 баллов.

4. Трассировка (с помощью точек останова) библиотечных функций.

Найти все загруженные в отлаживаемый процесс библиотеки (эта информация предоставляется через отладочные сообщения). Разобрать таблицы экспорта этих библиотек и найти все экспортируемые функции. Поставить точки останова на все экспортируемые функции. Придумать эвристический алгоритм раскручивания стека для получения stack trace (предполагать нотации cdecl, stdcall, fastcall для x64) с адресами возврата, возможными аргументами и содержимым стекового фрейма. При срабатывании точки останова выводить в лог-файл следующую информацию: имя библиотеки, имя функции, адрес функции, stack trace.

До 10 баллов.

5. Проверка корректности работы с динамической памятью.

Перехватывать функции работы с динамической памятью (malloc, free, VirtualAlloc, HeapAlloc и др.) с помощью программных точек останова. Придумать и реализовать техники (с помощью перехвата этих функций и, возможно, модификации их поведения) детектирования ошибок работы с памятью:

- отсутствие проверки возвращаемого значения при выделении памяти;
- неправильный указатель при освобождении памяти;
- обращение к памяти после освобождения;
- двойное освобождение памяти;
- обращение за пределами выделенного блока (перед ним и после него);
- утечка памяти.

До 10 баллов.

Пример кода для определения номеров:

```
a = 19
b = 11
for n in range(1, 18):
    print str(n) + ' ' + str((n*a+b)%18) + ' ' + str((n*a+b)%9)
```

VI. Реализовать файловый вирус, осуществляющий заражение 32-х и 64-х разрядных исполняемых и библиотечных PE-файлов и/или ELF-файлов.

При заражении не должна нарушаться работоспособность файла.

Вся функциональность должна располагаться во внедряемом коде. Не

допускается использование дополнительных файлов, программ и библиотек вируса.

Необходимо продемонстрировать всю реализованную функциональность на практике. Для этого найти или создать подходящие файлы.

n - номер студента в списке.

a=19

b=11

Там, где не сказано отдельно, описание подразумевает оба формата (PE, ELF). Для упрощения описание ведётся в терминах PE-формата (подразумевается, что соответствующие сущности есть в ELF, но с другими именами).

Базовая функциональность (к сдаче не принимаются задания с функциональностью меньше базовой).

20 баллов.

Требования:

1. Реализовать шеллкод, осуществляющий простое внедрение кода в PE- или ELF-файл без нарушения его работоспособности.

Расширить последнюю секцию, добавить ей право на исполнение. В новое место поместить весь шеллкод (никакие данные программы не должны измениться). Перенаправить точку входа на шеллкод. При запуске зараженной программы сначала должен отрабатывать внедрённый код, показывать некоторое сообщение (эмуляция полезной нагрузки) и передавать управление оригинальному коду.

2. Возможность заражения 32-х и 64-х разрядных файлов (после перекомпиляции под нужную разрядность).

Дополнительные задания.

Некоторое дополнительные задания местами не совместимы с базовой функциональностью. Тогда при выборе такого дополнительного задания соответствующий пункт из базовой функциональности не делать, а делать более сложную функциональность из дополнительного (баллы за задание суммируются с баллами за базовую).

Некоторые задания не совместимы между собой.

Для выбора доступны только следующие последовательности дополнительных заданий.

1 -> 2 -> 4

1 -> 2 -> 5

1 -> 3(подзадания 0, 1) -> 6

1 -> 3(подзадания 2, 3) -> 4

1 -> 3(подзадания 2, 3) -> 5

Т.е. сначала можно выбрать только задание 1. После него можно выбрать только задания 2 или 3. После задания 2 можно выбрать (произвольно) только задания 4 или 5. После задания 3 можно выбрать (в завимости от подзадания) только задания 4, 5 или 6.

1. Должен быть предусмотрен механизм самораспространения, т.е. запущенный из заражённой программы внедрённый шеллкод должен внедрять себя в другой исполняемый файл.

Должна быть предусмотрена защита от повторного заражения.

Способы пометить зараженный файл (при заражении в определённые места записывается предопределённая сигнатура):

0) Сигнатура в неиспользуемых полях DOS-заголовка.

1) Сигнатура в DOS-stub.

2) Сигнатура в неиспользуемых полях файлового NT-заголовка.

3) Сигнатура в неиспользуемых полях опционального NT-заголовка.

4) Сигнатура в именах секций

5) Сигнатура в неиспользуемой памяти в конце секции.

Необходимо выбрать способ с номером (a*n+b)%6.

До 5 баллов.

2. Вместо простого внедрения кода из базовой части.

Располагать весь внедряемый шеллкод (единым блоком) по смещению (317*(n+b))%100 от начала секции кода. Оригинальный код, на место которого внедряется шеллкод, необходимо куда-то перенести. Для этого нужно обеспечить новое место в файле. В этом новом месте стоит располагать только переносимые исходные данные файла.

Методы выделения дополнительной памяти:

0) Расширить (в файле и в оперативной памяти) последнюю секцию.

1) Добавить новую секцию.

2) Добавить в файл после первой секции дополнительное место (размер никакой секции при этом не меняется), сдвинув все лежащие после данные и поправив соответствующие файловые смещения. Данные из этой памяти можно получить только чтением из файла с диска.

3) Добавить в файл после второй секции дополнительное место (размер никакой секции при этом не меняется), сдвинув все лежащие после данные и поправив соответствующие файловые смещения. Данные из этой памяти можно получить только чтением из файла с диска.

4) Добавить в файл после третьей секции дополнительное место

(размер никакой секции при этом не меняется), сдвинув все лежащие после данные и поправив соответствующие файловые смещения. Данные из этой памяти можно получить только чтением из файла с диска.

Необходимо выбрать метод с номером (a*n+b)%5.

До 7 баллов.

3. Вместо простого внедрения кода из базовой части.

Обеспечить место для внедрения шеллкода одним из следующих методов:

0) Расширить первую секцию (должна быть кодовой) вниз по адресам (изменится абсолютный адрес ее начала и увеличится размер). При этом абсолютные адреса всех существующих в программе данных не изменятся, но ImageBase уменьшится. Значит необходимо соответствующим образом поправить все rva-адреса.

1) Расширить первую секцию (должна быть кодовой) вверх по адресам: её абсолютный адрес начала (и ImageBase) не меняется, но увеличивается размер. Соответственно необходимо подвинуть данные после секции. Также необходимо поправить соответствующие относительные и абсолютные адреса. Для правки абсолютных адресов использовать релоки (не заражать файлы, для которых это сделать невозможно).

Следующие пункты обеспечивают дополнительное место без увеличения размера файла (и размера какой-либо секции), сжимая исходные данные программы (в одной или нескольких секциях). Для сжатия можно использовать, например, библиотеку aPLib или функцию RtlCompressBuffer. Если после сжатия освобождается недостаточно места, то не заражать такой файл.

2) Сжимать данные в секции кода (будут сжиматься относительно плохо). В освободившееся после сжатия место поместить шеллкод.

3) Сжимать данные в секции данных (должны хорошо сжиматься). В освободившееся после сжатия место поместить оргинальный код из кодовой секции, который будет заменён шеллкодом.

Необходимо выбрать метод с номером (a*n+b)%4.

До 15 баллов.

4. При размещении шеллкода в файле необходимо так его располагать, чтобы не перетереть метаданные PE-файла, необходимые при загрузке. Наборы таких метаданных:

0) Директории: экспорта, импорта, локальной памяти потоков (TLS), конфигурации загрузки.

1) Директории: релоков, отложенного импорта, отладки, таблица

обработки исключений.

2) Информация импорта: массив дескрипторов импорта, имена библиотек, имена функций.

3) Информация импорта: массив дескрипторов импорта, массивы IAT (import address table) и INT (import name table).

4) Релоки: директория релоков, адреса, по которым релоки осуществляют поправки.

5) Директория конфигурации загрузки (IMAGE_LOAD_CONFIG_DIRECTORY), таблица безопасных обработчиков исключений.

6) Директория локальной памяти потоков (TLS), массив адресов callback-функций в директории TLS.

Необходимо выбрать набор с номером (a*n+b)%7. Должны быть проверки только метаданных из выбранного набора.

До 7 баллов.

5. При размещении шеллкода в файле необходимо корректно обрабатывать возможную ситуацию перетирания шеллкодом метаданных PE-файла, нужных для загрузки. При заражении шеллкод должен корректно перетирать (может быть обнулять) или модифицировать (может быть переносить) эти метаданные. А после исполнения шеллкод должен восстанавливать в оперативной памяти все перетёртые данные (в том числе и метаданные) и при необходимости их обрабатывать.

Возможные типы метаданных (и способов их обработки):

0) Импорт. Обнулять директорию импорта. После восстановления оригинальных данных обрабатывать её самостоятельно.

До 15 баллов.

1) Импорт. Переносить всю информацию импорта (директорию, массивы, имена функций и т.д.) в дополнительно выделенную память. Все абсолютные ссылки на элементы таблицы адресов импорта (IAT) находить с помощью релоков (если это невозможно, то не заражать такой файл) и править на новые адреса.

До 25 баллов.

2) Экспорт. Переносить всю информацию экспорта (директорию, массивы, имена функций и т.д.) в дополнительно выделенную память.

До 20 баллов.

3) Релоки. Перенести директорию релоков в дополнительно выделенную память.

До 7 баллов.

4) Релоки. Обнулять директорию релоков. После восстановления оригинальных данных обрабатывать её самостоятельно.

До 12 баллов.

Необходимо выбрать метод с номером (a*(n+b))%5.

6. Поддреживать одновременно (без перекомпиляции) в одном шеллкоде заражение 32-х и 64-х битных версий файлов. Соответственно шеллкод должен уметь работать в 32-х и 64-х битном программном окружении.

До 15 баллов.

Указания и рекомендации.

При заражении происходит модификация данных программы. Если модифицированные данные нужны после инициализации (во время исполнения кода), то внедрённый шеллкод может эти данные восстановить - тогда их модификацию никто не заметит. Если данные нужны системному загрузчику во время инициализации (например, данные для импорта), то их порча приведёт к ошибке загрузки файла. Неконтролируемая модификация таких данных не желательна. Поэтому либо надо находить расположение этих данных и их не затрагивать, либо модифицировать эти данные особым образом.

Шеллкод после выполнения полезной нагрузки должен загрузить реальный код по правильному адресу и передать управление настоящей точке входа (OEP).

Релоки, которые правят данные в секции, в которую внедряется шеллкод, не должны его портить при загрузке. Т.е. либо убирать все релоки, либо только те, которые могут испортить код. Соответственно во время исполнения шеллкода и восстанавления реального содержимого секции необходимо обработать релоки самостоятельно (т.е. убранные релоки необходимо сохранить в какой-то внутренней таблице).

Помимо шеллкода необходимо хранить дополнительные данные о зараженном файле:

- адрес оригинальной точки входа;
- адрес, где сохранены данные секции;
- адрес, где расположены перемещенные данные.

Их можно расположить перед шеллкодом. Соответственно шеллкод должен знать, как до них дотянуться.

Во время запуска зараженной программы (и, соответственно, шеллкода) должно происходить заражение других исполняемых файлов (в текущей директории или в какой-то системной директории). Т.е. новый

исполняемый файл должен меняться соответствующим образом и в него должен копироваться шеллкод из текущего.

Файлы не должны заражаться повторно. Для этого зараженный файлы надо как-то помечать (имя секции, определенные значения в неиспользуемых полях заголовков).

Чтобы не вызвать подозрений, файлы не должны заражаться слишком активно.

Первый запуск шеллкода, когда никаких зараженных файлов еще нет, производить с помощью программы execsc2.

Шеллкод отлаживается обычным образом (например, с помощью OllyDbg), начиная с точки входа зараженного файла.

Пример кода для определения номеров:

```
n = 1
a = 11
b = 13
for n in range(1,10):
    print 'student ' + str(n)
    print '\textra task1'
    print '\t\tsignature method: ' + str((a*n+b)%6)
    print '\textra task 2'
    print '\t\toffset: ' + str((317*(n+b))%100)
    print '\t\tmethod: ' + str((a*n+b)%5)
    print '\textra task 3'
    print '\t\tmethod: ' + str((a*n+b)%4)
    print '\textra task 4'
    print '\t\tset: ' + str((a*n+b)%7)
    print '\textra task 5'
    print '\t\tmethod: ' + str((a*(n+b))%5)
```

VII. Реализовать шеллкод с функциональностью бэкдора и руткита пользовательского уровня под Windows.

Шеллкод должен внедряться в процессы без нарушения их работоспособности и выполнять некоторую полезную нагрузку.

n - номер студента в списке (ориентироваться на список с оценками).

a=29

b=17

Базовая функциональность (к сдаче не принимаются задания с функциональностью меньше базовой).

20 баллов + баллы за выбранные задачи.

Вся реализуемая функциональность должна поддерживаться для 32-х разрядных ОС Windows XP, Windows 7, Windows 10 без перекомпиляции (т.е. во время исполнения необходимо определять версию системы и подстраиваться под неё).

Внедряться в существующие процессы (выделять память, копировать шеллкод, порождать поток).

Предоставлять по сети возможность исполнения команд (в каждый момент времени это делает один шеллкод в одном из заражённых процессов) :

0) загрузка файлов на систему с шеллкодом;

1) выгрузка файлов с системы с шеллкодом;

2) вывод списка всех процессов (имена, PID'ы);

3) листинг файлов в указанном каталогов;

4) запуск указанной программы;

5) вывод подключей указанного ключа реестра;

6) вывод параметров (имена) указанного ключа реестра;

7) вывод значения (типа REG_DWORD, REG_SZ) указанного параметра указанного ключа реестра;

8) вывод списка библиотек (имя, адрес загрузки) указанного (по PID'у) процесса.

Необходимо реализовать две команды с номерами (a*n+b)%9 и (((a+n)%13)*n+b*n)%9.

Руткит-функциональность.

Осуществлять перехват (с помощью сплайсинга) системных вызовов (в библиотеке ntdll.dll) и модифицировать возвращаемую ими информацию.

Ниже представлены 4 вида задач. Для каждого вида перечислены конкретные задачи. Необходимо выбрать от 0 до 2 задач (не больше одной из каждого вида). Доступная для выбора задача для i-го (i от 0 до 3) вида определяется следующим образом.

Пусть N - число задач этого вида.

Задача - (a*n+b*i)%N.

За каждую задачу 10 баллов.

Виды задач:

0. Модификация списка процессов, возвращаемого системным вызовом ZwQuerySystemInformation

Задачи:

0) Скрытие процесса (по PID'у и по имени).

1) Добавление в список (несуществующего) процесса с указанными именем и PID'ом.

2) Изменение у указанного (по PID'у и по имени) процесса имени на указанное.

3) Вместо оригинального списка процессов возвращать список процессов с указанными PID'ами и именами.

1. Модификация списка сетевых соединений IPv4, возвращаемого сетевым драйвером через системный вызов ZwDeviceIoControlFile.

Задачи:

0) Скрыть сетевое соединение (по номеру локального порта, по номеру удаленного порта, по номерам локального и удаленного портов).

1) Добавить в список сетевое соединение с указанными данными.

2) Изменить у указанного сетевого соединения (по номеру локального порта, по номеру удаленного порта, по номерам локального и удаленного портов) локальные и удалённые адреса и порты на указанные.

3) Вместо оригинального списка сетевых соединений возвращать список с указанными данными.

2. Модификация списка файлов, возвращаемого системным вызовом ZwQueryDirectoryFile (в примере драйвера обрабатывались все возможные значения FileInformationClass, для которых возращались имена, здесь можно обрабатывать только те значения, которые используются проводником).

Задачи:

0) Скрыть файл (по имени).

1) Добавить в список для указанной директории файл с указанным именем.

2) Изменить в списке для указанной директории имя файла.

3) Вместо оригинального списка файлов для указанной директории возвращать список с указанными данными.

3. Модификация списка ключей реестра, возвращаемого системным вызовом ZwEnumerateKey.

Задачи:

0) Скрыть ключ реестра (по имени, по пути).

1) Добавить в список для указанного ключа подключ с указанным именем.

2) Изменить в списке для указанного ключа имя подключа.

3) Вместо оригинального списка подключей для указанного ключа возвращать список с указанными данными.

Дополнительные задания.

Выбрать не более двух дополнительных заданий.

1. Внедрение шеллкода без выделения памяти в удалённом процессе внедряющим кодом (т.е. без использования VirtualAllocEx/ZwAllocateVirtualMemory).

Методы расположения шеллкода:

0) Расположить основной шеллкод в области нулей секций (не обязательно исполняемых) библиотек удалённого процесса. В области нулей исполняемых секций (библиотек kernel32.dll, ntdll.dll) расположить небольшой шеллкод, который будет собирать основной шеллкод и передавать ему управление.

10 баллов.

1) Расположить основной шеллкод в файле. В области нулей исполняемых секций (библиотек kernel32.dll, ntdll.dll) расположить небольшой шеллкод, который будет считывать основной шеллкод и передавать ему управление.

10 баллов.

2) Расположить основной шеллкод в директориях экспорта некоторых загруженных библиотек. В области нулей исполняемых секций (библиотек kernel32.dll, ntdll.dll) расположить небольшой шеллкод, который будет собирать основной шеллкод и передавать ему управление. В основном шеллкоде необходимо восстанавливать оригинальное содержимое перетёртых директорий экспорта.

15 баллов.

3) Расположить основной шеллкод вместо информации импорта (дескрипторы, имена функций и библиотек, таблица INT, но не таблица IAT) некоторых загруженных библиотек. В области нулей исполняемых секций (библиотек kernel32.dll, ntdll.dll) расположить небольшой шеллкод, который будет собирать основной шеллкод и передавать ему управление. В основном шеллкоде, при необходимости, восстанавливать оригинальное содержимое перетёртой информации импорта.

15 баллов.

4) Расположить основной шеллкод вместо не используемых после загрузки данных (релоки, DOS-заголовок, DOS-stub и т.д.) некоторых загруженных библиотек. В области нулей исполняемых секций (библиотек kernel32.dll, ntdll.dll) расположить небольшой шеллкод, который будет собирать основной шеллкод и передавать ему управление.

10 баллов.

5) Расположить основной шеллкод вместо ресурсов некоторых загруженных библиотек. В области нулей исполняемых секций (библиотек kernel32.dll, ntdll.dll) расположить небольшой шеллкод, который будет собирать основной шеллкод и передавать ему управление. В основном шеллкоде необходимо восстанавливать оригинальное содержимое перетёртых областей памяти.

15 баллов.

Необходимо выбрать метод с номером (a*n+b)%6.

2. Внедрение шеллкода без выделения памяти в удалённом процессе внедряющим кодом (несовместимо с предыдущим заданием).

Располагать внедряемый шеллкод в области нулей исполняемых секций. Не собирать шеллкод в единый блок, а выполнять так, как он был внедрён - разбитым на отдельные блоки в конце исполняемых секций.

Необходимо предложить и реализовать способ такой организации шеллкода. Каждый оригинальный способ будет принят только у одного сдающего.

20 баллов.

3. При внедрении передавать управление на шеллкод без порождения потока (т.е. без вызова функции CreateRemoteThread).

Способы передачи управления

0) С помощью сплайсинга перехватить функцию возврата из системного вызова (адрес которой в переменной SharedUserData!SystemCallStubRet (0x7ffe0304)). Перенаправить управление на внедрённый шеллкод, который должен сразу же снять перехват.

5 баллов.

1) Приостановить некоторый поток. Изменить его контекст так, чтобы регистр eip стал указывать на внедрённый шеллкод. Шеллкод должен восстановить оригинальный контекст.

7 баллов.

2) Подменить адрес возврата в текущем стековом фрейме некоторого потока.

7 баллов.

3) Подменить адрес первого VEH-обработчика в ntdll!LdrpVectorHandlerList и спровоцировать исключение.

10 баллов.

Необходимо выбрать метод с номером (a*n+b)%4.

4. Корректное внедрение в новые процессы при их создании.

Перехватывать функции создания процесса. Порождать процесс в приостановленном состоянии, внедрять в него шеллкод, перехватывать управление, после чего запускать поток.

До 15 баллов.

5. Реализовать интерфейс, который по сети позволит управлять всеми параметрами реализованной функциональности. Соответственно, необходимо придумать способ передавать эту управляющую информацию от одного шеллкода (который общается по сети) всем остальным.

До 15 баллов.

Указания и рекомандации

По аналоги с примером hooklib из lab/9 реализуемую функциональность сначала можно сделать и протестировать в виде библиотеки, внедряемой в процесс.

Для получения команд шеллкод всегда прослушивает некоторый порт. Исполнять команды должен только один процесс. Тот, кто прослушивает порт. Остальные периодически пытаются начать прослушивать этот порт, чтобы захватить его в случае завершения текущего общающегося процесса.

Для избежания конфликтов внедрять шеллкод в порождаемые процессы должен один процесс. Это может быть тот процесс, который захватил для прослушивания порт.

Библиотеки kernel32.dll и ntdll.dll грузятся во все процессы по одному адресу. Этим можно пользоваться. Другие библиотеки грузятся не во все процессы и могут быть загружены в разные процессы по разным адресам. Поэтому для их разбора необходимо читать память процесса, в который они загружены.

Получить информацию о стеке потока можно из его TEB'a. Получить адрес TEB'a можно с помощью вызова ZwQueryInformationThread.

Пример кода для определения номеров:

```
n = 1
a = 29
b = 17
for n in range(1,20):
    print 'student ' + str(n)
    print '\tcommands: ' + str((a*n+b)%9) + ' ' + str((((a+n)%13)*n+b*n)%9)
    for i in range(0,4):
```

```
        print '\ttype '+str(i)+': ' + 'task '+str((a*n+b*i)%4)
    print '\textra task 1: ' + str((a*n+b)%6)
    print '\textra task 3: ' + str((a*n+b)%4)
```

VIII. Реализовать шеллкод с функциональностью бэкдора и руткита пользовательского уровня под Linux для 32-х разрядных процессоров Intel.

Шеллкод должен внедряться в процессы без нарушения их работоспособности и выполнять некоторую полезную нагрузку.

n - номер студента в списке (ориентироваться на список с оценками).

a=23

b=13

Базовая функциональность (к сдаче не принимаются задания с функциональностью меньше базовой).

20 баллов + баллы за выбранные задачи.

Шеллкод должен отрабатывать как минимум на выданной системе VulnLinux с ядром 3.2.

Внедряться в существующие процессы (для которых хватит прав). Для этого подключаться к процессу в качестве отладчика, получать регистровый контекст, заменять код (предварительно сохранив) по адресу регистра eip на шеллкод. Шеллкод отрабатывает (ставит перехваты, настраивает своё дальнейшее функционирование) и оповещает внедряющий процесс (например, генерацией исключения), после чего на место регистра eip возвращается оригинальный код и восстанавливает сохранённый регистровый контекст.

Все необходимые адреса можно задавать статически (считаем, что рандомизация отключена: sysctl kernel.randomize_va_space=0).

Предоставлять по сети возможность исполнения команд (в каждый момент времени это делает один шеллкод в одном из заражённых процессов) :

0) загрузка файлов на систему с шеллкодом;

1) выгрузка файлов с системы с шеллкодом;

2) вывод списка всех процессов (имена, PID'ы);

3) листинг файлов в указанном каталогов;

4) запуск указанной программы;

5) вывод списка библиотек (имя, адрес загрузки) указанного (по PID'у) процесса;

6) реализовать аналог метода CONNECT протокола SOCKS4 (т.е. создавать сокет соединением с указанными адресом и портом и передавать

данные между этим сокетом и сокетом входящего соединения);

7) предоставлять удалённую оболочку (перенаправлять стандартные дескрипторы в сокет и порождать оболочку);

Необходимо реализовать две команды с номерами (a*n+b)%8 и (((a+n)%19)*n+b*n)%8.

Руткит-функциональность.

Осуществлять перехват (с помощью сплайсинга) системных вызовов (в библиотеке libc) и модифицировать возвращаемую ими информацию.

Ниже представлены 4 вида задач. Для каждого вида перечислены конкретные задачи. Необходимо выбрать от 0 до 2 задач (не больше одной из каждого вида). Доступная для выбора задача для i-го (i от 0 до 3) вида определяется следующим образом.

Пусть N - число задач этого вида.

Задача - (a*n+b*i)%N.

За каждую задачу 10 баллов.

Виды задач:

0. Модификация списка файлов, возвращаемого системными вызовами readdir, getdents, getdents64.

Задачи:

0) Скрыть файл (по имени).

1) Добавить в список для указанной директории файл с указанным именем.

2) Изменить в списке для указанной директории имя файла.

3) Вместо оригинального списка файлов для указанной директории возвращать список с указанными пользователем данными.

1. Модификация информации о процессах: список возвращается при листинге каталога /proc, информация о процессе получается чтением файлов в подкаталоге процесса (с именем равным PID'у) в каталоге /proc.

Задачи:

0) Скрытие процесса (по PID'у, по имени).

1) Изменение у указанного (по PID'у, по имени) процесса имени на указанное пользователем (модификация содержимого при чтении файла cmdline).

2. Модификация списка сетевых соединений, возвращаемого сетевым драйвером при чтении из файла /proc/net/tcp.

Задачи:

0) Скрыть сетевое соединение (по номеру локального порта, по номеру удаленного порта, по номерам локального и удаленного портов).

1) Добавить в список сетевое соединение с указанными пользователем данными.

2) Изменить у указанного сетевого соединения (по номеру локального порта, по номеру удаленного порта, по номерам локального и удаленного портов) локальные и удалённые адреса и порты на указанные пользователем.

3) Вместо оригинального списка сетевых соединений возвращать список с указанными пользователем данными.

3. Модификация списка модулей ядра, возвращаемого при чтении из файла /proc/modules.

Задачи:

0) Скрыть модуль ядра (по имени).

1) Добавить в список модуль с указанными пользователем данными.

2) Изменить в списке для указанного модуля (по имени) имя.

3) Вместо оригинального списка модулей возвращать список с указанными пользователем данными.

Дополнительные задания.

Выбрать не более двух дополнительных заданий.

1. Поиск всех необходимых адресов.

Сканированием адресного пространства или с помощью анализа файла /proc/self/maps найти адреса загрузки всех нужных библиотек и областей памяти (стек, куча, vdso).

Проанализировав загруженные библиотеки, найти адреса нужных функций.

До 20 баллов.

2. Не перетирать оригинальный код в удалённом процессе.

Необходимо найти/выделить свободное место, в которое можно поместить шеллкод. При изменении регистрового контекста остановленного процесса в регистр eip загружать адрес шеллкода.

Методы обеспечения свободной памяти для шеллкода (выбрать один):

1) Выделить память для шеллкода с помощью удалённого вызова mmap (см. указания).

10 баллов.

1) Расположить основной шеллкод в области нулей секций (не обязательно исполняемых) библиотек удалённого процесса. В области нулей исполняемых секций расположить небольшой шеллкод, который будет собирать основной шеллкод и передавать ему управление.

15 баллов.

3. Предложить и реализовать способ передачи управления на шеллкод без изменения регистрового контекста. Каждый оригинальный способ будет принят только у одного сдающего.

До 15 баллов.

4. Корректное внедрение в новые процессы при их создании (см. указания).

До 20 баллов.

5. Реализовать интерфейс, который по сети позволит управлять всеми параметрами реализованной функциональности. Соответственно, необходимо придумать способ передавать эту управляющую информацию от одного шеллкода (который общается по сети) всем остальным.

До 15 баллов.

Указания и рекомендации.

Список процессов следует получать через листинг каталога /proc.

Информацию о загруженных библиотеках можно получить из файла /proc/<pid>/maps. В этом файле хранится информация о всех выделенных диапазонах виртуального адресного пространства в удобном для программного разбора формате.

Для вызова системных вызовов в чужом процессе можно использовать следующий сценарий. Для приостановленного процесса (например, сразу после подключения отладчиком) получить регистровый контекст. Поменять значения регистров: в регистры ebx-ebp поместить аргументы, в eax - номер, в eip - адрес инструкции вызова системного вызова (sysenter, int 0x80). Нужные инструкции можно найти в vdso:

```
__kernel_vsyscall:
  push   ecx
  push   edx
  push   ebp
  mov    ebp,esp
  sysenter
  nop
  nop
  nop
  nop
  nop
  nop
  nop
```

```
int   0x80
pop   ebp
pop   edx
pop   ecx
ret
```

Лучше использовать int 0x80, так как при её использовании возврат из ядра будет происходить по следующему за ней адресу, а при sysenter - по некоторому фиксированному (скорее всего на инструкцию pop ebp за инструкцией int 0x80 в функции __kernel_vsyscall).

По адресу возврата из системного вызова можно поставить точку останова. После чего восстановить регистровый контекст.

Таким образом без выполнения шеллкода можно выполнить значительную функциональность в удалённом процессе.

Для выделения памяти можно воспользовать описанным методом вызова системных вызовов (для mmap). Также можно трассировать системные вызовы (PTRACE_SYSCALL), дождаться вызова mmap и увеличить размер и права выделяемой памяти (эту дополнительную память можно использовать для своих целей).

Чтение/запись памяти удалённого процесса можно осуществлять через ptrace (PTRACE_PEEKTEXT, PTRACE_POKETEXT), файл /proc/<pid>/mem (виртуальные адреса - смещения в этом файле), вызовы process_vm_readv, process_vm_writev.

Для порождения потока необходимо вызвать системный вызов clone (с флагами CLONE_FILES | CLONE_FS | CLONE_IO | CLONE_SIGHAND | CLONE_SYSVSEM | CLONE_THREAD | CLONE_VM). Проще это делать через библиотечный вызов, чем через системный (системный вызов реализован в arch/x86/kernel/process.c).

Для внедрения в новые процессы необходимо перехватить вызов execve (либо семейство exec*). В перехватчике породить новый процесс. В родительском процессе начать отладку дочернего, в котором произойдёт запуск новой программы. Родительский процесс с помощью ptrace (SYSCALL, ...) дожидается загрузки библиотеки libc (libc_fd = open (libc), mmap (..., libc_fd,...)) и внедряет код, который осуществляет перехват функций в libc.

Модуль безопасности Yama вводит ограничение на возможность доступа к другим процессам через ptrace. Считаем, что дополнительные ограничения не активированы (т.е. /proc/sys/kernel/yama/ptrace_scope=0). В любом случае внедряться надо во все процессы, в которые получится.

Пример кода для определения номеров:

```
n = 1
a = 23
b = 13
for n in range(1,20):
    print 'student ' + str(n)
    print '\tcommands: ' + str((a*n+b)%8) + ' ' + str((((a+n)%19)*n+b*n)%8)
    type_sizes = [4, 2, 4, 4]
    for i in range(0,4):
        print '\ttype '+str(i)+': ' + 'task '+str((a*n+b*i)%type_sizes[i])
```

3.2.3 Примеры домашних, аудиторных заданий и заданий на экзамене

1. Дизассемблировать программы и найти правильный логин и пароль.
2. С помощью дизассемблерования и отладки найти правильный код.
3. Реализовать отладчик.
4. Реализовать скрипт для отладчика.
5. Реализовать плагин для отладчика.
6. Реализовать скрипт для дизассемблера.
7. Реализовать плагин для дизассемблера.
8. Реализовать дизассемблер PE-файла.
9. Реализовать функцию, возвращающую адрес загрузки библиотеки kernel32.dll.
10. Реализовать поиск адреса загрузки по адресу внутри некоторого модуля.
11. Написать шеллкоды.
12. Реализовать трассировщик под Linux.
13. Реализовать перехват функций в процессах под Linux.

3.2.4 Примеры вопросов на экзамене

1. Дизассемблирование. Дизассемблеры. Скрипты и плагины для дизассемблеров.
2. Приемы антидизассемблирования.
3. Отладка программ без исходных кодов. Отладчики.
4. Приемы антиотладки.
5. Использование Windows Debuging API.

6. Модификация PE-файлов. Инструменты для модификации и просмотра PE-файлов.
7. Протекторы. Упаковщики.
8. Создание шеллкодов под Windows. Техника создания позиционно-независимого кода. Динамический поиск библиотечных функций.
9. Методы заражения исполняемых файлов.
10. Внедрение кода в чужой процесс.
11. Интерфейс системных вызовов в Linux. Шеллкоды под Linux.

3.2.5 Пример билета

1. Реализовать плагин для отладчика.
2. Приемы антиотладки.

4. ПОРЯДОК ПРОВЕДЕНИЯ И КРИТЕРИИ ОЦЕНИВАНИЯ ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ

4.1. Порядок проведения промежуточной аттестации

За своевременное и самостоятельно выполнение учебных работ в течение семестра студент получает рейтинговые баллы. Сумма за выполнение основных заданий в полном объёме - 100. Сверх этой суммы могут начисляться баллы за выполнение дополнительных заданий.

Основные баллы выставляются за выполнение объемных заданий (будем называть их зачётными), которые выполняются дома и сдаются в течение семестра. Сумма в 100 баллов делится между зачётными заданиями (не обязательно равномерно). При выдаче зачётного задания определено сколько баллов выставляется за реализацию определённой функциональности. Для зачётных заданий может быть определена функциональность повышенной сложности, за выполнение которой выставляются дополнительные баллы. Также дополнительные баллы могут быть выставлены по усмотрению преподавателя за особо примечательную реализацию.

По пройденному материалу выдаются небольшие задания для выполнения дома и/или во время семинарских занятий. За эти задания выставляются небольшие дополнительные баллы. Сдавать их можно либо в день выдачи либо на следующем занятии.

Пропуск по неуважительной причине одной пары влечет вычет 1 балла из итоговой суммы за семестр.

При нехватке баллов преподавателем может быть предоставлено

дополнительное задание или возможность доделать задание, в котором была оценена не вся функциональность.

Итоговая оценка за дисциплину выставляется по результатам выполнения заданий текущего контроля. При необходимости во время экзамена может быть предоставлена возможность получить дополнительные баллы (не более 20), выполнив дополнительные задания и ответив (в устной форме) на вопросы.

4.2. Критерии оценивания промежуточной аттестации по видам оценочных средств.

4.2.1 Критерии оценивания зачётных заданий

Выполнение заданий предполагает некоторую программную реализацию, к которой предъявляются обычные требования по качеству кода. Код должен быть удобочитаемым, хорошо структурированным, расширяемым, удобным в сопровождении, написанным в едином стиле. Должна быть проведена функциональная декомпозиция (на подпрограммы, модули, пакеты и т.д.), реализованы необходимые программные абстракции. Реализации алгоритмов должны быть логичными и понятными. Иначе возможна сбавка до 5 баллов. За программные ошибки, приводящие к неработоспособности кода для некоторых возможных случаев, возможна сбавка до 10 баллов (в зависимости от критичности ошибки). За программные ошибки, приводящие к аварийному некорректному завершению программы, возможна сбавка до 10 баллов (в зависимости от критичности ошибки).

При сдаче зачётного задания производится опрос по техническим деталям реализации и по теории, используемой при выполнении заданий. Неудовлетворительный ответ будет означать несамостоятельность выполнения задания, что влечёт выставление 0 баллов за соответствующую функциональность. Если для одного задания это повторяется более 2 раз, то за всё задание выставляется 0 баллов без возможности повторной сдачи.

4.2.2 Критерии оценивания домашних и аудиторных заданий

Домашние и аудиторные задания – это небольшие задания, за которые обычно выставляется 1-2 балла. Они оцениваются атомарно: либо задание выполнено (выставляется указанное при выдаче задания количество баллов), либо не выполнено (0 баллов).

4.2.3 Критерии оценивания заданий на экзамене

Дополнительные задания, выдаваемые на зачёте и экзамене являются относительно объемными, за них выставляется до 10-15 баллов. Поэтому к ним применимы описанные выше критерии оценивания зачётных заданий с соответствующей корректировкой баллов: сбавка за некорректную работу до 5 баллов, за аварийное завершение – до 5 баллов.

4.2.4 Критерии оценивания ответа в устной форме

Ответ оценивается по трём параметрам:

1) построение ответа (структура ответа, грамотность речи, последовательность и т.д.);

2) фактическая полнота ответа;

3) собственный анализ излагаемого материала, его оценка в контексте взаимодействия с другими областями, умение применять на практике.

Параметры оценивания	**Критерии оценивания**	**Баллы**
1.Построение ответа	Студент самостоятельно правильно выстраивает структуру ответа, изложение последовательное, речь грамотная без оговорок.	2
	Изложение студента непоследовательное и обрывочное, взаимосвязи частей ответа не всегда прослеживаются. Раскрытие сути ответа невозможно без уточняющих вопросов.	1
	Студент испытывает существенные трудности при самостоятельном построении ответа, способен только давать краткие ответы на конкретные вопросы.	0
2.Фактическая полнота ответа	Студент правильно ответил на теоретический вопрос билета. Показал отличные знания в рамках усвоенного учебного материала. Ответил на все дополнительные вопросы.	4
	Студент ответил на теоретический вопрос билета с небольшими неточностями. Показал хорошие знания в рамках усвоенного учебного материала. Ответил на большинство дополнительных вопросов.	3
	Студент ответил на теоретический вопрос билета с существенными неточностями. Показал удовлетворительные знания в рамках усвоенного учебного материала. При ответах на дополнительные вопросы было допущено много неточностей.	2
	При ответе на теоретический вопрос билета студент продемонстрировал недостаточный уровень знаний. При ответах на дополнительные вопросы было допущено множество неправильных ответов.	1
	Студент не ответил на вопрос.	0
3.Собственный анализ излагаемого	Студент ясно осознаёт место излагаемого материала в общей структуре профессионального знания, знает стандартные примеры использования и предлагает свои, даёт собственные компетентные оценки.	4
	Студент осознаёт взаимосвязи и знает стандартные примеры использования. Допускает неточности при самостоятельном анализе.	3

материала	Студент в общих чертах осознаёт взаимосвязи и знает стандартные примеры использования. При проведении самостоятельного анализа нуждается в уточняющих вопросах, при этом допускает существенные неточности.	2
	Студент знает основные стандартные примеры использования. Не осознаёт взаимосвязей с другими областями.	1
	Студент не осознаёт взаимосвязей и практическое приложение излагаемого материала.	0

4.3. Результаты промежуточной аттестации и уровни сформированности компетенций

Баллы, полученные за выполнение заданий текущего контроля и промежуточной аттестации, переводятся в оценки за экзамен и зачет следующим образом:

91 и более баллов – отлично;
76 - 90 баллов – хорошо;
61 - 75 баллов – удовлетворительно;
60 и менее баллов – неудовлетворительно.

Особенности проведении процедуры оценивания результатов обучения инвалидов и лиц с ограниченными возможностями здоровья обозначены в рабочей программе дисциплины (модуля).

Уровни сформированности компетенций определяются следующим образом:

Оценка	Отлично	Хорошо	Удовлетворительно	Неудовлетворительно
Баллы	более 90 баллов	76-90 баллов	61-75 баллов	0-60 баллов
Уровень освоения проверяемых компетенций	высокий	средний	базовый	недостаточный

Специальность 10.05.01 Компьютерная безопасность
Специализация № 1 «Анализ безопасности компьютерных систем»
Фонд оценочных средств по дисциплине «Защита программ и данных»
2026 год набора, очная форма обучения

Фонд оценочных средств дисциплины (модуля) одобрен и рекомендован:

Проректор по учебной работе	утверждено 27.02.26	А.А. Саламатов

Ученым советом математического факультета

Протокол заседания № 8 от 19.02.2026

Председатель Ученого совета математического факультета	согласовано	Е.А. Сбродова

Заседанием кафедры компьютерной безопасности и прикладной алгебры

Протокол заседания № 08 от 14.02.2026

Заведующий кафедрой	согласовано	А. Н. Ручай
Автор (составитель)		И. А. Маткин

Структура фондов оценочных средств соответствует приказу ректора ФГБОУ ВО «ЧелГУ» от 27 сентября 2022 №573-1