Protecting customers from a private-sector offensive actor using 0-day exploits and DevilsTongue malware

microsoft.com/en-us/security/blog/2021/07/15/protecting-customers-from-a-private-sector-offensive-actor-using-0-day-exploits-and-devilstongue-malware/

July 15, 2021

By

The Microsoft Threat Intelligence Center (MSTIC) alongside the Microsoft Security Response Center (MSRC) has uncovered a private-sector offensive actor, or PSOA, that we are calling SOURGUM in possession of now-patched, Windows 0-day exploits (CVE-2021-31979 and CVE-2021-33771).

Private-sector offensive actors are private companies that manufacture and sell cyberweapons in hacking-as-a-service packages, often to government agencies around the world, to hack into their targets' computers, phones, network infrastructure, and other devices. With these hacking packages, usually the government agencies choose the targets and run the actual operations themselves. The tools, tactics, and procedures used by these companies only adds to the complexity, scale, and sophistication of attacks. We take these threats seriously and have moved swiftly alongside our partners to build in the latest protections for our customers.

MSTIC believes SOURGUM is an Israel-based private-sector offensive actor. We would like to thank the Citizen Lab, at the University of Toronto's Munk School, for sharing the sample of malware that initiated this work and their collaboration during the investigation. In their blog, Citizen Lab asserts with high confidence that SOURGUM is an Israeli company commonly known as Candiru. Third-party reports indicate Candiru produces "hacking tools [that] are used to break into computers and servers"*.*

As we shared in the Microsoft on the Issues blog, Microsoft and Citizen Lab have worked together to disable the malware being used by SOURGUM that targeted more than 100 victims around the world including politicians, human rights activists, journalists, academics, embassy workers, and political dissidents. To limit these attacks, Microsoft has created and built protections into our products against this unique malware, which we are calling *DevilsTongue*. We have shared these protections with the security community so that we can collectively address and mitigate this threat. We have also issued a software update that will protect Windows customers from the associated exploits that the actor used to help deliver its highly sophisticated malware.

SOURGUM victimology

Media reports (1, 2, 3) indicate that PSOAs often sell Windows exploits and malware in hacking-as-a-service packages to government agencies. Agencies in Uzbekistan, United Arab Emirates, and Saudi Arabia are among the list of Candiru's alleged previous customers. These agencies, then, likely choose whom to target and run the cyberoperations themselves.

Microsoft has identified over 100 victims of SOURGUM's malware, and these victims are as geographically diverse as would be expected when varied government agencies are believed to be selecting the targets. Approximately half of the victims were found in Palestinian Authority, with most of the remaining victims located in Israel, Iran, Lebanon, Yemen, Spain (Catalonia), United Kingdom, Turkey, Armenia, and Singapore. To be clear, the identification of victims of the malware in a country doesn't necessarily mean that an agency in that country is a SOURGUM customer, as international targeting is common.

Any Microsoft 365 Defender and Microsoft Defender for Endpoint alerts containing detection names for the DevilsTongue malware name are signs of compromise by SOURGUM's malware. We have included a comprehensive list of detection names below for customers to perform additional hunting in their environments.

Exploits

SOURGUM appears to use a chain of browser and Windows exploits, including 0-days, to install malware on victim boxes. Browser exploits appear to be served via single-use URLs sent to targets on messaging applications such as WhatsApp.

During the investigation, Microsoft discovered two Windows 0-day exploits for vulnerabilities tracked as CVE-2021-31979 and CVE-2021-33771, both of which have been fixed in the July 2021 security updates. These vulnerabilities allow privilege escalation, giving an attacker the ability to escape browser sandboxes and gain kernel code execution. If customers have taken the July 2021 security update, they are protected from these exploits.

CVE-2021-31979 fixes an integer overflow within Windows NT-based operating system (NTOS). This overflow results in an incorrect buffer size being calculated, which is then used to allocate a buffer in the kernel pool. A buffer overflow subsequently occurs while copying memory to the smaller-than-expected destination buffer. This vulnerability can be leveraged to corrupt an object in an adjacent memory allocation. Using APIs from user mode, the kernel pool memory layout can be groomed with controlled allocations, resulting in an object being placed in the adjacent memory location. Once corrupted by the buffer overflow, this object can be turned into a user mode to kernel mode read/write primitive. With these primitives in place, an attacker can then elevate their privileges.

CVE-2021-33771 addresses a race condition within NTOS resulting in the use-after-free of a kernel object. By using multiple racing threads, the kernel object can be freed, and the freed memory reclaimed by a controllable object. Like the previous vulnerability, the kernel pool memory can be sprayed with allocations using user mode APIs with the hopes of landing an object allocation within the recently freed memory. If successful, the controllable object can be used to form a user mode to kernel mode read/write primitive and elevate privileges.

DevilsTongue malware overview

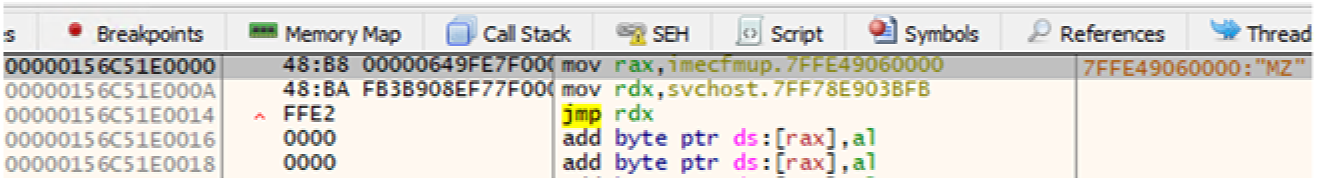
DevilsTongue is a complex modular multi-threaded piece of malware written in C and C++ with several novel capabilities. Analysis is still on-going for some components and capabilities, but we're sharing our present understanding of the malware so defenders can use this intelligence to protect networks and so other researchers can build on our analysis.

For files on disk, PDB paths and PE timestamps are scrubbed, strings and configs are encrypted, and each file has a unique hash. The main functionality resides in DLLs that are encrypted on disk and only decrypted in memory, making detection more difficult. Configuration and tasking data is separate from the malware, which makes analysis harder. DevilsTongue has both user mode and kernel mode capabilities. There are several novel detection evasion mechanisms built in. All these features are evidence that SOURGUM developers are very professional, have extensive experience writing Windows malware, and have a good understanding of operational security.

When the malware is installed, a first-stage 'hijack' malware DLL is dropped in a subfolder of *C:\Windows\system32\IME\*; the folders and names of the hijack DLLs blend with legitimate names in the *\IME* directories. Encrypted second-stage malware and config files are dropped into subfolders of *C:\Windows\system32\config\* with a *.dat* file extension. A third-party legitimate, signed driver *physmem.sys* is dropped to the *system32\drivers* folder. A file called *WimBootConfigurations.ini* is also dropped; this file has the command for following the COM hijack. Finally, the malware adds the hijack DLL to a COM class registry key, overwriting the legitimate COM DLL path that was there, achieving persistence via COM hijacking.

From the COM hijacking, the DevilsTongue first-stage hijack DLL gets loaded into a *svchost.exe* process to run with SYSTEM permissions. The COM hijacking technique means that the original DLL that was in the COM registry key isn't loaded. This can break system functionality and trigger an investigation that could lead to the discovery of the malware, but DevilsTongue uses an interesting technique to avoid this. In its *DllMain* function it calls *LoadLibrary* on the original COM DLL so it is correctly loaded into the process. DevilsTongue then searches the call stack to find the return address of *LoadLibraryExW* (i.e., the function currently loading the DevilsTongue DLL), which would usually return the base address of the DevilsTongue DLL.

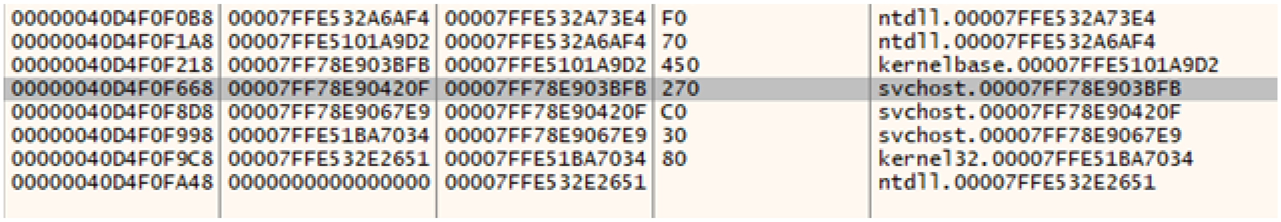
Once the *LoadLibraryExW* return address has been found, DevilsTongue allocates a small buffer with shellcode that puts the COM DLL's base address (*imecfmup.7FFE49060000* in Figure 1) into the *rax* register and then jumps to the original return address of *LoadLibraryExW* (*svchost.7FF78E903BFB* in Figures 1 and 2). In Figure 1 the COM DLL is named *imecfmup* rather than a legitimate COM DLL name because some DevilsTongue samples copied the COM DLL to another location and renamed it.



```
00000156C51E0000     48:B8 00000649FE7F00( mov rax,imecfmup.7FFE49060000     7FFE49060000:"MZ"
00000156C51E000A     48:BA FB3B908EF77F00( mov rdx,svchost.7FF78E903BFB
00000156C51E0014   ^ FFE2                  jmp rdx
00000156C51E0016     0000                  add byte ptr ds:[rax],al
00000156C51E0018     0000                  add byte ptr ds:[rax],al
```

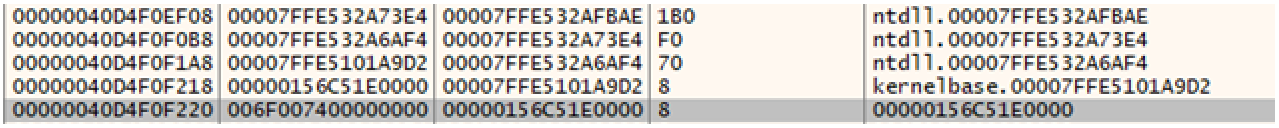
Figure 1. DevilsTongue *return address modification shellcode*

DevilsTongue then swaps the original *LoadLibraryExW* return address on the stack with the address of the shellcode so that when *LoadLibraryExW* returns it does so into the shellcode (Figures 2 and 3). The shellcode replaces the DevilsTongue base address in *rax* with the COM DLL's base address, making it look like *LoadLibraryExW* has returned the COM DLL's address. The *svchost.exe* host process now uses the returned COM DLL base address as it usually would.



```
00000040D4F0F0B8 00007FFE532A6AF4 00007FFE532A73E4 F0    ntdll.00007FFE532A73E4
00000040D4F0F1A8 00007FFE5101A9D2 00007FFE532A6AF4 70    ntdll.00007FFE532A6AF4
00000040D4F0F218 00007FF78E903BFB 00007FFE5101A9D2 450   kernelbase.00007FFE5101A9D2
00000040D4F0F668 00007FF78E90420F 00007FF78E903BFB 270   svchost.00007FF78E903BFB
00000040D4F0F8D8 00007FF78E9067E9 00007FF78E90420F C0    svchost.00007FF78E90420F
00000040D4F0F998 00007FFE51BA7034 00007FF78E9067E9 30    svchost.00007FF78E9067E9
00000040D4F0F9C8 00007FFE532E2651 00007FFE51BA7034 80    kernel32.00007FFE51BA7034
00000040D4F0FA48 0000000000000000 00007FFE532E2651       ntdll.00007FFE532E2651
```

Figure 2. Call stack before stack swap, LoadLibraryExW in kernelbase returning to svchost.exe (0x7FF78E903BFB)



```
00000040D4F0EF08 00007FFE532A73E4 00007FFE532AFBAE 1B0   ntdll.00007FFE532AFBAE
00000040D4F0F0B8 00007FFE532A6AF4 00007FFE532A73E4 F0    ntdll.00007FFE532A73E4
00000040D4F0F1A8 00007FFE5101A9D2 00007FFE532A6AF4 70    ntdll.00007FFE532A6AF4
00000040D4F0F218 00000156C51E0000 00007FFE5101A9D2 8     kernelbase.00007FFE5101A9D2
00000040D4F0F220 006F007400000000 00000156C51E0000 8     00000156C51E0000
```

Figure 3. Call stack after stack swap, LoadLibraryExW in kernelbase returning to the shellcode address (0x156C51E0000 from Figure 1)

This technique ensures that the DevilsTongue DLL is loaded by the *svchost.exe* process, giving the malware persistence, but that the legitimate COM DLL is also loaded correctly so there's no noticeable change in functionality on the victim's systems.

After this, the hijack DLL then decrypts and loads a second-stage malware DLL from one of the encrypted *.dat* files. The second-stage malware decrypts another *.dat* file that contains multiple helper DLLs that it relies on for functionality.

DevilsTongue has standard malware capabilities, including file collection, registry querying, running WMI commands, and querying SQLite databases. It's capable of stealing victim credentials from both LSASS and from browsers, such as Chrome and Firefox. It also has dedicated functionality to decrypt and exfiltrate conversations from the Signal messaging app.

It can retrieve cookies from a variety of web browsers. These stolen cookies can later be used by the attacker to sign in as the victim to websites to enable further information gathering. Cookies can be collected from these paths (** is a wildcard to match any folders*):

- *%LOCALAPPDATA%\Chromium\User Data\\*\Cookies*
- *%LOCALAPPDATA%\Google\Chrome\User Data\\*\Cookies*
- *%LOCALAPPDATA%\Microsoft\Windows\INetCookies*
- *%LOCALAPPDATA%\Packages\\*\AC\\*\MicrosoftEdge\Cookies*
- *%LOCALAPPDATA%\UCBrowser\User Data_i18n\\*\Cookies.9*
- *%LOCALAPPDATA%\Yandex\YandexBrowser\User Data\\*\Cookies*
- *%APPDATA%\Apple Computer\Safari\Cookies\Cookies.binarycookies*
- *%APPDATA%\Microsoft\Windows\Cookies*
- *%APPDATA%\Mozilla\Firefox\Profiles\\*\cookies.sqlite*

- *%APPDATA%\Opera Software\Opera Stable\Cookies*

Interestingly, DevilsTongue seems able to use cookies directly from the victim's computer on websites such as Facebook, Twitter, Gmail, Yahoo, Mail.ru, Odnoklassniki, and Vkontakte to collect information, read the victim's messages, and retrieve photos. DevilsTongue can also send messages as the victim on some of these websites, appearing to any recipient that the victim had sent these messages. The capability to send messages could be weaponized to send malicious links to more victims.

Alongside DevilsTongue a third-party signed driver is dropped to *C:\Windows\system32\drivers\physmem.sys*. The driver's description is "Physical Memory Access Driver," and it appears to offer a "by-design" kernel read/write capability. This appears to be abused by DevilsTongue to proxy certain API calls via the kernel to hinder detection, including the capability to have some of the calls appear from other processes. Functions capable of being proxied include *CreateProcessW, VirtualAllocEx, VirtualProtectEx, WriteProcessMemory, ReadProcessMemory, CreateFileW* and *RegSetKeyValueW*.

Prevention and detection

To prevent compromise from browser exploits, it's recommended to use an isolated environment, such as a virtual machine, when opening links from untrusted parties. Using a modern version of Windows 10 with virtualization-based protections, such as Credential Guard, prevents DevilsTongue's LSASS credential-stealing capabilities. Enabling the attack surface reduction rule "Block abuse of exploited vulnerable signed drivers" in Microsoft Defender for Endpoint blocks the driver that DevilsTongue uses. Network protection blocks known SOURGUM domains.

Detection opportunities

This section is intended to serve as a non-exhaustive guide to help customers and peers in the cybersecurity industry to detect the DevilsTongue malware. We're providing this guidance with the expectation that SOURGUM will likely change the characteristics we identify for detection in their next iteration of the malware. Given the actor's level of sophistication, however, we believe that outcome would likely occur irrespective of our public guidance.

File locations

The hijack DLLs are in subfolders of *\system32\ime\* with names starting with '*im'.* However, they are blended with legitimate DLLs in those folders. To distinguish between the malicious and benign, the legitimate DLLs are signed (on Windows 10) whereas the DevilsTongue files aren't. Example paths:

- *C:\Windows\System32\IME\IMEJP\imjpueact.dll*

- *C:\Windows\system32\ime\IMETC\IMTCPROT.DLL*
- *C:\Windows\system32\ime\SHARED\imecpmeid.dll*

The DevilsTongue configuration files, which are AES-encrypted, are in subfolders of *C:\Windows\system32\config\* and have a *.dat* extension. The exact paths are victim-specific, although some folder names are common across victims. As the files are AES-encrypted, any files whose size mod 16 is 0 can be considered as a possible malware config file. The config files are always in new folders, not the legitimate existing folders (e.g., on Windows 10, never in \Journal, \systemprofile, \TxR etc.). Example paths:

- *C:\Windows\system32\config\spp\ServiceState\Recovery\pac.dat*
- *C:\Windows\system32\config\cy-GB\Setup\SKB\InputMethod\TupTask.dat*
- *C:\Windows\system32\config\config\startwus.dat*

Commonly reused folder names in the config file paths:

- *spp*
- *SKB*
- *curv*
- *networklist*
- *Licenses*
- *InputMethod*
- *Recovery*

The .ini reg file has the unique name *WimBootConfigurations.ini* and is in a subfolder of *system32\ime\*. Example paths:

- *C:\Windows\system32\ime\SHARED\WimBootConfigurations.ini*
- *C:\Windows\system32\ime\IMEJP\WimBootConfigurations.ini*
- *C:\Windows\system32\ime\IMETC\WimBootConfigurations.ini*

The Physmem driver is dropped into system32:

> *C:\Windows\system32\drivers\physmem.sys*

Behaviors

The two COM keys that have been observed being hijacked for persistence are listed below with their default clean values. If their default value DLL is in the *\system32\ime\* folder, the DLL is likely DevilsTongue.

- *HKLM\SOFTWARE\Classes\CLSID\{CF4CC405-E2C5-4DDD-B3CE-5E7582D8C9FA}\InprocServer32 = %systemroot%\system32\wbem\wmiutils.dll (clean default value*)

- *HKLM\SOFTWARE\Classes\CLSID\{7C857801-7381-11CF-884D-00AA004B2E24}\InProcServer32 = %systemroot%\system32\wbem\wbemsvc.dll (clean default value*)

File content and characteristics

This Yara rule can be used to find the DevilsTongue hijack DLL:

```
import "pe"
rule DevilsTongue_HijackDll
{
meta:
description = "Detects SOURGUM's DevilsTongue hijack DLL"
author = "Microsoft Threat Intelligence Center (MSTIC)"
date = "2021-07-15"
strings:
$str1 = "windows.old\\windows" wide
$str2 = "NtQueryInformationThread"
$str3 = "dbgHelp.dll" wide
$str4 = "StackWalk64"
$str5 = "ConvertSidToStringSidW"
$str6 = "S-1-5-18" wide
$str7 = "SMNew.dll" // DLL original name
// Call check in stack manipulation
// B8 FF 15 00 00   mov     eax, 15FFh
// 66 39 41 FA      cmp     [rcx-6], ax
// 74 06            jz      short loc_1800042B9
// 80 79 FB E8      cmp     byte ptr [rcx-5], 0E8h ; 'è'
$code1 = {B8 FF 15 00 00 66 39 41 FA 74 06 80 79 FB E8}
// PRNG to generate number of times to sleep 1s before exiting
// 44 8B C0 mov r8d, eax
// B8 B5 81 4E 1B mov eax, 1B4E81B5h
// 41 F7 E8 imul r8d
// C1 FA 05 sar edx, 5
// 8B CA    mov ecx, edx
// C1 E9 1F shr ecx, 1Fh
// 03 D1    add edx, ecx
// 69 CA 2C 01 00 00 imul ecx, edx, 12Ch
// 44 2B C1 sub r8d, ecx
// 45 85 C0 test r8d, r8d
// 7E 19    jle  short loc_1800014D0
$code2 = {44 8B C0 B8 B5 81 4E 1B 41 F7 E8 C1 FA 05 8B CA C1 E9 1F 03 D1 69 CA
2C 01 00 00 44 2B C1 45 85 C0 7E 19}
condition:
filesize < 800KB and
uint16(0) == 0x5A4D and
(pe.characteristics & pe.DLL) and
(
4 of them or
($code1 and $code2) or
(pe.imphash() == "9a964e810949704ff7b4a393d9adda60")
)
}
```

Microsoft Defender Antivirus detections

Microsoft Defender Antivirus detects DevilsTongue malware with the following detections:

- *Trojan:Win32/DevilsTongue.A!dha*
- *Trojan:Win32/DevilsTongue.B!dha*
- *Trojan:Script/DevilsTongueIni.A!dha*
- *VirTool:Win32/DevilsTongueConfig.A!dha*
- *HackTool:Win32/DevilsTongueDriver.A!dha*

Microsoft Defender for Endpoint alerts

Alerts with the following titles in the security center can indicate DevilsTongue malware activity on your network:

- *COM Hijacking*
- *Possible theft of sensitive web browser information*
- *Stolen SSO cookies*

Azure Sentinel query

To locate possible SOURGUM activity using Azure Sentinel, customers can find a Sentinel query containing these indicators in this GitHub repository.

Indicators of compromise (IOCs)

No malware hashes are being shared because DevilsTongue files, except for the third part driver below, all have unique hashes, and therefore, are not a useful indicator of compromise.

Physmem driver

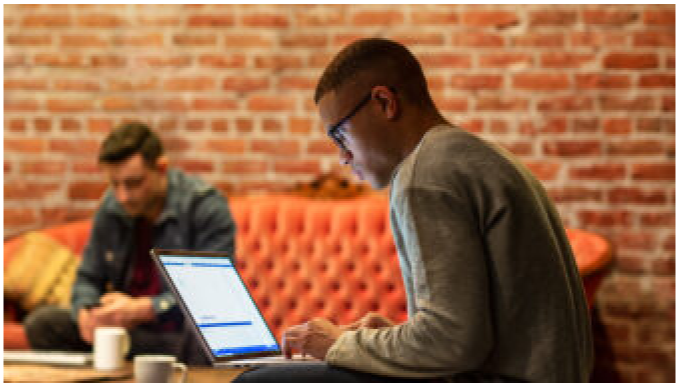
Note that this driver may be used legitimately, but if it’s seen on path *C:\Windows\system32\drivers\physmem.sys* then it is a high-confidence indicator of DevilsTongue activity. The hashes below are provided for the one driver observed in use.

- *MD5: a0e2223868b6133c5712ba5ed20c3e8a*
- *SHA-1: 17614fdee3b89272e99758983b99111cbb1b312c*
- *SHA-256: c299063e3eae8ddc15839767e83b9808fd43418dc5a1af7e4f44b97ba53fbd3d*

Domains

- noc-service-streamer[.]com
- fbcdnads[.]live
- hilocake[.]info
- backxercise[.]com
- winmslaf[.]xyz
- service-deamon[.]com
- online-affiliate-mon[.]com
- codeingasmylife[.]com
- kenoratravels[.]com
- weathercheck[.]digital
- colorpallatess[.]com
- library-update[.]com
- online-source-validate[.]com
- grayhornet[.]com
- johnshopkin[.]net
- eulenformacion[.]com
- pochtarossiy[.]info

Related Posts



Research

Threat intelligence

Microsoft Incident Response

Threat actors

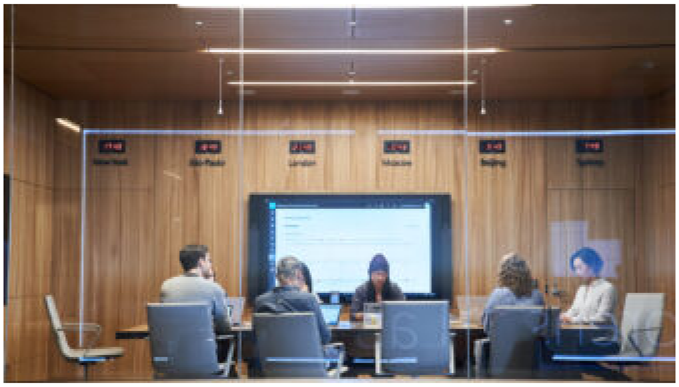
Oct 2517 min read

Octo Tempest crosses boundaries to facilitate extortion, encryption, and destruction

Microsoft has been tracking activity related to the financially motivated threat actor Octo Tempest, whose evolving campaigns represent a growing concern for many organizations across multiple industries.

Uncursing the ncurses: Memory corruption vulnerabilities found in library

A set of memory corruption vulnerabilities in the ncurses library could have allowed attackers to chain the vulnerabilities to elevate privileges and run code in the targeted program's context or perform other malicious actions.



The five-day job: A BlackByte ransomware intrusion case study

In a recent investigation by Microsoft Incident Response of a BlackByte 2.0 ransomware attack, we found that the threat actor progressed through the full attack chain, from initial access to impact, in less than five days, causing significant business disruption for the victim organization.

DEV-0196: QuaDream's "KingsPawn" malware used to target civil society in Europe, North America, the Middle East, and Southeast Asia

Microsoft analyzes a threat group tracked as DEV-0196, the actor's iOS malware "KingsPawn", and their link to an Israel-based private sector offensive actor (PSOA) known as QuaDream, which reportedly sells a suite of exploits, malware, and infrastructure called REIGN, that's designed to exfiltrate data from mobile devices.