**Лабораторная работа № 1 – Изучение методов исследования исполняемых файлов с элементами самозащиты**

**Цель работы**

Изучение методов защиты вредоносного программного обеспечения и способов их обхода. Изучение методов анализа бинарного кода.

**Теоретические сведения**

При анализе исполняемых файлов широко применяется ряд технологий, позволяющий автоматизировать многие рутинные задачи. К таким технологиям относится написание скриптов и плагинов к средствам дизассемблерования и отладки, использование средств инструментации исполняемого кода, использование технологии символьного выполнения.

**Формулировка задания**

В ходе выполнения лабораторной работы необходимо выполнить следующие действия:

1. Получить у преподавателя исполняемые файлы, соответствующий индивидуальному номеру варианта задания. Полученная программа осуществляет считывание ключевой информации. В случае ее правильного ввода, осуществляется печать на экране строки «Success!» и вывод «секретного кода», в ином случае выводится «Access denied!» и программа завершается. (2 ключа – преобразование входа и просто вывод)
2. Изучить подходы, применяемые для защиты программного обеспечения.
3. Осуществить анализ полученной программы на наличие защитных механизмов с использованием как статических, так и динамических методов анализа.
4. Обойти защитные механизмы, присутствующие в программе путем ее модификации и перехвата (mhook/minhook), используемых ей функций.
5. Определить в бинарном коде программы место вывода секретного кода.
6. Получить «секретный» код путем исполнения модифицированной программы.
7. Разработать скрипты IDAPython, находящие механизмы самозащиты в программе и обходящие их. Разработанные скрипты должны выполнять следующие действия:
   1. Нахождение криптографических примитивов (xor).
   2. Нахождение закодированных строк (xor).
   3. Нахождение мест чтения PEB в коде.
   4. Нахождение мест замера времени (rdtsc, GetTickCount).
   5. Нахождение мест использования «редких» инструкций (int 2d, ss, и т.д.).
   6. Нахождение других признаков механизмов защиты бинарного кода.
8. Используя среду символьного выполнения (BARF, Triton, S2E, BitBlaze/FuzzBall, BAP, Fuzzgrind, MiAsm, angr) получить входное значение программы и секретный код.
9. Протестировать разработанные скрипты на исследуемом исполняемом файле.
10. Протестировать разработанные скрипты на исполняемом файле программы, соответствующей указанному варианту из архива bin2. Осуществить анализ полученного файла и решить crackme.
11. В отчете необходимо привести следующую информацию:

* описание алгоритма работы программы;
* пароль, запрашиваемый программой;
* «секретный код», полученный при анализе программы;
* место вывода «секретного кода» в бинарном коде программы;
* перечень механизмов защиты и фрагменты ассемблерного кода, отвечающие за их реализацию;
* описание принципа действия используемых механизмов защиты;
* описание процесса модификации программы и обхода защитных механизмов;
* описание решения crackme из архива bin2;
* исходный код реализованных скриптов IDAPython;
* исходный код скриптов для средств символьного исполнения.

**Полезные ссылки**

1. Using IDAPython to Make Your Life Easier: Part 1

<http://researchcenter.paloaltonetworks.com/2015/12/using-idapython-to-make-your-life-easier-part-1/>

1. <http://securityxploded.com/api-call-tracing-with-pefile-pydbg-and-idapython.php>
2. <https://firebitsbr.wordpress.com/2015/09/28/list-of-pluginsscripts-for-ida-pro-both-re-malware-vulnerabilities-and-exploits/>
3. <http://blog.talosintel.com/2016/12/project-first-share-knowledge-speed-up.html>
4. Анти-дизассемблерные приемы

<http://malwinator.com/anti-disassembly-used-in-malware-a-primer/>

<http://www.malwinator.com/anti-disassembly-techniques-used-by-malware-a-primer-part-2/>

1. ANTI-DISASSEMBLY

<http://staff.ustc.edu.cn/~bjhua/courses/security/2014/readings/anti-disas.pdf>

1. Примеры используемых антиотладочных приемов в ВПО

<http://research.dissect.pe/docs/blackhat2012-paper.pdf>

<https://github.com/rrbranco/blackhat2012>

1. Introduction Into Windows Anti-Debugging

<http://www.codeproject.com/Articles/29469/Introduction-Into-Windows-Anti-Debugging>

1. An Anti-Reverse Engineering Guide

<http://www.codeproject.com/Articles/30815/An-Anti-Reverse-Engineering-Guide>

1. Anti-Debug Protection Techniques: Implementation and Neutralization

<http://www.codeproject.com/Articles/1090943/Anti-Debug-Protection-Techniques-Implementation-an>

1. The “Ultimate”Anti – Debugging Reference

<http://pferrie.host22.com/papers/antidebug.pdf>

1. OpenRCE Anti Reverse Engineering Techniques Database

<http://www.openrce.org/reference_library/anti_reversing>

1. Anti Reverse Engineering Protection Techniques to Use Before Releasing Software

<https://www.apriorit.com/dev-blog/367-anti-reverse-engineering-protection-techniques-to-use-before-releasing-software>

1. The art of unpacking

<https://www.blackhat.com/presentations/bh-usa-07/Yason/Whitepaper/bh-usa-07-yason-WP.pdf>

1. Anti-Memory Dumping Techniques

<http://resources.infosecinstitute.com/anti-memory-dumping-techniques/>

1. Пишем упаковщик PE-файлов по шагам

<https://kaimi.ru/2012/09/pe-packer-step-by-step-1/>

1. Об упаковщиках в последний раз: Часть первая - теоретическая

<http://uinc.ru/articles/41/>

1. Об упаковщиках в последний раз. Часть вторая

<http://cracklab.narod.ru/doc/wasm2.htm>

1. Разработка своего PE-упаковщика

<https://habrahabr.ru/company/xakep/blog/139138/>

1. Empirical identification, comparison and circumvention of current Antivirus detection techniques

<https://www.blackhat.com/docs/us-14/materials/us-14-Mesbahi-One-Packer-To-Rule-Them-All-WP.pdf>

1. Strings Obfuscation System

<http://www.codeproject.com/Articles/502283/Strings-Obfuscation-System>

1. Binary Obfuscation

<https://www.defcon.org/images/defcon-17/dc-17-presentations/defcon-17-sean_taylor-binary_obfuscation.pdf>

1. Binary code obfuscation through C++ template metaprogramming

<https://eden.dei.uc.pt/~sneves/pubs/2012-snfa2.pdf>