Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Информационные сети. Основы безопасности

ОТЧЁТ
к лабораторной работе №7
на тему

ЗАЩИТА ПО ОТ НЕСАНКЦИОНИРОВАННОГО ИСПОЛЬЗОВАНИЯ

Выполнил: студент гр.253501
Станишевский А.Д.

Проверил: ассистент кафедры информатики
Герчик А.В.

Минск 2025

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

Целью работы является исследование методов защиты исходного кода от обратной инженерии и декомпиляции через обфускацию, а также разработка демонстрационного программного решения, иллюстрирующего процесс преобразования читаемого кода в сложную для анализа структуру. В рамках работы реализованы две версии программы (исходная и обфусцированная), позволяющие провести сравнительный анализ эффективности различных методов обфускации, таких как замена идентификаторов, удаление комментариев и строк, а также генерация запутанного синтаксиса. Программа моделирует сценарий защиты кода, при котором использование обфускации затрудняет понимание логики программы и предотвращает несанкционированное извлечение конфиденциальной информации.

Особое внимание уделено реализации функционала, наглядно показывающего этапы обфускации – замены осмысленных имен переменных на случайные символы.

Таким образом, в ходе работы должны быть получены навыки анализа уязвимостей, связанных с распространением открытого исходного кода, освоены методы его защиты через применение современных алгоритмов обфускации, а также создано приложение, наглядно демонстрирующее процесс обфускации.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Обфускация кода (от англ. obfuscation – запутывание) – это процесс намеренного усложнения исходного кода программы с целью затруднить его анализ, понимание и модификацию. Обфускация применяется для защиты программного обеспечения от реверс-инжиниринга, декомпиляции и несанкционированного использования. Она особенно важна в условиях, когда программное обеспечение распространяется в виде бинарных файлов или скриптов, доступных для анализа.

Цели обфускации включают защиту интеллектуальной собственности, сокрытие логики работы программы, препятствие модификации и защиту от автоматизированных анализов. Основные методы обфускации включают переименование идентификаторов: например, переменные, функции и классы переименовываются в бессмысленные символы, что затрудняет понимание назначения элементов программы; добавление мусорного кода, который не влияет на выполнение, но усложняет анализ; шифрование строк с расшифровкой во время выполнения; изменение контроля потока выполнения путём добавления лишних переходов, циклов или условий; разбиение кода на множество небольших фрагментов, вызываемых в произвольном порядке; использование антиотладочных техник, таких как проверка на наличие отладчиков; и полиморфизм кода, при котором генерируются различные варианты кода для одной и той же функциональности. Минификация – удаление пробелов, комментариев и других человекочитаемых элементов – также является распространённым методом.

Преимущества обфускации заключаются в повышении безопасности, снижении риска финансовых потерь, сохранении конкурентных преимуществ и упрощённой реализации с помощью современных инструментов. Однако у обфускации есть недостатки, включая ограниченную защиту, увеличение времени выполнения, рост размера кода и сложность отладки.

Обфускация широко применяется в мобильных приложениях для защиты Android- и iOS-приложений от декомпиляции, в веб-разработке для обфускации JavaScript с целью предотвращения кражи клиентского кода, в игровой индустрии для защиты игровых движков и алгоритмов от взлома, а также в программном обеспечении с лицензионной защитой для предотвращения модификации программ с целью обхода лицензионных ограничений.

Таким образом, обфускация кода является важным инструментом в области информационной безопасности, помогающим защитить программное обеспечение от несанкционированного доступа и модификации. Однако она не должна рассматриваться как единственная мера защиты. Для достижения максимальной безопасности рекомендуется комбинировать обфускацию с другими методами, такими как шифрование, цифровые подписи и использование аппаратных средств защиты.

3 ХОД РАБОТЫ

На вход подается поле, в которое пользователь может ввести код на языке JavaScript, который будет использован для преобразования. На выходе пользователь получает код, с выполненной обфускацией с возможностью его загрузить или протестировать. На рисунке 3.1 представлена часть преобразуемого кода.

Рисунок 3.1 – Часть преобразуемого кода

На рисунке 3.2 представлен часть результата выполнения программы.

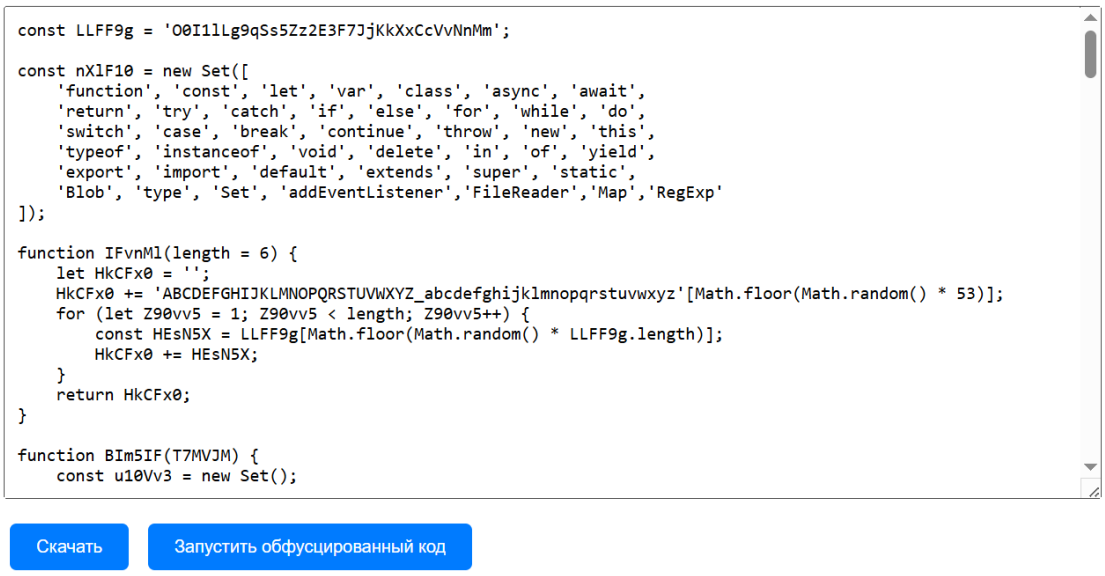

Рисунок 3.2 – Часть результата выполнения программы

Таким образом, защищенная программа успешно справляется с поставленными задачами.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была достигнута поставленная цель – исследование методов защиты исходного кода JavaScript от обратной инженерии и реализация демонстрационного приложения, иллюстрирующего как процесс обфускации, так и её роль в защите программного обеспечения. Разработанные версии программы (исходная и обфусцированная) наглядно демонстрируют ключевые принципы обеспечения безопасности при распространении кода: замену осмысленных идентификаторов на случайные символы.

Успешная деобфускация исходного кода, которая становится невозможной или крайне затруднительной после применения обфускации, подтверждает необходимость использования таких методов для защиты логики работы программ. В частности, использование понятных имен переменных, функций и комментариев позволяет злоумышленнику легко анализировать и модифицировать код, что может привести к краже интеллектуальной собственности, извлечению конфиденциальных данных или созданию несанкционированных копий программы.

Реализация обфусцированной версии программы показала, что применение современных алгоритмов обфускации, таких как замена идентификаторов, модификация строк, эффективно затрудняет анализ кода. Дополнительные меры, такие как защита регулярных выражений и строковых литералов от ошибочной обработки, позволяют значительно повысить надежность обфускации и минимизировать риск потери функциональности программы.

Таким образом, в результате проделанной работы были получены практические навыки анализа и предотвращения угроз, связанных с распространением открытого исходного кода, а также создано приложение, которое может быть использовано для демонстрации процесса обфускации.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг программного кода

```
const confusingChars = 'O0I1lLg9qSs5Zz2E3F7JjKkXxCcVvNnMm';
const KEYWORDS = new Set([
    'function', 'const', 'let', 'var', 'class', 'async', 'await',
    'return', 'try', 'catch', 'if', 'else', 'for', 'while', 'do',
    'switch', 'case', 'break', 'continue', 'throw', 'new', 'this',
    'typeof', 'instanceof', 'void', 'delete', 'in', 'of', 'yield',
    'export', 'import', 'default', 'extends', 'super', 'static',
    'Blob', 'type', 'Set', 'addEventListener','FileReader','Map','RegExp'
]);
function generateConfusingName(length = 6) {
    let name = '';
    name                                                                    +=
'ABCDEFGHIJKLMNOPQRSTUVWXYZ_abcdefghijklmnopqrstuvwxyz'[Math.floor(Math.rando
m() * 53)];
    for (let i = 1; i < length; i++) {
        const     char     =     confusingChars[Math.floor(Math.random()     *
confusingChars.length)];
        name += char;}return name}
function findDeclaredIdentifiers(code) {
    const declaredIdentifiers = new Set();

    const reservedParts = [];
    let cleanCode = code.replace(/(['"`])(?:\\\1|.)*?\1/g, function (match) {
        reservedParts.push(match);
        return `__RESERVED_PART_` + (reservedParts.length - 1)  + `__`;
    }).replace(/\/(?:\\\/|.)*?\/[gimuy]*/g, function (match) {
        reservedParts.push(match);
        return `__RESERVED_PART_` + (reservedParts.length - 1)  + `__`;
    });
    const dotNotationPattern = /(\w+)\.(\w+)(?:\s*\()?/g;
    let match;
    while ((match = dotNotationPattern.exec(cleanCode))) {
        const propertyOrMethod = match[2];
        if (!KEYWORDS.has(propertyOrMethod)) {
            cleanCode                  =                  cleanCode.replace(new
RegExp(`\\b`+propertyOrMethod+`\\s*\\(`, 'g'),
                `__SAFE_METHOD_`+propertyOrMethod+`__(`);}}
    const declarationPatterns = [
        /(?:const|let|var)\s+([a-zA-Z_$][\w$]*)/g,
        /function\s+([a-zA-Z_$][\w$]*)/g,
        /class\s+([a-zA-Z_$][\w$]*)/g,
        /([a-zA-Z_$][\w$]*)\s*\(/g,
        /([a-zA-Z_$][\w$]*)\s*:/g,
    ];
    declarationPatterns.forEach(pattern => {
        let declMatch;
        while ((declMatch = pattern.exec(cleanCode))) {
            const identifier = declMatch[1];
            if (!KEYWORDS.has(identifier) &&
                !identifier.includes('.') &&
                !identifier.startsWith('__SAFE_METHOD_') &&
                !identifier.startsWith('__RESERVED_PART_')) {
                declaredIdentifiers.add(identifier);
            }}});
    return declaredIdentifiers;
}
```