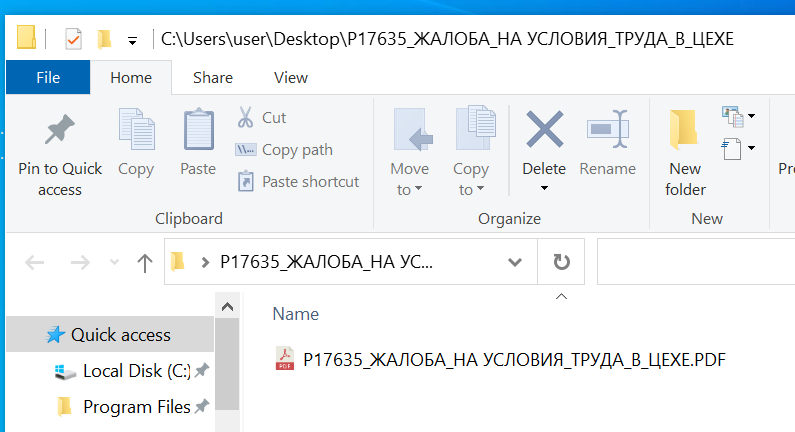


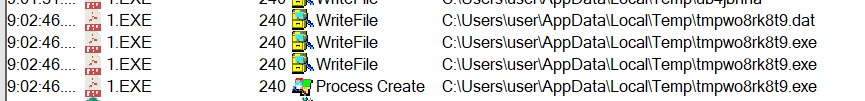
Полученное письмо содержит архив с паролем.



Внутри которого лежит pdf файл, но не совсем. На самом деле он имеет перевёрнутое расширение используя арабский символ. Истинное его расширение .exe. Проведём его анализ.

Файл представляет собой PyInstaller. Извлечём его внутренности при помощи <https://github.com/pyinstxtractor/pyinstxtractor-ng/releases/tag/2024.08.25>. Можем увидеть обфусцированный файл dropper.pyc.

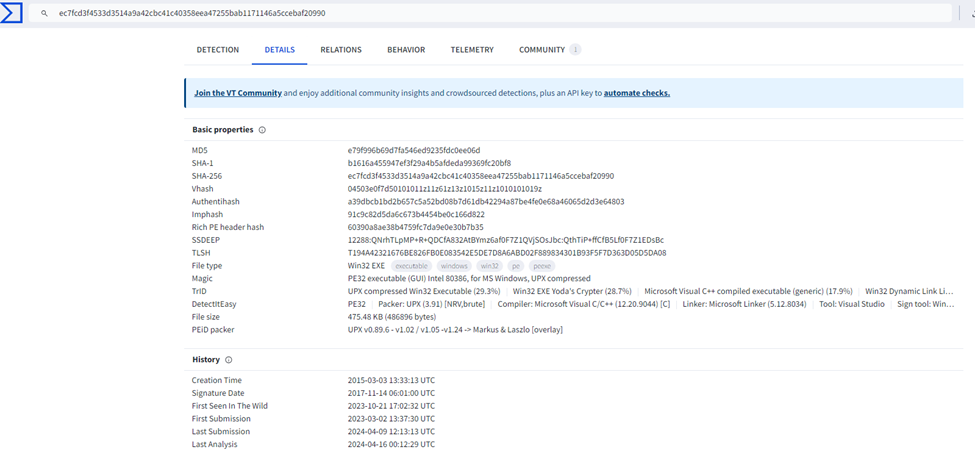
Попробуем запустить родительский файл и отследить его исполнение в Process Monitor.



Данный файл кладёт на диск 2 файла (.dat и .exe), после чего запускает исполняемый.

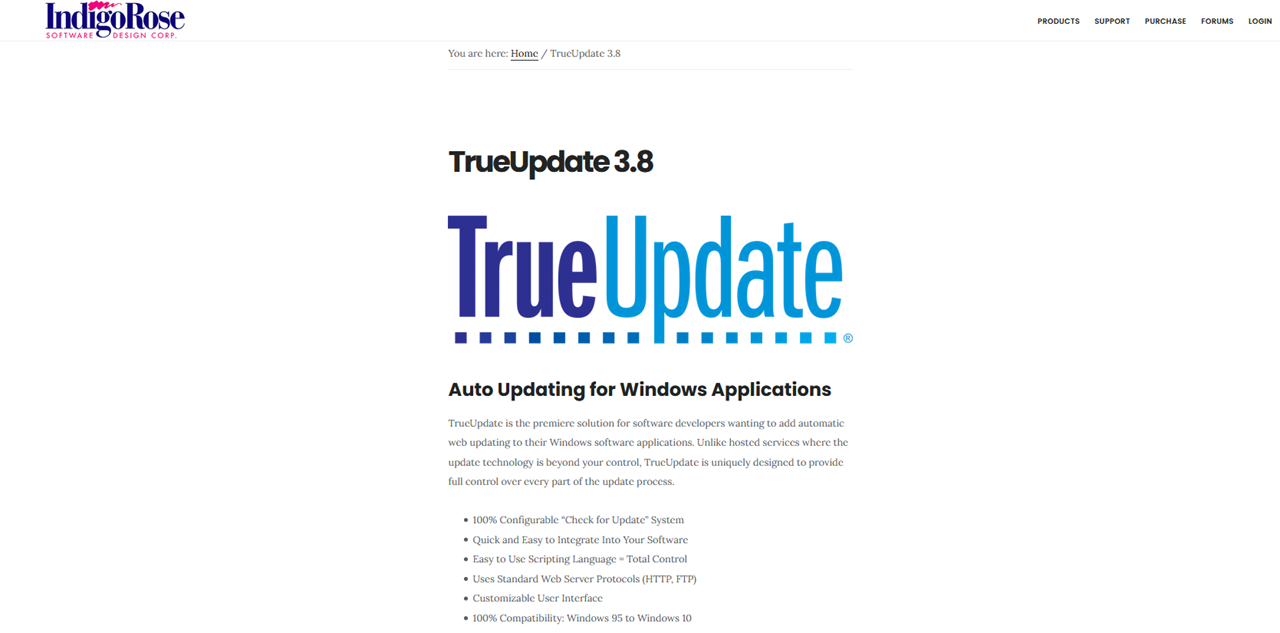
Отловив эти файлы попробуем их проанализировать.

Файла с одним названием, но с разными расширениями.

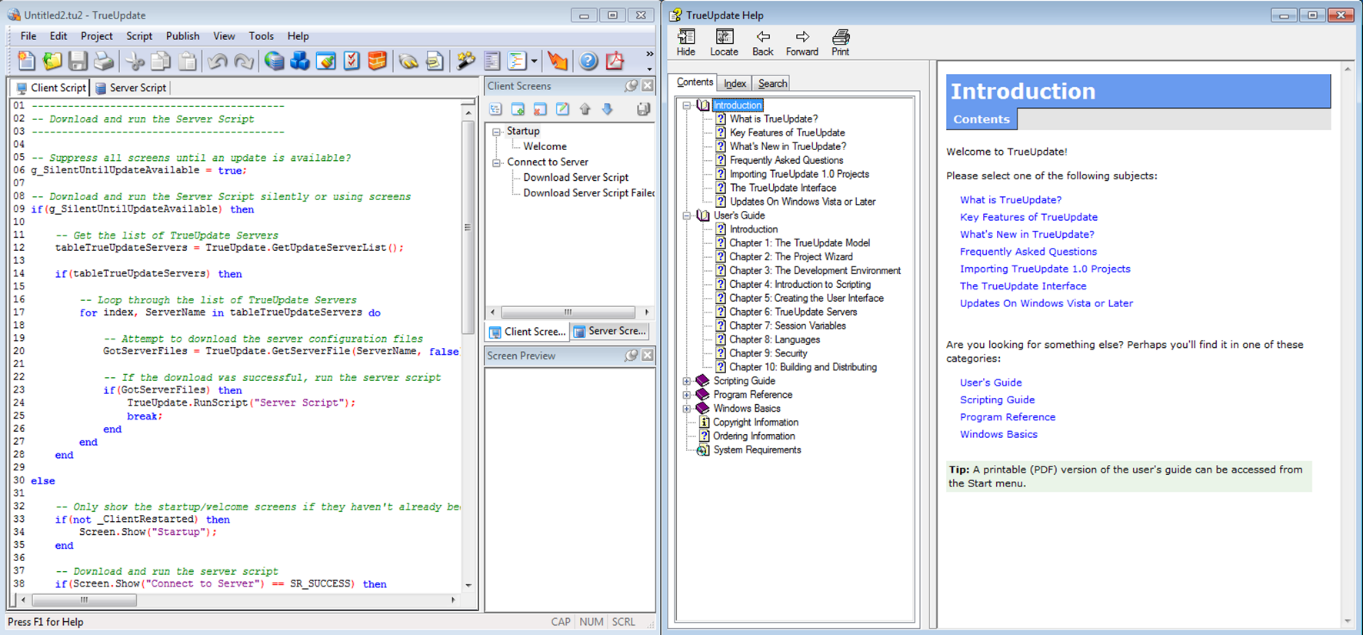
[](https://github.com/DevelopersOOIB/RDG_CTF_TB_2024/blob/main/Reverse/Suspicion%20mail/image-4.png)

Exe файл известен уже давно и помечается как чистый.

Заявляется, что это **TrueUpdate Client**, который разработан компанией **Indigo Rose**, и даже веб-сайт этой компании указан. Видим решение для разработчиков, желающих использовать автоматическое обновление ПО».

[](https://github.com/DevelopersOOIB/RDG_CTF_TB_2024/blob/main/Reverse/Suspicion%20mail/image-6.png)

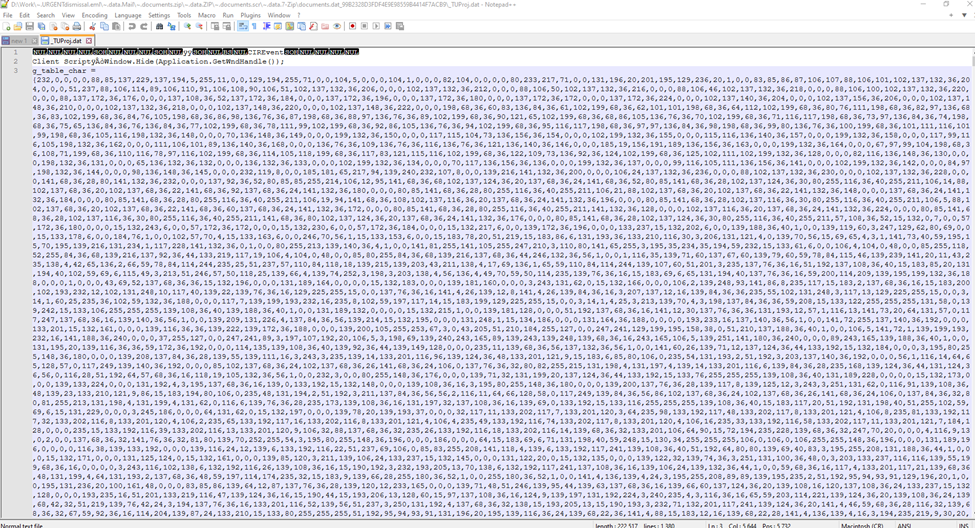
Для понимания работы можно изучить пробную версию TrueUpdate.

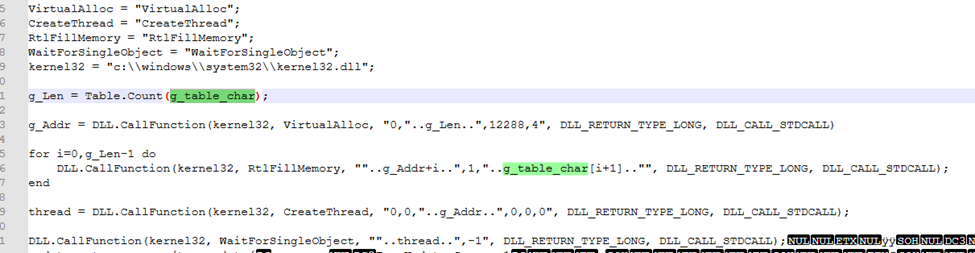
[](https://github.com/DevelopersOOIB/RDG_CTF_TB_2024/blob/main/Reverse/Suspicion%20mail/image-7.png)

Из руководства, это IDE, при помощи которой разработчик может развернуть инфраструктуру для удобного распространения и обновления своего ПО. Для каждого проекта в данной IDE при помощи **LUA-подобного** скриптового языка пишется пара скриптов – один для выполнения на клиенте, на котором происходит обновление, а второй – для работы на сервере обновлений. Клиентская часть затем собирается в дистрибутив и распространяется среди пользователей ПО, а серверная размещается на контролируемых разработчиком серверах, с которых в дальнейшем и происходит процесс скачивания обновлений клиентской частью. Действия клиентской части полностью определяются содержимым скрипта, сам скрипт находится в упакованном виде внутри прилагающегося **dat-файла** (который является защищенным паролем **ZIP-архивом**), а **.exe** является ничем иным, как своего рода интерпретатором, извлекающим клиентский скрипт из **dat-файла** и выполняющим его. Отсюда следует и второй вывод – пароль от архива содержится внутри**.exe**, и если каким-то образом узнать его, то можно попробовать подменить скрипт в архиве. Давайте вернёмся к нашему файлу.

Открываем .exe в **IDA**. Сам файл упакован, но так как в качестве упаковщика использован **UPX**, распаковать его не составляет никакого труда. Далее находим строку **«\_TUProj.dat»,** которая соответствует названию файла со скриптом внутри зашифрованного архива, по перекрестным ссылкам смотрим, где она используется, и находим следующий фрагмент кода, выполняющий работу по извлечению содержимого архива в память процесса: [](https://github.com/DevelopersOOIB/RDG_CTF_TB_2024/blob/main/Reverse/Suspicion%20mail/image-9.png)

Вся предлагаемая TrueUpdate «защита» клиентского кода сводится к стандартному шифрованию **ZIP-архива**, пароль от которого хранится в исполняемом файле в открытом виде. после извлечения из архива файла **\_TUProj.dat**. После распаковки файла, видим в нём скрипт.

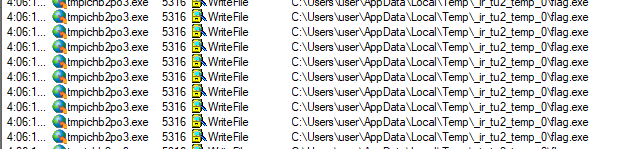
[](https://github.com/DevelopersOOIB/RDG_CTF_TB_2024/blob/main/Reverse/Suspicion%20mail/image-10.png)

[](https://github.com/DevelopersOOIB/RDG_CTF_TB_2024/blob/main/Reverse/Suspicion%20mail/image-11.png)

Как видим, кроме клиентского скрипта этот файл содержит также ряд иных ресурсов, использующихся программой. Сразу же обращает на себя внимание переменная **g\_table\_char**, которой присваивается набор чисел, сильно напоминающий шеллкод. Затем вызовом **VirtualAlloc** происходит выделение области памяти, достаточной для размещения в ней шеллкода, и побайтовое его копирование вызовами **RtlFillMemory**. После того, как шеллкод успешно размещён в памяти, вызовом **CreateThread** создаётся новый поток, начинающий выполнение шеллкода, а скрипт ожидает его завершения. После того, как шеллкод отработал, скрипт завершает выполнение.

Результатом анализа шелкода будет флаг.

Казалось бы, причём тут flag.exe, исполнение которого тоже видно в process monitor и который можно найти в .dat архиве?



А ни причём, этот файл никак не относится к таске и флагу. Основная часть кода находится в \_TUProj.dat.