Dusting off Old Fingerprints: NSO Group's Unknown MMS Hack

By Cathal McDaid

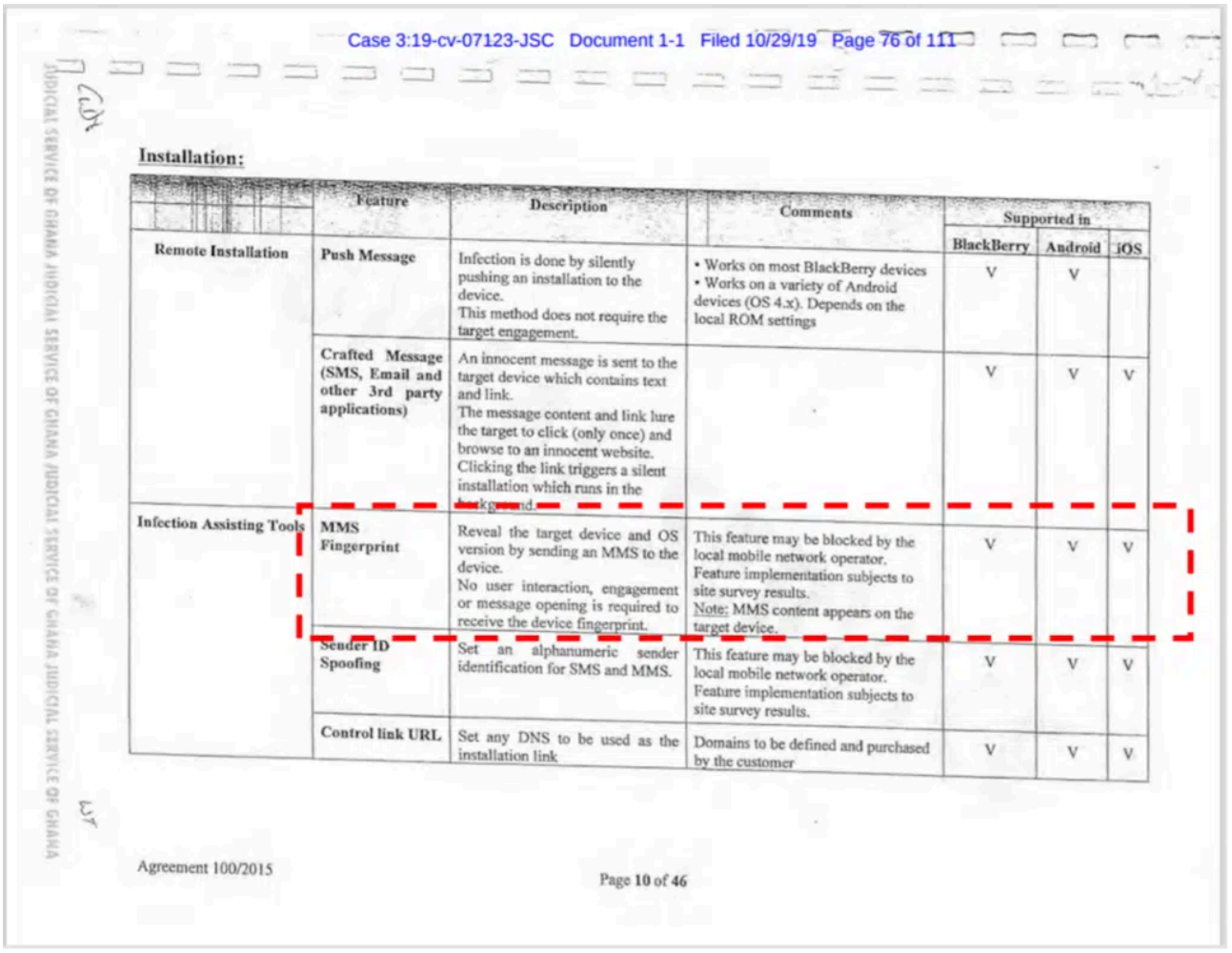
Published: 2024-02-15 · Archived: 2026-04-05 13:26:19 UTC

In the ever-evolving landscape of mobile network security, it's essential to stay vigilant and informed about the latest threats. However to do so, sometimes we end up looking in places we would not normally look. Today, we delve into a previously unknown mobile network attack known as the "MMS Fingerprint" attack, reportedly used by NSO Group, a well-known actor in the realm of surveillance technologies. How we found this attack – which had essentially been hiding in plain sight – and how this attack might work, takes a bit of explaining.

Hiding in Plain Sight

In May 2019, WhatsApp, a widely used encrypted messaging platform, discovered a vulnerability in its system that allowed attackers to install Pegasus spyware on users' devices. The flaw was exploited through a WhatsApp voice call, and it could compromise a device without the owner's knowledge. WhatsApp blamed NSO Group for exploiting the vulnerability, which reportedly targeted a range of individuals, including journalists, human rights activists, lawyers, and government officials. The attack was global in scope, with victims in various countries. Subsequently, in October 2019 WhatsApp sued NSO Group. Since then, appeals by NSO group to have the case stop have failed both in the US appeal court and the US Supreme Court.

While interesting from a strategic and high-level viewpoint, at first glance there is nothing new here as this has been in the public domain for years. However, as part of its court case, WhatsApp/Facebook committed into evidence in October 2019 several exhibits. Most of this material was examined and discussed in the public domain. However, what was not discussed were some specific details within a copy of a contract between a NSO Group reseller and the telecom regulator of Ghana. Within that contract, in Exhibit A-1, was a list of "Features and Capabilities" offered by NSO Group. To telecom security specialists like us, these features were largely known, however there was a feature title that was (at first sight) unknown. This was the entry termed **"MMS Fingerprint".**



Case 3:19-cv-07123-JSC Document 1-1 Filed 10/29/19 Page 76 of 111

Installation:

	Feature	Description	Comments	Supported in		
				BlackBerry	Android	iOS
Remote Installation	Push Message	Infection is done by silently pushing an installation to the device. This method does not require the target engagement.	• Works on most BlackBerry devices • Works on a variety of Android devices (OS 4.x). Depends on the local ROM settings	V	V	
	Crafted Message (SMS, Email and other 3rd party applications)	An innocent message is sent to the target device which contains text and link. The message content and link lure the target to click (only once) and browse to an innocent website. Clicking the link triggers a silent installation which runs in the background.		V	V	V
Infection Assisting Tools	MMS Fingerprint	Reveal the target device and OS version by sending an MMS to the device. No user interaction, engagement or message opening is required to receive the device fingerprint.	This feature may be blocked by the local mobile network operator. Feature implementation subjects to site survey results. Note: MMS content appears on the target device.	V	V	V
	Sender ID Spoofing	Set an alphanumeric sender identification for SMS and MMS.	This feature may be blocked by the local mobile network operator. Feature implementation subjects to site survey results.	V	V	V
	Control link URL	Set any DNS to be used as the installation link	Domains to be defined and purchased by the customer	V	V	V

Agreement 100/2015

Page 10 of 46

"MMS Fingerprint" is not a term that was known about in the industry. At the time of writing a Google search doesn't find any other entries with this term, other than the court case itself. While we always must consider that NSO Group may simply be "inventing" or exaggerating the capabilities it claims to have (in our experience, surveillance companies regularly over-promise their capabilities), the fact this was on a contract rather than an advertisement suggests that it was more likely to be for real.

How Could an MMS Fingerprint Attack Work?

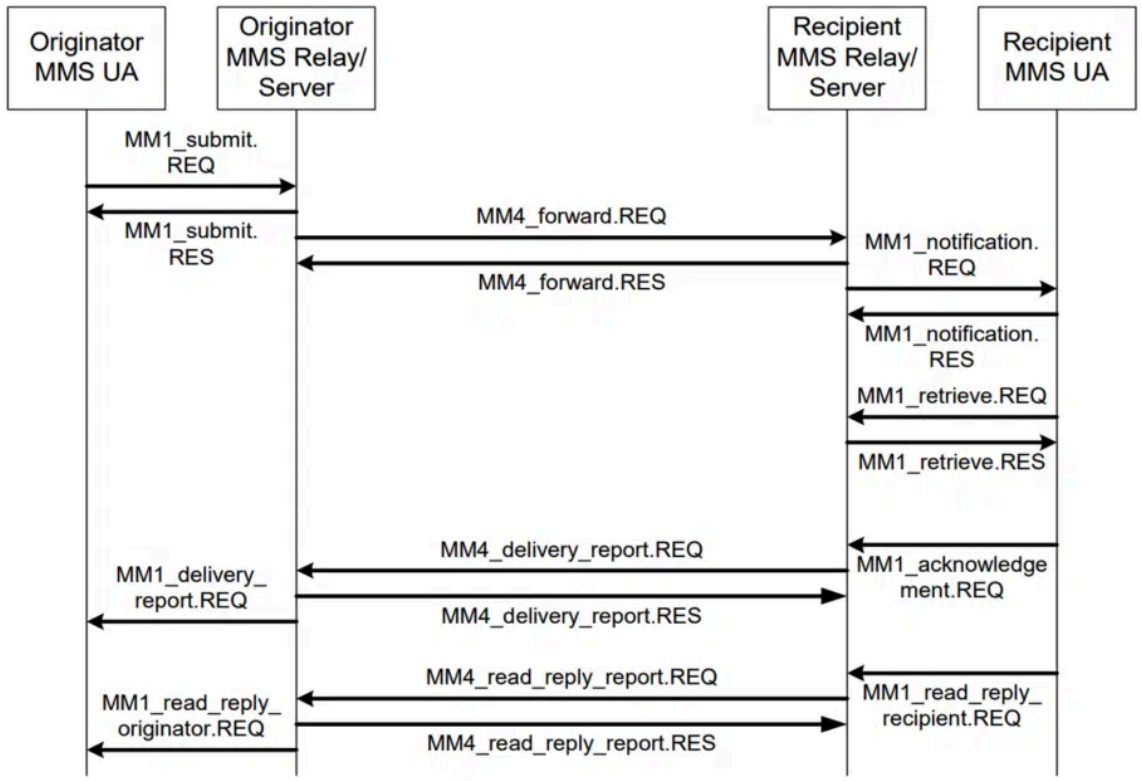
Going forward on that basis, I decided to consider how exactly an MMS Fingerprint **could** work. Next to no technical details were given, other than it stated it would:

- *Reveal the target device and OS version by sending an MMS to the device.*
- *No user interaction, engagement or message opening is required to receive the device fingerprint.*

A starting point is the fact that Blackberry, Android, iOS devices were all listed as possible meant an OS-specific hack seemed unlikely. As a result we were probably looking at something vulnerable within the MMS flow itself. In looking at the MMS flow, I concluded that perhaps – despite its name – the attack wasn't happening over MMS, but rather via something else. To explain this, we have to look at the overall MMS flow itself, which is somewhat 'messy', and that confusingly, sometimes the MMS flow is not using MMS.

Sometimes an MMS is not an MMS

Below is a diagram of a complete end to end MMS transfer. There are several main stages in an MMS flow.



1. Initially the MMS message is submitted by the sender mobile device in a MM1_submit.REQ to the sender's MMSC

2. This then is forwarded by the sender's MMSC to the recipient's MMSC over the MM4 interface, using a MM4_forward.REQ. As a side note, this is different from how SMS works where the Sender's SMSC is responsible for both submission and delivery, so straight away things are more complex

3. The recipient's MMSC then notifies the recipient device that a MMS is waiting for it via a MM1_notification.REQ

4. The recipient then retrieves the waiting MMS from its MMSC via a MM1_retrieve.REQ & MM1_retrieve.RES

5. The later stages then involve a delivery report being sent to the sender MMSC/sender device once the MMS has been retrieved,

6. Finally, an (optional) read report is sent to the sender MMSC/sender device once the MMS has been read.

So far, so (reasonably) clear. However things are less straight-forward when you being to investigate. One complexity in MMS is that when it was developed, the MMS standards designers needed a way for the recipient device to be notified there was an MMS waiting (for the recipient) in a time where there was no guarantee that every device was MMS compatible. At the same time they did not want a system where the recipient mobile

device was connected to the data channel when it did not need to be. As a result, MM1_notification.REQ used a system which would be supported by the vast majority of handsets, that is SMS. This also meant that the recipient device would only connect to the data channel once it was successfully paged over SMS. Technically speaking, MM1_notification.REQ is a special type of SMS, known as a binary SMS (WSP Push), that is used to notify to the recipient MMS device's user agent that an MMS message is waiting in the recipient MMSC for retrieval (using the field *X-MMS-Content-Location*).

In addition, the subsequent MM1_retrieve.REQ, isn't really an MMS 'message' either. What this is, is a HTTP GET to the URL address contained in the X-MMS-Content-Location field in the previous MM1_notification.REQ. This GET should normally go to the recipient's "mms" APN, to retrieve the MMS from the MMSC. The interesting thing here, is that within this HTTP GET, user device information is included. It was suspected that this may be the point that targeted device information could be leaked, and the MMS Fingerprint could be "lifted".

Looking for Fingerprints and Why

This was certainly the assumption, however we needed to test the attack to be sure. To do this, we obtained some sample SIM cards from a random western European operator, and after some trial and error successfully sent MM1_notification.REQs (binary SMSs) to it. In these MM1_notification.REQs we set the content location to be a URL that pointed to a web server we controlled. Sure enough we were able to get the target device that received our MM1_notification.REQ to automatically do a GET to our URL, once it received the attack MM1_notification.REQ (binary SMS). This HTTP GET exposed the device's *UserAgent* and *x-wap-profile* fields.

A screenshot is below of a Wireshark decode of the MMS notification we sent, and the GET we subsequently received. This we believe is how an attacker would plan to execute an "MMS Fingerprint" attack, and we were able to show it was possible in real life.

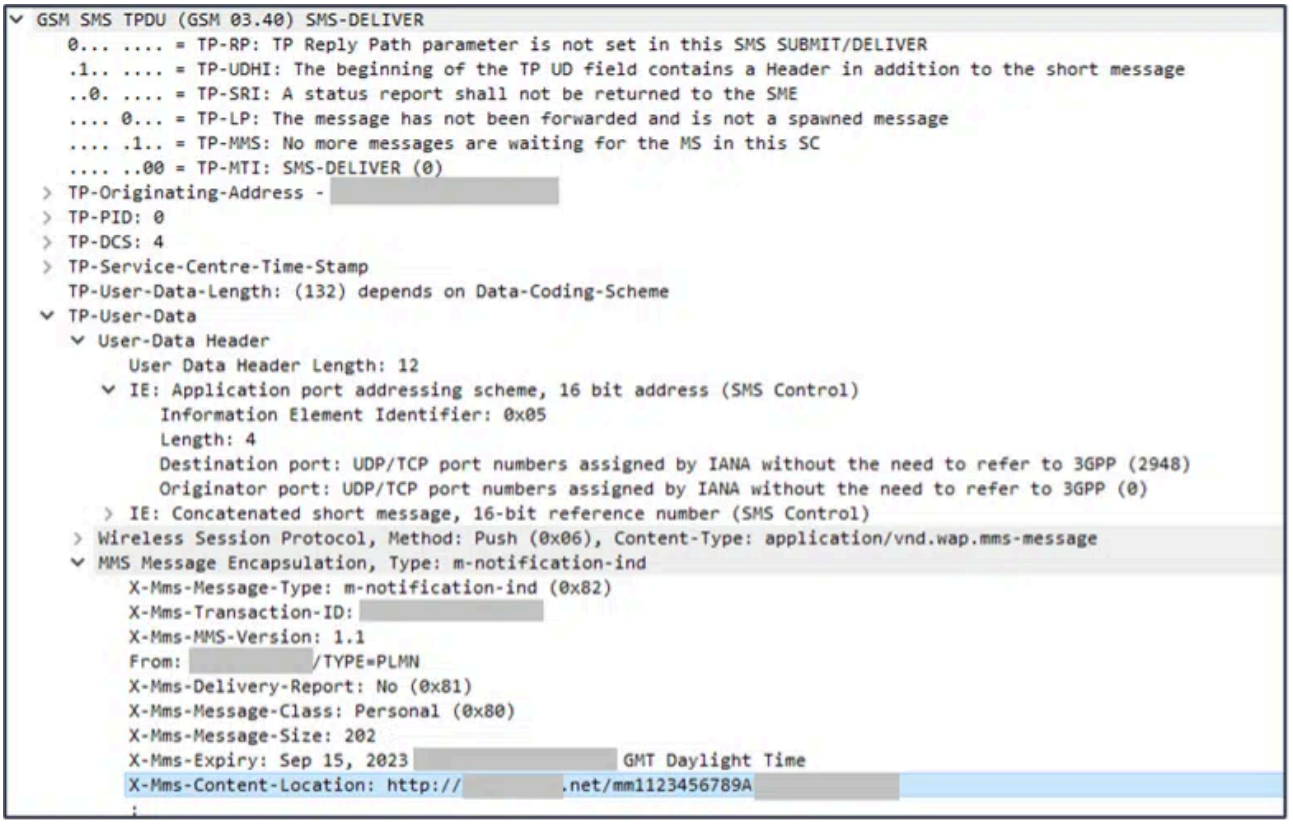


Figure 1: Initial Attack MM1_ notifications.ind(MM1_notification.REQ) sent to target

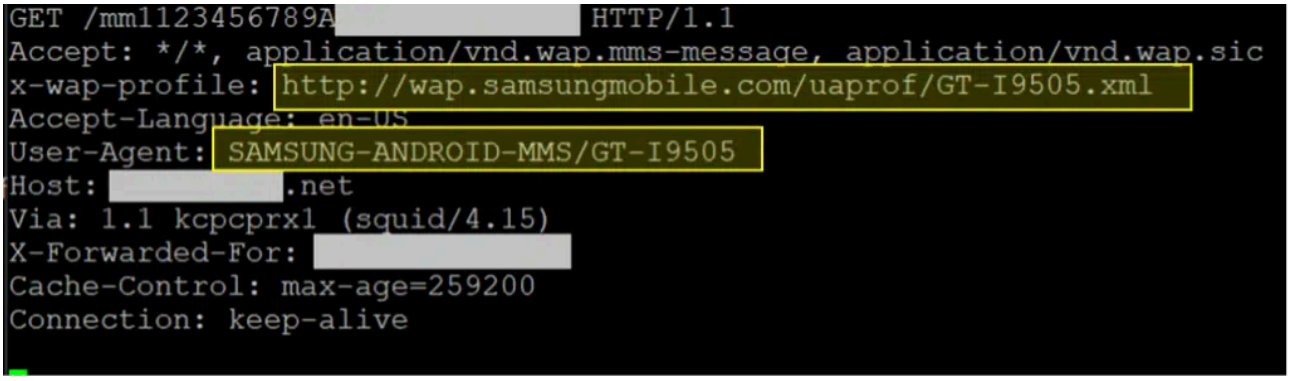


Figure 2: Subsequent MM1_retrieve.REQ(HTTP GET) received from targeted handset. Target device information in yellow

The next question to ask is why you would do this – what value is there in knowing the *UserAgent* and *x-wap-profile*. To answer that its worth explaining what they are:

- The (MMS) UserAgent is a string that typically identifies the OS and device – in this case a Samsung phone running Android.
- x-wap-profile points to a *UAProf* (User Agent Profile) file that describes the capabilities of a mobile handset.

Both of these can be very useful for malicious actors. Attackers could use this information to exploit specific vulnerabilities or tailor malicious payloads (such as the Pegasus exploit) to the recipient device type. Or it could

be used to help craft phishing campaigns against the human using the device more effectively. We have observed before that surveillance companies, when presented with the chance to get device information, invariably do. For example, the Simjacker attackers requested to get IMEI (device identifier) as well as location information over 93% of the time. In this case, UserAgent could potentially be more useful than IMEI as they reflect what's installed on the device, rather than the static info contained in the IMEI field. One important note though is that the *MMS* UserAgent, is different from the *browser* UserAgent which more people may be aware of (and which has its own privacy concerns and changes in). Both strings identify devices requesting online content but they represent different useragents running on the device.

Details on the Attack

The next question to answer is, having replicated the attack, do we think it is being used today? Thankfully, in our investigation over the last few months we did not observe any use of this vulnerability in the wild today. We are not present in every operator in the world so it may be used, but we did not observe any of the known surveillance company sources we monitor using this technique. This may not be unexpected – as the contract document is from 2015, and attackers may no longer use this method.

Second – NSO group state in their documentation that there is a caveat to the MMS Fingerprint they offer, which is "*Note: MMS content appears on the target device*". This was the same for us in our initial tests of our MM1_notification.REQ attacks. In these, an MMS message notification is received by the target device which the phone owner can see. However, we were able to optimise the attack, and better conceal it, by changing the binary SMS to be a *silent SMS*. This was done by setting a TP-PID value of 0x40. As a result, no MMS content would appear on the targeted device, and the targeted person would not see anything on their phone. If the MMS fingerprint attack was done by using MM1_notification.REQ then it is hard to believe that an attacker wouldn't have known about this optimisation. One reason though for not using this is that encoding the message this way (as a silent SMS) would cause the message to stand out more on mobile operator networks, and so be more easily blocked by the recipient mobile operator.

Wiping Your Fingerprints

This takes us to the question on how to stop these attacks. In general, it is not that difficult to block the MM1_notification.REQ attack we tested, however there are some complications. Similarly, NSO themselves make it clear that when they list MMS Fingerprint that *"This feature may be blocked by the local mobile network operator"* .

Firstly, on the device itself, subscribers are not helpless. One technique is that mobile subscribers could disable MMS auto-retrieval on their handset, to prevent the device automatically connecting as well. Typically this is configured on devices to download MMS automatically if the subscriber is home, and with a user prompt while abroad. But subscribers could set it to always require a user prompt. This same recommendation has been made for other MMS exploits like Stagefright. This however should not be the recommended first line of defence, and some devices do not allow you to modify this in any case.

The recommended first line of defence would be on the network side. The vast majority of mobile operators today filter these Binary SMS/MM1_notification messages being sent, however what they are less strict on is them

being received. This is because in the past some international/intercarrier MMS handling systems would route MMS messages from one operator to another not via MM4, but by sending the MM1 notification to the subscribers of a different operator. In our analysis we still observed this being done today for 'legitimate' MMS in some networks. This was probably much more common in the past when some operators had MMS, and others did not. As a result, operators would need to investigate themselves before implement blocking of this. Inbound message Blocking would also need to handle outbound roamers, although this is standard functionality of any serious messaging security solution. For more assistance, the GSMA has produced a document called *FS.42* which contains recommendations on how mobile operator can protect themselves from Binary SMS attacks, of which this attack would fall under.

Another line of defence on the network side is what could be done after a malicious binary SMS message is received. Blocking need not only mean stopping the MM1_notification.REQ being received but other techniques could also be used to impede the attack. For example, mobile operators could look at restricting internet access via the MMS ports from handsets, this would mean that even if the message was received, it would not connect to the attacker-controlled IP address.

Conclusion

NSO Group have offered in the past a previously unknown and undiscussed telecom attack technique called MMS Fingerprint. Based on the description and our knowledge and experience in this space we were able to recreate and test an attack using the MMS flow which seems to match what was described in the legal exhibit. This attack is a stark reminder of how the mobile ecosystem can continually be threatened by a poorly understood but still heavily used technology that was developed without strong security in mind. While the MMS fingerprint attack that NSO Group described is not extensive – only returning device info – it was useful enough to have been offered as a named attack technique to governments.

The history of Binary SMS attacks has given us a slow but very steady list of new reported vulnerabilities over the last 20 years, of which this one is just the latest example. Companies like NSO group have an extensive track record of executing attacks against mobile phone users for many years, and Mobile Operators should evaluate their protection in place against this and other Binary SMS attacks.

With thanks to Fredrik Söderlund for his assistance.
