Clipping Wings: Our Analysis of a Pegasus Spyware Sample

Published: 2024-03-25 · Archived: 2026-04-05 21:53:47 UTC

Black Hat Asia 2024 Preview from iVerify VP of Research, Matthias Frielingsdorf

According to

Kaspersky

, mobile threats now account for 40% of all detected threats. Similarly, Google's Threat Analysis Group (TAG) recently

reported

that 20 out of 25 zero-days discovered in 2023 were exploited by commercial spyware vendors (CSVs). Once thought to be contained to high-risk individuals such as journalists and political dissidents, the immediate threat of mobile-first zero-days has finally hit businesses resulting in intellectual property theft, credential breaches, corporate espionage, and ransomware attacks.

One of the most commonly detected spyware in 2023 was Pegasus, developed by NSO Group and the subject of Amnesty International's investigative journalism initiative: the

Pegasus Project

. Several months ago our team discovered evidence of Pegasus during an analysis of a customer's iPhone and we were able to extract the PassKit file from the device's backup.

This post is a preview of my briefing at the upcoming Black Hat Asia conference, where I'll show the results of our analysis of the malware sample!

Mobile-First Zero-Days: Acknowledging Past Research

Previous studies from the research community identified several iOS vulnerabilities exploited by attacks to install Pegasus on target devices.

One of them, which was subsequently fixed by Apple was CVE-2023-41064, which allowed an attacker to craft an image to gain code execution inside of ImageIO using the WebP image format. Existing resources provide an excellent description of this vulnerability, including:

Additionally, CitizenLab previously released details about an exploit chain called PWNYOURHOME that used a combination of HomeKit and iMessage to attack devices. They later revealed details about BLASTPASS, an exploit discovered in the wild targeting iMessage and PassKit (Wallet).

Our analysis of an actual BLASTPASS exploit sample adds to the industry's current body of knowledge catalyzed by these researchers and more.

Backups, Backups, Backups

For this analysis, we had access to the customer's iTunes backups, crash logs, and sysdiagnose files. One of the best things about Threat Hunter is that we can gather these artifacts remotely without needing physical access to the device.

This was one of the rare cases where we could spot something obvious right away.

We started our analysis by reviewing the available files and directory of crash logs. To our surprise, this was one of the rare cases where we should spot something obvious right away. We found 25 crashes of the ***homed*** process occurring within a 19-minute window. As we've seen before, recurring crashes of the same process often suggest the presence of a nasty bug or an attempted exploit. We often see both when analyzing suspicious iPhones since iOS is far from bug-free, which you'll notice if you routinely check your crash logs or Apple's Security Releases.

An important note about the homed crashes we saw – they weren't the kind of crashes we'd characterize as severe like Segmentation Fault or Null Pointer Deref. These are pretty normal crashes, but they occurred with unusual frequency.

We also noticed a couple of crashes of the ***MessagesBlastDoorService***. BlastDoor is the tightened sandbox that Apple introduced in iOS 14 to thwart iMessage Exploitation after NSO abused it. In theory, that should have resulted in every iMessage attachment being rendered and parsed safely inside of BlastDoor. However, we saw a large number of MessagesBlastDoor crashes happening within a short period, starting approximately 30 minutes after the last crash of homed.

Could all of this be just a coincidence – a combined series of ***homed*** and ***MessagesBlastDoorService*** crashes occurring so close together?

Not likely. To understand why, let's go back to details found in the crash logs.

Not only do we eventually find more severe crash types, we also see ***MessagesBlastDoorService*** trying to unarchive a very large or very repetitive NSKeyedArchiver. The CallStack of the crashing stack screams suspicious and other ***MessagesBlastDoorService*** crashes follow the same pattern.

MessagesBlastDoorService-2023-08-23-104248.ips Open with Console

```
Thread 1 name:   Dispatch queue: com.apple.root.default-qos.overcommit
Thread 1 Crashed:
0   CoreFoundation                          0x1c191e644 __CFStringEncodeByteStream + 76
1   Foundation                              0x1bbbdfeb0 -[NSString(NSStringOtherEncodings)
getBytes:maxLength:usedLength:encoding:options:range:remainingRange:] + 260
2   CoreFoundation                          0x1c1914c70 -[NSTaggedPointerString
getBytes:maxLength:usedLength:encoding:options:range:remainingRange:] + 100
3   Foundation                              0x1bbbdec9c -[NSString(NSStringOtherEncodings)
getCString:maxLength:encoding:] + 136
4   Foundation                              0x1bbbdeab8 NSClassFromString + 76
5   Foundation                              0x1bbbfe838 _decodeObjectBinary + 1648
6   Foundation                              0x1bbbd748c -[NSKeyedUnarchiver _decodeArrayOfObjectsForKey:] + 1592
7   Foundation                              0x1bbbd6a88 -[NSArray(NSArray) initWithCoder:] + 152
8   Foundation                              0x1bbbfeb4c _decodeObjectBinary + 2436
9   Foundation                              0x1bbbd748c -[NSKeyedUnarchiver _decodeArrayOfObjectsForKey:] + 1592
10  Foundation                              0x1bbbd6a88 -[NSArray(NSArray) initWithCoder:] + 152
11  Foundation                              0x1bbbfeb4c _decodeObjectBinary + 2436
12  Foundation                              0x1bbbd748c -[NSKeyedUnarchiver _decodeArrayOfObjectsForKey:] + 1592
13  Foundation                              0x1bbbd6a88 -[NSArray(NSArray) initWithCoder:] + 152
14  Foundation                              0x1bbbfeb4c _decodeObjectBinary + 2436
15  Foundation                              0x1bbbd748c -[NSKeyedUnarchiver _decodeArrayOfObjectsForKey:] + 1592
16  Foundation                              0x1bbbd6a88 -[NSArray(NSArray) initWithCoder:] + 152
17  Foundation                              0x1bbbfeb4c _decodeObjectBinary + 2436
18  Foundation                              0x1bbbd748c -[NSKeyedUnarchiver _decodeArrayOfObjectsForKey:] + 1592
19  Foundation                              0x1bbbd6a88 -[NSArray(NSArray) initWithCoder:] + 152
20  Foundation                              0x1bbbfeb4c _decodeObjectBinary + 2436
21  Foundation                              0x1bbbd748c -[NSKeyedUnarchiver _decodeArrayOfObjectsForKey:] + 1592
22  Foundation                              0x1bbbd6a88 -[NSArray(NSArray) initWithCoder:] + 152
23  Foundation                              0x1bbbfeb4c _decodeObjectBinary + 2436
24  Foundation                              0x1bbbd748c -[NSKeyedUnarchiver _decodeArrayOfObjectsForKey:] + 1592
25  Foundation                              0x1bbbd6a88 -[NSArray(NSArray) initWithCoder:] + 152
26  Foundation                              0x1bbbfeb4c _decodeObjectBinary + 2436
27  Foundation                              0x1bbbd748c -[NSKeyedUnarchiver _decodeArrayOfObjectsForKey:] + 1592
28  Foundation                              0x1bbbd6a88 -[NSArray(NSArray) initWithCoder:] + 152
29  Foundation                              0x1bbbfeb4c _decodeObjectBinary + 2436
30  Foundation                              0x1bbbd748c -[NSKeyedUnarchiver _decodeArrayOfObjectsForKey:] + 1592
31  Foundation                              0x1bbbd6a88 -[NSArray(NSArray) initWithCoder:] + 152
32  Foundation                              0x1bbbfeb4c _decodeObjectBinary + 2436
33  Foundation                              0x1bbbd748c -[NSKeyedUnarchiver _decodeArrayOfObjectsForKey:] + 1592
34  Foundation                              0x1bbbd6a88 -[NSArray(NSArray) initWithCoder:] + 152
35  Foundation                              0x1bbbfeb4c _decodeObjectBinary + 2436
36  Foundation                              0x1bbbd748c -[NSKeyedUnarchiver _decodeArrayOfObjectsForKey:] + 1592
37  Foundation                              0x1bbbd6a88 -[NSArray(NSArray) initWithCoder:] + 152
38  Foundation                              0x1bbbfeb4c _decodeObjectBinary + 2436
39  Foundation                              0x1bbbd748c -[NSKeyedUnarchiver _decodeArrayOfObjectsForKey:] + 1592
40  Foundation                              0x1bbbd6a88 -[NSArray(NSArray) initWithCoder:] + 152
41  Foundation                              0x1bbbfeb4c _decodeObjectBinary + 2436
42  Foundation                              0x1bbbd748c -[NSKeyedUnarchiver _decodeArrayOfObjectsForKey:] + 1592
43  Foundation                              0x1bbbd6a88 -[NSArray(NSArray) initWithCoder:] + 152
44  Foundation                              0x1bbbfeb4c _decodeObjectBinary + 2436
45  Foundation                              0x1bbbd748c -[NSKeyedUnarchiver _decodeArrayOfObjectsForKey:] + 1592
46  Foundation                              0x1bbbd6a88 -[NSArray(NSArray) initWithCoder:] + 152
47  Foundation                              0x1bbbfeb4c _decodeObjectBinary + 2436
48  Foundation                              0x1bbbd748c -[NSKeyedUnarchiver _decodeArrayOfObjectsForKey:] + 1592
49  Foundation                              0x1bbbd6a88 -[NSArray(NSArray) initWithCoder:] + 152
50  Foundation                              0x1bbbfeb4c _decodeObjectBinary + 2436
51  Foundation                              0x1bbbd748c -[NSKeyedUnarchiver _decodeArrayOfObjectsForKey:] + 1592
52  Foundation                              0x1bbbd6a88 -[NSArray(NSArray) initWithCoder:] + 152
53  Foundation                              0x1bbbfeb4c _decodeObjectBinary + 2436
54  Foundation                              0x1bbbd748c -[NSKeyedUnarchiver _decodeArrayOfObjectsForKey:] + 1592
55  Foundation                              0x1bbbd6a88 -[NSArray(NSArray) initWithCoder:] + 152
```

Ultimately, our analysis of the crash logs revealed the following:

- 25 crashes of the homed process in under 19 minutes
- Followed by a 30-minute break
- 35 crashes of the MessagesBlastDoorService in less than 28 Minutes

homed-2023-08-23-095121.000.ips
homed-2023-08-23-095121.ips
homed-2023-08-23-095132.ips
homed-2023-08-23-095133.ips
homed-2023-08-23-095145.ips
homed-2023-08-23-095154.ips
homed-2023-08-23-095205.ips
homed-2023-08-23-095215.ips
homed-2023-08-23-095307.000.ips
homed-2023-08-23-095307.ips
homed-2023-08-23-095318.ips
homed-2023-08-23-095328.ips
homed-2023-08-23-095339.ips
homed-2023-08-23-095350.ips
homed-2023-08-23-095400.ips
homed-2023-08-23-095410.ips
homed-2023-08-23-095421.ips
homed-2023-08-23-095431.ips
homed-2023-08-23-095442.ips
homed-2023-08-23-095453.ips
homed-2023-08-23-095504.ips
homed-2023-08-23-095514.ips
homed-2023-08-23-095524.ips
homed-2023-08-23-095537.ips
homed-2023-08-23-100521.ips
MessagesBlastDoorService-2023-08-23-103804.000.ips
MessagesBlastDoorService-2023-08-23-103804.ips
MessagesBlastDoorService-2023-08-23-103817.ips
MessagesBlastDoorService-2023-08-23-103824.ips
MessagesBlastDoorService-2023-08-23-103834.ips
MessagesBlastDoorService-2023-08-23-103854.ips
MessagesBlastDoorService-2023-08-23-103956.000.ips
MessagesBlastDoorService-2023-08-23-103956.ips
MessagesBlastDoorService-2023-08-23-104009.ips
MessagesBlastDoorService-2023-08-23-104047.ips
MessagesBlastDoorService-2023-08-23-104131.ips
MessagesBlastDoorService-2023-08-23-104248.ips
MessagesBlastDoorService-2023-08-23-104616.ips
MessagesBlastDoorService-2023-08-23-104626.ips
MessagesBlastDoorService-2023-08-23-104637.ips
MessagesBlastDoorService-2023-08-23-104657.ips
MessagesBlastDoorService-2023-08-23-104737.ips
MessagesBlastDoorService-2023-08-23-104857.ips
MessagesBlastDoorService-2023-08-23-105234.000.ips
MessagesBlastDoorService-2023-08-23-105234.ips
MessagesBlastDoorService-2023-08-23-105246.ips
MessagesBlastDoorService-2023-08-23-105304.ips
MessagesBlastDoorService-2023-08-23-105425.ips
MessagesBlastDoorService-2023-08-23-105439.ips
MessagesBlastDoorService-2023-08-23-105459.ips
MessagesBlastDoorService-2023-08-23-110036.ips
MessagesBlastDoorService-2023-08-23-110049.ips
MessagesBlastDoorService-2023-08-23-110107.ips
MessagesBlastDoorService-2023-08-23-110654.ips

Does this mean the exploit attempt was successful? Is it Pegasus or something else trying to copy NSO's exploits? To answer these questions, we next looked at the iTunes backup. For step-by-step instructions on how to create forensic artifacts using iTunes, check out my talk from Objective by the Sea v6.0.

We started looking at the timestamps in the iTunes backup that correlated to the suspicious cash logs. However, without a good strategy, backup analysis can be very time-consuming due to the overwhelming volume of data. Going through all of the messaging data, thousands of pictures, and other databases in the backup can easily add up to thousands of lines of text for inspection.

In this blog post, we're not going to go through all of the strategies we use, but we are sharing one that leads to the same outcome as taking the reference point from the crash logs.

To Find a Needle in a Haystack, Remove the Hay

This strategy requires an understanding of certain processes involved in the transfer of data on the device, such as ***IMTransferAgent***, which downloads iMessage Attachments from the Apple Server. This process often occurs before attachments are stored on disk and was highlighted in Kaspersky's analysis of Operation Triangulation.

Looking for instances of this process is a good place to start, but be mindful that iTunes backups include two sources of network behavior associated with this process: DataUsage.sqlite and OSAnalyticsADDaily. The former only stores network usage for cellular data and the latter stores the last time the process used network data on wifi or cellular. If the device does not use cellular data or is often on wifi, there won't be much to find through this approach.

Fortunately, in this case, our search for ***IMTransferAgent*** found multiple results and one of them correlates precisely to when the ***MessagesBlastDoorService*** crashes appeared.

2023-08-23 10:06:53.340859	Datausage	live_usage	IMTransferAgent/com.apple.datausage.messages (Bundle ID: com.apple.datausage.messages, ID: 488) WIFI IN: 0.0, WIFI OUT: 0.0 - WWAN IN: 32561646.0, WWAN OUT: 621714.0
2023-08-23 10:22:51.720639	Datausage	current_usage	IMTransferAgent/com.apple.datausage.messages (Bundle ID: com.apple.datausage.messages, ID: 488)

Because ***IMTransferAgent*** is used to transport data to and from iMessage servers, it makes sense to check if there is any file activity around the same time. The iTunes backup shows that 8 files were received shortly after the moment ***IMTransferAgent*** was used and MessagesBlastDoorService crashed. Moreover, those files did not have an identified sender; they were "Sent from None."

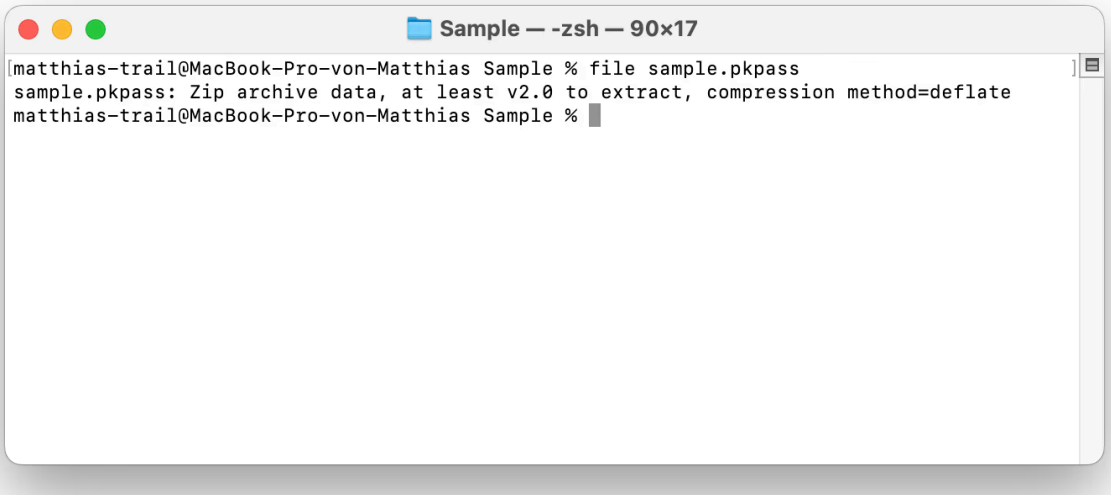
The received items were all named "sample.pkpass" and they have a combined size of approximately 175 KB. As a reminder, pkpass is the file extension for PassKit Passes. There's a nice write-up about it from Apple here. Additionally, CitizenLabs' report also mentions that the BLASTPASS exploit chain contained a PassKit file that was sent over iMessage.

The plot thickens!

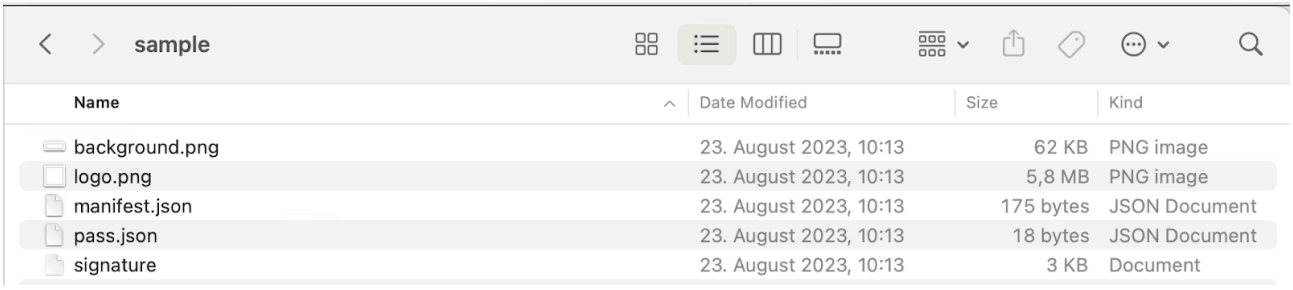
Inside the PassKit File: Findings and Conclusions

Next, let's look inside the file. We located the sample in the iTunes backup – a possible indication that the exploit chain was unsuccessful or it was not NSO because they typically try to clean up the iMessage Attachment folder after a successful exploitation.

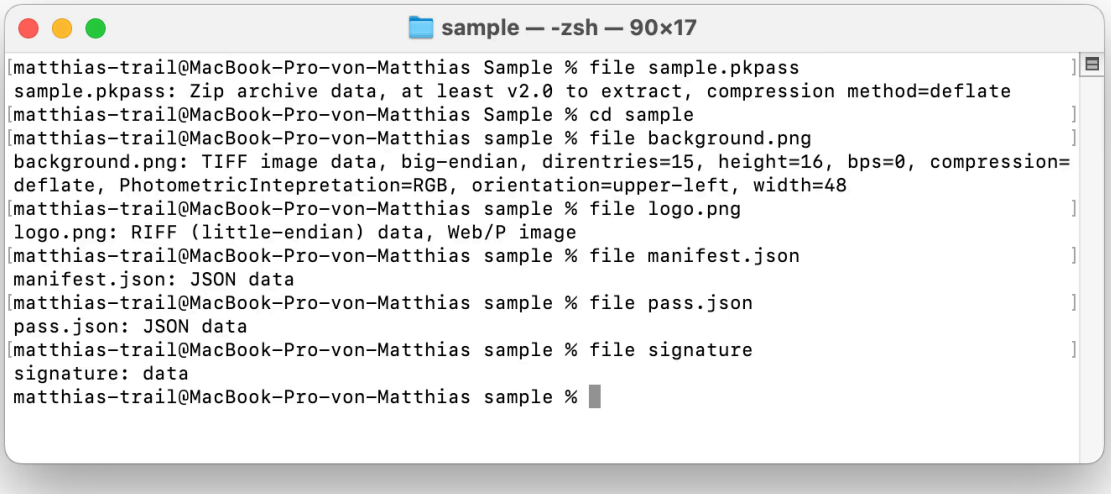
We started by asking *file* what the sample.pkpass is:



Decompressing the 175KB file expands it to a 5.9MB archive containing the following files:



Running *file* on all of those yields this:

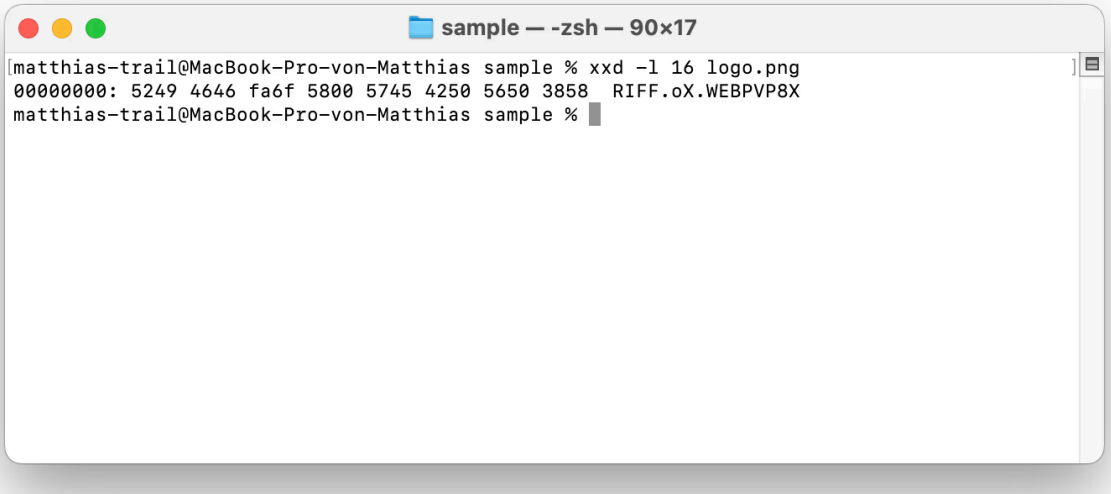


Pass.json is not very interesting; it merely indicates this is a PKpass file. Yes, thank you.

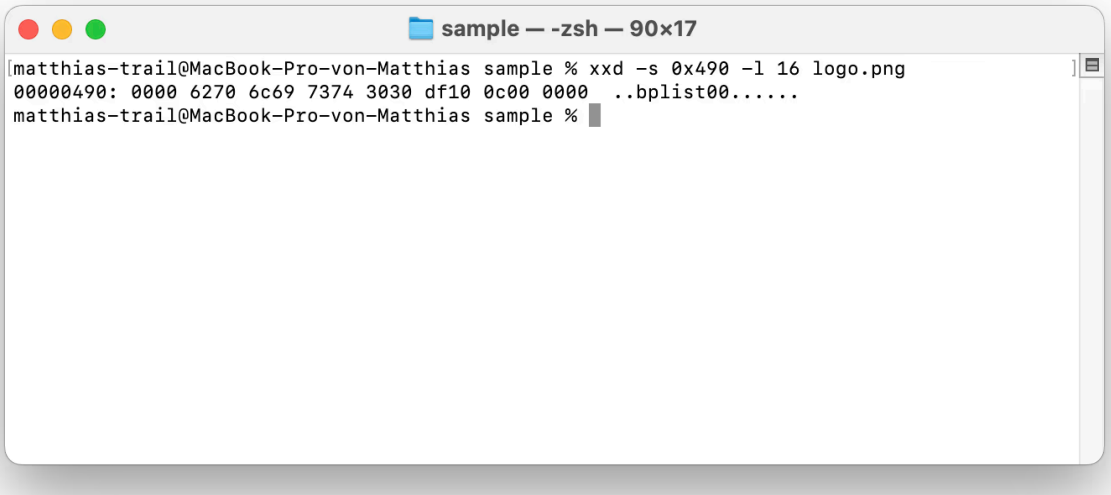
Signature contains the signature of the file and *manifest* contains a list of the files included in the archive as well as their SHA1 hash. *background.png* is essentially an empty picture and not interesting for our investigation.

But *logo.png* is different. It's a large 5.8MB picture which is not a png, but a WebP picture, another reference to the BLASTPASS exploit chain.

Looking at the raw hex bytes with *xxd* confirms that we indeed have a WebP file:



The file also includes the starting header for an Apple binary plist format: bplist



That’s interesting!

Binary Plist are usually PropertyList documents based on a NSKeyedArchiver, which gives us another link between the sample.pkpass file and the MessagesBlastDoorService crashes because many of the functions in the backtrace were functions related to unarchiving NSKeyedArchiver objects!

Our analysis continued beyond this point. At this stage we suspected that NSO (or someone impersonating their playbook) first tried to use the PWNYOURHOME exploit and when that failed, shifted to BLASTPASS.

If you are interested in more details about our BLASTPASS analysis or our Threat Hunter tool, please check out my briefing at Black Hat Asia in April. We will also share more about it here on the blog!

Source: https://www.iverify.io/post/clipping-wings-our-analysis-of-a-pegasus-spyware-sample