www.mandiant.com /resources/blog/esxi-hypervisors-malware-persistence

Bad VIB(E)s Part One: Investigating Novel Malware Persistence Within ESXi Hypervisors

As endpoint detection and response (EDR) solutions improve malware detection efficacy on Windows systems, certain state-sponsored threat actors have shifted to developing and deploying malware on systems that do not generally support EDR such as network appliances, SAN arrays, and VMware ESXi servers.

Earlier this year, Mandiant identified a novel malware ecosystem impacting VMware ESXi, Linux vCenter servers, and Windows virtual machines that enables a threat actor to take the following actions:

1. Maintain persistent administrative access to the hypervisor
2. Send commands to the hypervisor that will be routed to the guest VM for execution
3. Transfer files between the ESXi hypervisor and guest machines running beneath it
4. Tamper with logging services on the hypervisor
5. Execute arbitrary commands from one guest VM to another guest VM running on the same hypervisor

This malware ecosystem was initially detected during an intrusion investigation when Mandiant identified attacker commands sourced from the legitimate VMware Tools process, `vmtoolsd.exe`, on a Windows virtual machine hosted on a VMware ESXi hypervisor. Mandiant analyzed the boot profile for the ESXi hypervisors and identified a never-before-seen technique in which a threat actor leveraged malicious vSphere Installation Bundles ("VIBs") to install multiple backdoors on the ESXi hypervisors. We call these backdoors VIRTUALPITA and VIRTUALPIE (Figure 1).

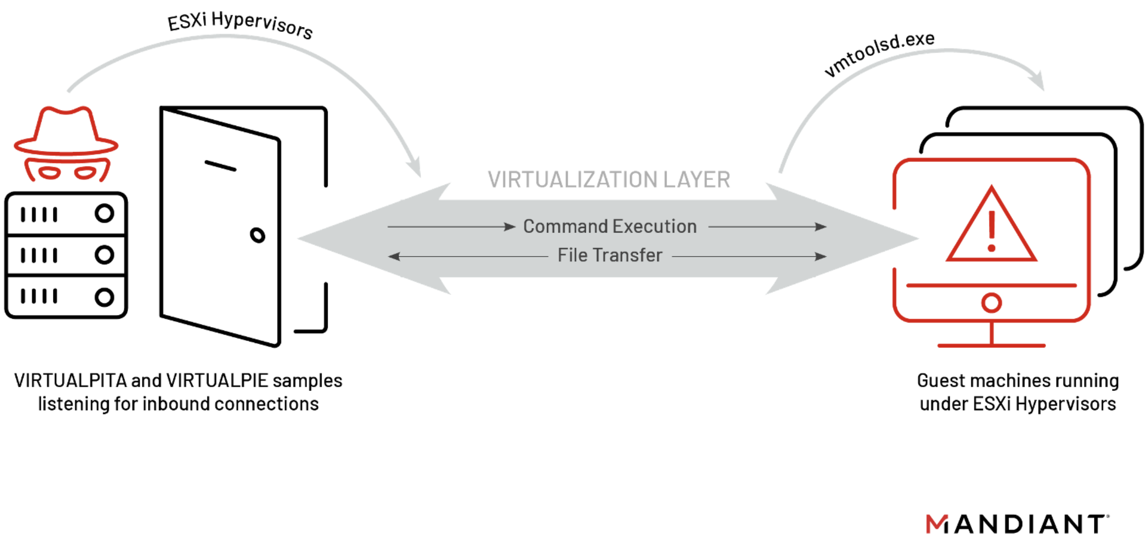

Figure 1: Visualization of ESXi attack path

It is important to highlight that this is not an external remote code execution vulnerability; the attacker needs admin-level privileges to the ESXi hypervisor before they can deploy malware. Mandiant has no evidence of a zero-day vulnerability being used to gain initial access or deploy the malicious VIBs at the time of writing this post.

Details on how to manually detect if malicious or anomalous VIBs are currently installed in your ESXi environment are outlined in our hardening blog post. VMware has released additional information on protecting vSphere.

vSphere Installation Bundles (VIB)

Mandiant identified two (2) new malware families installed through malicious vSphere Installation Bundles (VIBs), which we have named VIRTUALPITA and VIRTUALPIE.

VMware VIBs are collections of files that are designed to facilitate software distribution and virtual system management. Since ESXi utilizes an in-memory filesystem, file edits are not saved across reboots. A VIB package can be used to create startup tasks, custom firewall rules, or deploy custom binaries upon the restart of an ESXi machine. These packages are generally utilized by administrators to deploy updates and maintain systems; however, this attacker was seen leveraging the packages as a persistence mechanism to maintain access across ESXi hypervisors.

VIBs can be broken down into three (3) components:

- An XML descriptor file
- A “VIB payload” (.vgz archive)
- A signature file – A digital signature used to verify the host acceptance level of a VIB

The XML Descriptor File is a config which contains references to the following:

- The payload to be installed
- VIB metadata, such as the name and install date
- The signature file that belongs to the VIB

The VIB payload is a .vgz archive which contains directories and files that will be created on the ESXi machine through the VIB. These files can then be called on to execute on boot when the VIBs are loaded.

The signature file is used to verify the host acceptance level of a VIB. The acceptance level is the digital signature system used by VMware to specify what testing has been done by VMware or partners before a VIB is published. Acceptance levels are set for hosts, image profiles, and individual VIBs. The four (4) acceptance levels along with their XML Descriptor short names are listed below:

- VMWareCertified (certified)
- VMwareAccepted (accepted)
- PartnerSupported (partner)
- CommunitySupported (community)

Per VMware documentation, the default minimum acceptance level a VIB needs to be installed on a ESXi host is `PartnerSupported`. This acceptance level indicates the VIBs are published by a partner that VMware trusts. While this is the default acceptance level, it can be changed manually by an ESXi administrative account. The command used to install VIBs `esxcli software vib install` does not normally allow installations below the minimum acceptance level, but the --force flag can be used to ignore any systems acceptance level requirements when installing the VIB.

The malicious VIBs observed were labelled as `PartnerSupported`. Mandiant’s review of the Signature Files determined they were empty, and that an attacker modified the XML descriptor file to change the `acceptance-level` field from `community` to `partner`. A `CommunitySupported` acceptance-level indicates that the VIB was created by a third party which was not reviewed nor signed by VMware or its trusted partners. This indicated the attacker masqueraded these VIB files as `PartnerSupported` even though they only met the requirements of a `CommunitySupported` VIB. This also indicated that the VIB was created by an individual or company outside of VMware partner programs and has not gone through any VMware-approved testing program. Figure 2 contains an excerpt of the modified XML descriptor file that was identified.

```xml
<vib version="5.0">
    <type>bootbank</type>
    <name>ata-pata-pdc20211</name>
    <version>1.0-3vmw.670.0.0.8169922</version>
    <vendor>VMW</vendor>
    <summary>[Fling] ata_pdc20211: ata driver for VMware ESX</summary>
    <description>Promise PATA Driver</description>
    <release-date>2015-02-06T20:09:15.653618+00:00</release-date>
    <urls/>
    <relationships>
        <depends/>
        <conflicts/>
        <replaces/>
        <provides/>
        <compatibleWith/>
    </relationships>
    <software-tags/>
    <installdate>REDACTED</installdate>
    <system-requires>
        <maintenance-mode>false</maintenance-mode>
    </system-requires>
    <file-list>
        <file>etc/rc.local.d/vmware_rhttpio.sh</file>
        <file>usr/lib/vmware/weasel/consoleui/rhttpproxy-IO</file>
    </file-list>
    <acceptance-level>partner</acceptance-level>
    <live-install-allowed>true</live-install-allowed>
    <live-remove-allowed>true</live-remove-allowed>
    <cimom-restart>false</cimom-restart>
    <stateless-ready>true</stateless-ready>
    <overlay>false</overlay>
    <payloads>
        <payload name="payload1" size="18042" type="vgz">
            <checksum checksum-type="sha-256">a03b96cbf7ae79ac4084fe6fb64822aefb4cded814bf7b80b9abcab50fc1f926</checksum>
            <checksum checksum-type="sha-1" verify-process="gunzip">5bea11ea46729f14ae4d9cae5e57ce3a2d3f298b</checksum>
        </payload>
    </payloads>
</vib>
```

Figure 2: Modified Descriptor XML

While the `acceptance-level` field was modified in the Descriptor XML by the attacker, the ESXi system still did not allow for a falsified VIB file to be installed below the minimal set acceptance level. To circumvent this, the attacker abused the `--force` flag to install malicious `CommunitySupported` VIBs.

Testing confirmed that modified fields from the XML descriptor file reflected the changes made in the output of commands used to list and verify VIBs. This included the modification of the `<acceptance-level>` field which tricks the command, `esxcli software vib list`, into displaying the incorrect acceptance level of the installed VIBs. The VMware command, `esxcli software vib signature verify`, verifies the signatures of installed VIB packages and displays the following fields:

- VIB Name
- Version
- Vendor
- Acceptance Level
- The result of the VIB's signature verification

Mandiant confirmed that this command detected when these acceptance levels were falsified, which identified the malicious VIBs. This command displays the acceptance level specified by the XML descriptor file, but the `Signature Verification` column clarifies if the signature file did not match the respective Descriptor XML. If the signature cannot be verified, the `Signature Verification` column will contain the value `Signature`

`Not Available: Host may have been upgraded from an older ESXi version`. An example of this can be seen in Figure 3.

```
[root@<REDACTED>:~] esxcli software vib signature verify

Name                          Version                              Vendor  Acceptance Level  Signature Verification
----------------------------  -----------------------------------  ------  ----------------  -----------------------------------------------------------------------------
ata-libata-92                 3.00.9.2-16vmw.670.0.0.8169922       VMW     VMwareCertified   Succeeded
ata-pata-amd                  0.3.10-3vmw.670.0.0.8169922          VMW     VMwareCertified   Succeeded
ata-pata-atiixp               0.4.6-4vmw.670.0.0.8169922           VMW     VMwareCertified   Succeeded
ata-pata-cmd64x               0.2.5-3vmw.670.0.0.8169922           VMW     VMwareCertified   Succeeded
ata-pata-hpt3x2n              0.3.4-3vmw.670.0.0.8169922           VMW     VMwareCertified   Succeeded
ata-pata-pdc20211             1.0-3vmw.670.0.0.8169922             VMW     PartnerSupported  Signature Not Available: Host may have been upgraded from an older ESXi version
ata-pata-pdc2027x             1.0-3vmw.670.0.0.8169922             VMW     VMwareCertified   Succeeded
ata-pata-serverworks          0.4.3-3vmw.670.0.0.8169922           VMW     VMwareCertified   Succeeded
ata-pata-sil680               0.4.8-3vmw.670.0.0.8169922           VMW     VMwareCertified   Succeeded
ata-pata-via                  0.3.3-2vmw.670.0.0.8169922           VMW     VMwareCertified   Succeeded
block-cciss                   3.6.14-10vmw.670.0.0.8169922         VMW     VMwareCertified   Succeeded
bnxtnet                       20.6.101.7-24vmw.670.3.73.14320388   VMW     VMwareCertified   Succeeded
```

Figure 3: Example of falsified VIB acceptance level being seen in esxcli software vib signature verify

VIRTUALPITA (VMware ESXi)

VIRTUALPITA is a 64-bit passive backdoor that creates a listener on a hardcoded port number on a VMware ESXi server. The backdoor often utilizes VMware service names and ports to masquerade as a legitimate service. It supports arbitrary command execution, file upload and download, and the ability to start and stop `vmsyslogd`. During arbitrary command execution, the malware also sets the environmental variable `HISTFILE` to `0` to further hide activity that occurred on the machine. Variants of this malware were found to listen on a Virtual Machine Communication Interface (VMCI) and log this activity to the file `sysclog`.

VIRTUALPIE (VMware ESXi)

VIRTUALPIE is a lightweight backdoor written in Python that spawns a daemonized IPv6 listener on a hardcoded port on a VMware ESXi server. It supports arbitrary command line execution, file transfer capabilities, and reverse shell capabilities. Communications use a custom protocol and are encrypted using RC4.

The first malicious VIB named `lsu-lsi-lsi-mrarpid-plugin` referenced the payload `lsu_lsi_.v05` (MD5: 2716c60c28cf7f7568f55ac33313468b) which contained the following three (3) files for which details can be found in Table 1:

- /etc/rc.local.d/vmware_local.sh (MD5: bd6e38b6ff85ab02c1a4325e8af29ce4)
- /bin/rdt (MD5: 8e80b40b1298f022c7f3a96599806c43)
- /bin/vmsyslog.py (MD5: 61ab3f6401d60ec36cd3ac980a8deb75)

Table 1: Lsu-lsi-lsi-mrarpid-plugin Malicious VIB contents

File Name	Description
vmware_local.sh	A bash installation script to be placed into `/etc/rc.local.d/` to ensure its actions will be executed upon each bootup of ESXi. Uses the `esxcli` command line utility to enable a firewall rule for backdoor traffic, execute both backdoors, and remove every file created by the VIB from the disk.
rdt	An ELF backdoor (VIRTUALPITA) that creates a listener on the hard coded TCP port 2233. Capable of arbitrary command execution, file transfer capabilities and the ability to start/stop `vmsyslogd`. VMWare documentation recorded this port is normally utilized by the vSAN reliable datagram transport (RDT) service on the ESXi version which was reviewed.
vmsyslog.py	A lightweight backdoor (VIRTUALPIE) written in Python that spawns a daemonized IPv6 listener the hardcoded port 546. Capable of arbitrary command line execution, file transfer capabilities, and reverse shell capabilities. Communications use a custom protocol and are encrypted using RC4.

The second malicious VIB named `ata-pata-pdc20211` referenced the payload `payload1.v00` (MD5: 9ea86dccd5bbde47f8641b62a1eeff07) which contained the following two (2) files for which details can be found in Table 2:

- /etc/rc.local.d/vmware_rhttpio.sh (MD5: 9d5cc1ee99ccb1ec4d20be1cee10173e)
- /usr/lib/vmware/weasel/consoleui/rhttpproxy-io (MD5: 2c28ec2d541f555b2838099ca849f965)

Table 2: Lsu-lsi-lsi-mrarpid-plugin Malicious VIB contents

File Name	Description
vmware_rhttpio.sh	A bash installation script to be placed into `/etc/rc.local.d/` to ensure its actions will be executed upon each bootup of ESXi. This script executes the ELF backdoor
rhttpproxy-io	An ELF backdoor (VIRTUALPITA) that creates a listener on the hard coded VMCI socket port 18098. Capable of arbitrary command execution, file transfer capabilities and the ability to start/stop vmsyslogd. The following sample generates an additional log not seen in other samples which fetches the systems context ID (CID). The generated log `/var/log/sysclog` records in the following format `[<date/timestamp>]\n\r[!]<<PID>>:<CID>:<port>\n\n`

VIRTUALPITA (LINUX)

Mandiant discovered two (2) additional VIRTUALPITA samples listening on TCP port 7475 that were persistent as an `init.d` startup service on Linux vCenter systems. To disguise themselves, the binaries shared the name of the legitimate binary `ksmd`. KSMD (Kernel Same-Page Merging Daemon) is normally in charge of memory-saving de-duplication on Linux and would not be listening on this port. The samples were found under the following directories:

- /usr/libexec/setconf/ksmd (MD5: 744e2a4c1da48869776827d461c2b2ec)
- /usr/bin/ksmd (MD5: 93d50025b81d3dbcb2e25d15cae03428)

These backdoors were capable of arbitrary command execution, file transfer capabilities and the ability to start/stop vmsyslogd.

VIRTUALGATE (Windows)

The Windows guest virtual machines which were hosted by the infected hypervisors also contained a unique malware sample located at `C:\Windows\Temp\avp.exe`. This malware, which we refer to as VIRTUALGATE, is a utility program written in C that is comprised of two (2) parts, a dropper, and the payload. The memory only dropper deobfuscates a second stage DLL payload that uses VMware's virtual machine communication interface (VMCI) sockets to run commands on a guest virtual machine from a hypervisor host, or between guest virtual machines on the same host.

Command Execution through vmtoolsd.exe

Reviewing the ESXi hypervisors during an active attacker event, Mandiant identified an attacker executed commands to the guest machines through the process `/bin/rdt` (VIRTUALPITA). The parent process `/bin/rdt` spawned a bash shell which called on a python script specifying the target guest machine and commands to run as seen in Figure 4.

Figure 4: Commands executed by attacker on ESXi Hypervisor

```
-> /bin/rdt -i

....-> sh -c /bin/sh
```

```
........-> /bin/sh

..........-> python e.py 127.0.0.1 vpxuser <password> <target guest machine>
C:\Windows\system32\cmd.exe /c dir /od /s /a c:\ > C:\Windows\Temp\TS_<REDACTED>.tmp 2>nul
```

Commands passed as arguments into e.py were also seen being executed by the targeted Windows guest machine, running as a child process under `vmtoolsd.exe`. This execution chain can be seen in Figure 5. The parent binary `/bin/rdt` was not present on disk but was able to be recovered by dumping the processes memory of the ESXi hypervisor. The python script that sent out commands to the guest machines, `e.py`, was unable to be recovered.

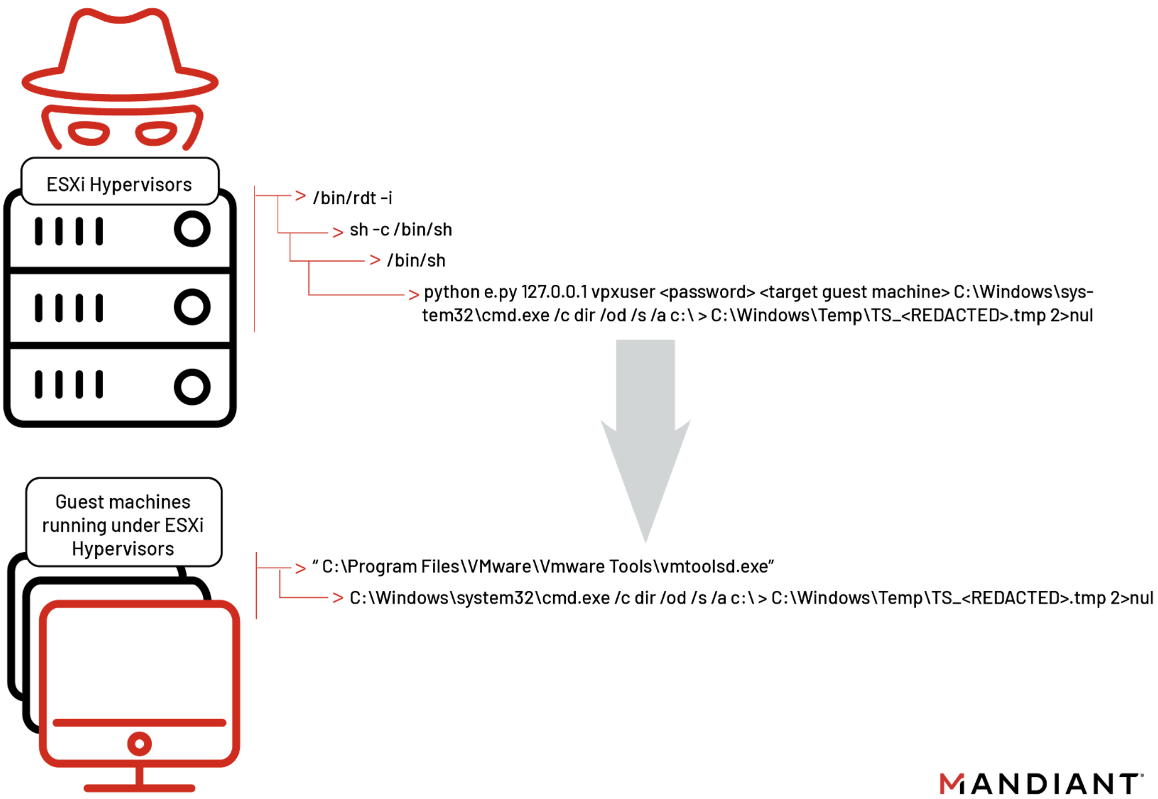

Figure 5: vmtoolsd.exe executing commands passed by ESXi

The commands the attacker ran through `vmtoolsd.exe` on the guest virtual machines were primarily focused on the enumeration and compression of files across both the system and connected file shares utilizing the native tools "dir" and "makecab". Samples of these redacted commands can be seen in Figure 6.

Figure 6: File enumeration and compression commands utilized by attacker on Guest Virtual Machines

```
C:\Windows\system32\cmd.exe" /c dir /od /s /a s:\ > C:\Windows\Temp\ts_<REDACTED>.tmp 2>null


C:\Windows\System32\cmd.exe makecab  /F C:\Windows\Temp\TS_<REDACTED>.txt  /D
compressiontype=lzx /D compressionmemory=21 /D maxdisksize=1024000000 /D
diskdirectorytemplate=C:\Windows\Temp\   /D cabinetnametemplate=TS_<REDACTED>.cab
```

Mandiant also identified the attacker targeting a virtualized system for credential harvesting. The attacker used MiniDump to dump process memory and search for cleartext credentials. Figure 7 shows an excerpt of these commands.

Figure 7: Credential Dumping on Guest Machine

```
-- " C:\Program Files\VMware\Vmware Tools\vmtoolsd.exe"
---- "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"
------ rundll32.exe C:\windows\System32\comsvcs.dll MiniDump <Process ID>
C:\Windows\Temp\TS_<REDACTED>.tmp full
```

Once the process memory was dumped, a powershell script was used to parse the resultant file for any cleartext credentials. Figure 8 shows the contents of script used for credential harvesting. The attacker also targeted KeyPass password database files.

Figure 8: PowerShell Password Search Script

```
$b = New-Object System.IO.streamReader("C:\windows\Temp\<REDACTED>.tmp",[Text.Encoding]::UTF8)

$n = 0

while (($b1 =$b.ReadLine()) -ne $null)

{

    if($b1 -like '*&password=*'){

    $n++

      Write-Host "YES $n"

      Write-Host $b1

    }

}

if($n -eq 0){Write-Host "NO!"}

$b.Dispose()
```

The attacker cleared the `C:\Windows\Temp` directory following most activity, but small errors were made which left behind trace artifacts. As shown in Figure 9, the attacker sent the output of a `dir` listing to a `.tmp` file. Since the attacker used the Linux syntax (2>null) to suppress errors instead of the Windows syntax (2>nul), all errors were forwarded to the file null in the working directory `C:\Windows\System32\null`. This file was only created if a file enumerated with the `dir` command had a directory path too long to be displayed.

Figure 9: Failed writing Error to Null in Windows Command

```
C:\Windows\system32\cmd.exe" /c dir /od /s /a s:\ > C:\Windows\Temp\TS_<REDACTED>.tmp 2>null
```

Attribution

Mandiant has begun tracking this activity as UNC3886. Given the highly targeted and evasive nature of this intrusion, we suspect UNC3886 motivation to be cyber espionage related. Additionally, we assess with low confidence that UNC3886 has a China-nexus. Each investigation conducted by Mandiant includes analysts from our Advanced Practices team who work to correlate activity observed in the thousands of investigations to which Mandiant responds. At times, we do not have the data available to directly attribute intrusion activity to a previously known group. In these cases, we create a new UNC group to track the activity that we observed. An UNC group is a cluster of related cyber intrusion activity, which includes observable artifacts such as adversary infrastructure, tools, and tradecraft, that we are not yet ready to give a classification such as APT or FIN. For more details on how Mandiant uses UNC groups, see our blog post: DebUNCing Attribution: How Mandiant Tracks Uncategorized Threat Actors.

Conclusion

While we noted the technique used by UNC3886 requires a deeper level of understanding of the ESXi operating system and VMWare's virtualization platform, we anticipate a variety of other threat actors will use the information outlined in this research to begin building out similar capabilities. Mandiant recommends organizations using

ESXi and the VMware infrastructure suite follow the hardening steps outlined in this blog post to minimize the attack surface of ESXi hosts.

MITRE ATT&CK Techniques

Collection

- T1560: Archive Collected Data
- T1560.001: Archive via Utility

Execution

- T1059: Command and Scripting Interpreter
- T1059.001: PowerShell
- T1059.003: Windows Command Shell
- T1059.004: Unix Shell
- T1059.006: Python
- T1129: Shared Modules

Command and Control

- T1105: Ingress Tool Transfer
- T1573.001: Symmetric Cryptography

Defense Evasion

- T1027: Obfuscated Files or Information
- T1070: Indicator Removal on Host
- T1070.003: Clear Command History
- T1070.004: File Deletion
- T1140: Deobfuscate/Decode Files or Information
- T1202: Indirect Command Execution
- T1218.011: Rundll32
- T1497: Virtualization/Sandbox Evasion
- T1497.001: System Checks
- T1620: Reflective Code Loading

Discovery

- T1016: System Network Configuration Discovery
- T1083: File and Directory Discovery

Lateral Movement

- T1021: Remote Services
- T1021.004: SSH

Credential Access

- T1003: OS Credential Dumping
- T1003.001: LSASS Memory

Persistence

- T1547: Boot or Logon Autostart Execution

Indicators of Compromise

Type	Value	Description
MD5	2716c60c28cf7f7568f55ac33313468b	Malicious VIB .vgz Payload
SHA1	5ffa6d539a4d7bf5aacc4d32e198cc1607d4a522	
SHA256	2be5f4520846bf493b4694789841907d058fe08d59fff6bad7abe1db8ed96e7d	
MD5	bd6e38b6ff85ab02c1a4325e8af29ce4	Malicious VIB Deployment Script
SHA1	17fb90d01403cb3d1566c91560f8f4b7dd139aa8	
SHA256	e68872c49aaedeb3bde3ff5fd2ad6f70658687dc02d04f12ebc7cb28e821cc88	
MD5	8e80b40b1298f022c7f3a96599806c43	VIRTUALPITA
SHA1	e9cbac1f64587ce1dc5b92cde9637affb3b58577	
SHA256	c2ef08af063f6d416233a4b2b2e991c177fc72d70a76c24bca9080521d41040f	
MD5	61ab3f6401d60ec36cd3ac980a8deb75	VIRTUALPIE
SHA1	93d5c4ebec2aa45dcbd6ddbaad5d80614af82f84	
SHA256	4cf3e0b60e880e6a6ba9f45187ac5454813ae8c2031966d8b264ae0d1e15e70d	
MD5	9ea86dccd5bbde47f8641b62a1eeff07	Malicious VIB Payload
SHA1	b90b19781fde2c35963eb3eac4ce2acc6f5019fb	
SHA256	23eb8d056f18e7c69ec3568f2833c9d09e91df98d11b11de235331ef42756fe5	
MD5	9d5cc1ee99ccb1ec4d20be1cee10173e	Malicious VIB Deployment Script
SHA1	9d191849d6c57bc8a052ec3dac2aa9f57c3fe0cd	
SHA256	4d995eb87b0685124b7f1640d1ab431f5a1ab991ade02750b876ed5c523234bb	
MD5	2c28ec2d541f555b2838099ca849f965	VIRTUALPITA
SHA1	e35733db8061b57b8fcdb83ab51a90d0a8ba618c	
SHA256	505eb3b90cd107cf7e2c20189889afdff813b2fbb98bbdeab65cde520893b168	
MD5	744e2a4c1da48869776827d461c2b2ec	VIRTUALPITA
SHA1	a3cc666e0764e856e65275bd4f32a56d76e51420	
SHA256	4a6f559426493abc0d056665f23457e2779abd3482434623e1f61f4cd5b41843	
MD5	93d50025b81d3dbcb2e25d15cae03428	VIRTUALPITA
SHA1	abff003edf67e77667f56bbcfc391e2175cb0f8a	
SHA256	13f11c81331bdce711139f985e6c525915a72dc5443fbbfe99c8ec1dd7ad2209	
MD5	fe34b7c071d96dac498b72a4a07cb246	VIRTUALPITA
SHA1	0962e10dc34256c6b31509a5ced498f8f6a3d6b6	
SHA256	5731d988781c9a1d2941f7333615f6292fb359f6d48498f32c29878b5bedf00f	
Descriptor	lsu-lsi-lsi-mrarpid-plugin	VIB Name

XML Name		
Filename	lsu_lsi_.v05	VIB .vgz payload
Fullpath	/etc/rc.local.d/vmware_local.sh	VIB Deployment Script
Fullpath	/bin/rdt	VIRTUALPITA
Fullpath	/bin/vmsyslog.py	
Descriptor XML Name	ata-pata-pdc20211	VIB Name
Filename	payload1.v00	VIB .vgz payload
Fullpath	/etc/rc.local.d/vmware_rhttpio.sh	VIB Deployment Script
Fullpath	/usr/lib/vmware/weasel/consoleui/rhttpproxy-io	VIRTUALPITA
Fullpath	/usr/libexec/setconf/ksmd	VIRTUALPITA (Linux)
Fullpath	/usr/bin/ksmd	VIRTUALPITA (Linux)
Fullpath	C:\Windows\Temp\avp.exe	VIRTUALGATE
Fullpath and Hash (MD5)	C:\Windows\Temp\Silverlight\wmpd.exe 76df41ee75d5077f2c5bec70747b3c99	Deleted File created by vmtoolsd.exe and executed by vmtoolsd.exe child process

Yara Detections

```
rule M_APT_VIRTUALPITA_1

{

       meta:

             author = "Mandiant"

             md5 = "fe34b7c071d96dac498b72a4a07cb246"

             description = "Finds opcodes to set a port to bind on 2233, encompassing the setsockopt(), htons(), and bind() from 40973d to 409791 in fe34b7c071d96dac498b72a4a07cb246"

      strings:

            $x = {8b ?? ?? 4? b8 04 00 00 00 [0 - 4] ba 02 00 00 00 be 01 00 00 00 [0 - 2] e8 ?? ?? ?? ?? 89 4? ?? 83 7? ?? 00 79 [0 - 50] ba 10 00 00 00 [0 - 10] e8}

      condition:

            uint32(0) == 0x464c457f and all of them

}
rule M_APT_VIRTUALPITA_2

{
```

```
       meta:

              author = "Mandiant"

              md5 = "fe34b7c071d96dac498b72a4a07cb246"

              description = "Finds opcodes to decode and parse the recieved data in the socket
buffer in fe34b7c071d96dac498b72a4a07cb246.  Opcodes from 401a36 to 401adc"

      strings:

            $x = {85 c0 74 ?? c7 05 ?? ?? ?? ?? fb ff ff ff c7 8? ?? ?? ?? ?? 00 00 00 00 e9 ??
?? ?? ?? 4? 8b 05 ?? ?? ?? ?? 4? 83 c0 01 4? 89 05 ?? ?? ?? ?? c7 4? ?? 00 00 00 00 e9 ?? ?? ??
?? 8b 4? ?? 4? 98 4? 8d 9? ?? ?? ?? ?? 4? 8d ?? e0 4? 8b 0? 4? 89 0? 4? 8b 4? ?? 4? 89 4? ?? 8b
4? ?? 4? 98 4? 8d b? ?? ?? ?? ?? b? ?? ?? ?? ?? e8 ?? ?? ?? ?? c7 4? ?? 00 00 00 00 eb ?? 8b 4?
?? 8b 4? ?? 01 c1 8b 4? ?? 03 4? ?? 4? 98 0f b6 9? ?? ?? ?? ?? 8b 4? ?? 4? 98 0f b6 8? ?? ?? ??
?? 31 c2 4? 63 c1 88 9? ?? ?? ?? ?? 83 4? ?? 01}

      condition:

            uint32(0) == 0x464c457f and all of them

}
rule M_APT_VIRTUALPITA_3

{

       meta:

              author = "Mandiant"

              md5 = "fe34b7c071d96dac498b72a4a07cb246"

              description = "Finds opcodes from 409dd8 to 409e46 in
fe34b7c071d96dac498b72a4a07cb246 to set the HISTFILE environment variable to 'F' with a
putenv() after loading each character individually."

      strings:

            $x = {4? 8b 4? ?? c6 00 48 4? 8b 4? ?? 4? 83 c0 05 c6 00 49 4? 8b 4? ?? 4? 83 c0 01
c6 00 49 4? 8b 4? ?? 4? 83 c0 06 c6 00 4c 4? 8b 4? ?? 4? 83 c0 02 c6 00 53 4? 8b 4? ?? 4? 83 c0
07 c6 00 45 4? 8b 4? ?? 4? 83 c0 03 c6 00 54 4? 8b 4? ?? 4? 83 c0 08 c6 00 3d 4? 8b 4? ?? 4? 83
c0 04 c6 00 46 4? 8b 4? ?? 4? 83 c0 09 c6 00 00 4? 8b 7? ?? e8}

      condition:

            uint32(0) == 0x464c457f and all of them

}
rule M_APT_VIRTUALPITA_4

{

       meta:

              author = "Mandiant"

              md5 = "fe34b7c071d96dac498b72a4a07cb246"

              description = "Finds opcodes from 401f1c to 401f4f in
fe34b7c071d96dac498b72a4a07cb246 to decode text with multiple XORs"

      strings:

            $x = {4? 8b 4? ?? 4? 83 c1 30 4? 8b 4? ?? 4? 8b 10 8b 4? ?? 4? 98 4? 8b 04 ?? ?? ??
?? ?? 4? 31 c2 4? 8b 4? ?? 4? 83 c0 28 4? 8b 00 4? c1 e8 10 0f b6 c0 4? 98 4? 8b 04}

      condition:
```

```
            uint32(0) == 0x464c457f and all of them

}
rule M_Hunting_Script_LaunchAndDelete_1

{

       meta:

             author = "Mandiant"

             md5 = "bd6e38b6ff85ab02c1a4325e8af29ce4"

             description = "Finds scripts that launch and then delete files, indicative of
cleaning up tracks and remaining in-memory only."

      strings:

            $ss = /setsid[^\n\r]{,250}-i[\r\n]{,5}rm/

      condition:

            all of them

}
rule M_Hunting_Python_Backdoor_CommandParser_1

{

      meta:

author = "Mandiant"

            md5 = "61ab3f6401d60ec36cd3ac980a8deb75"

            description = "Finds strings indicative of the vmsyslog.py python backdoor."

      strings:

            $key1 = "readInt8()" ascii wide

            $key2 = "upload" ascii wide

            $key3 = "download" ascii wide

            $key4 = "shell" ascii wide

            $key5 = "execute" ascii wide

            $re1 = /def\srun.{,20}command\s?=\s?self\.conn\.readInt8\(\).{,75}upload.
{,75}download.{,75}shell.{,75}execute/s

      condition:

            filesize < 200KB and all of them

}
```

Acknowledgements

Special thanks to Brad Slaybaugh, Joshua Kim, Zachary Smith, Kirstie Failey, Nick Simonian, and Charles Carmakal for their assistance with the investigation, technical review, and creating detections for the malware families discussed in this blog post. In addition, we would also like to thank VMware for their collaboration on this research.