
DFIR WITH DINOSAURS
UNEARTHING ARTIFACTS AND HOST HUNTING WITH VELOCIRAPTOR
Wes Lambert
2021
BLUE TEAM
CON

Introduction

Open source endpoint forensics, monitoring, and response

Written in Go

Linux, macOS, Windows

Lightweight, scalable, flexible, and fast!

Architecture

Server

Client

Administrative (Web) Interface

Deployment Modes

Standalone

Cloud

Triage/Offline collector

Artifact Development/Evaluation

VQL (Velociraptor Query Language)

Similar to SQL syntax

Columns | Plugin/Args | Filter Condition

```
SELECT X, Y, Z FROM plugin(arg=1)
WHERE X = 1
```

Used for:

- Collecting data from endpoints
- Performing monitoring and response
- Managing the Velociraptor server

Artifacts

One or multiple queries written in VQL

Encapsulate expert knowledge

One of the following types:

- Client Artifact
- Client Event Artifact
- Server Artifact
- Server Event Artifact

Create a new artifact

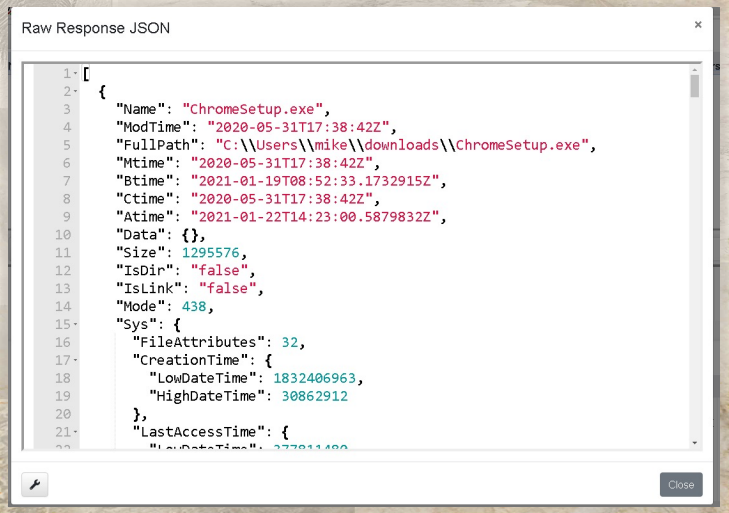
```
1  name: Custom.Artifact.Name
2  description: |
3     This is the human readable description of the artifact.
4
5  # Can be CLIENT, CLIENT_EVENT, SERVER, SERVER_EVENT
6  type: CLIENT
7
8  parameters:
9     - name: FirstParameter
10      default: Default Value of first parameter
11
12 sources:
13   - precondition:
14       SELECT OS From info() where OS = 'windows' OR OS = 'linux'
15
16     query: |
17       SELECT * FROM info()
18       LIMIT 10
19
```

Searching for Files

glob()

Search by filename

Search by file size or other properties




```
[
 {
   "Name": "ChromeSetup.exe",
   "ModTime": "2020-05-31T17:38:42Z",
   "FullPath": "C:\\Users\\mike\\downloads\\ChromeSetup.exe",
   "Mtime": "2020-05-31T17:38:42Z",
   "Btime": "2021-01-19T08:52:33.1732915Z",
   "Ctime": "2020-05-31T17:38:42Z",
   "Atime": "2021-01-22T14:23:00.5879832Z",
   "Data": {},
   "Size": 1295576,
   "IsDir": "false",
   "IsLink": "false",
   "Mode": 438,
   "Sys": {
     "FileAttributes": 32,
     "CreationTime": {
       "LowDateTime": 1832406963,
       "HighDateTime": 30862912
     },
     "LastAccessTime": {
```


https://docs.velociraptor.app/docs/forensic/filesystem/

Searching Content

yara()

Search for URLs in process memory

Search binaries for malware signatures

No need to parse files unless there is a match

```
LET YaraRule = '''
rule URL {
  strings: $a = /https?:\\/\\/[a-z0-9\\/+&#:\\?.-]+/i
  condition: any of them
}
'''

SELECT * FROM foreach(
row={
   SELECT FullPath FROM glob(globs='''C:\Users\*\AppD
}, query={
   SELECT str(str=Strings.Data) AS Hit,
          String.Offset AS Offset,
          FileName
   FROM yara(files=FullPath, rules=YaraRule)
})
```

https://docs.velociraptor.app/docs/forensic/searching/

NTFS Analysis

NTFS Accessor

$MFT – record of files on disk

$I30 – residual data in index stream slack space

USN Journal – filesystem changes

VSS copies – analyze files from earlier snapshots

https://docs.velociraptor.app/docs/forensic/ntfs/

Evidence of Execution

Prefetch – first and last execution
BAM – last execution
Shimcache – last modified
Amcache – SHA1, first execution
SRUM – application usage, CPU time, network data

https://docs.velociraptor.app/docs/forensic/evidence_of_execution/

Event Logs

Windows Event Logs/EVTX parsing

ETW – generation/publishing of events for event logs (providers/consumers)

https://docs.velociraptor.app/docs/forensic/event_logs/

Volatile State

WMI – system configuration details

Mutants – named mutex object used by malware to prevent multiple instances from running at same time

Process analysis – process creation, executable location, command line invocation

Mapped memory – identify/inspect linked DLLs

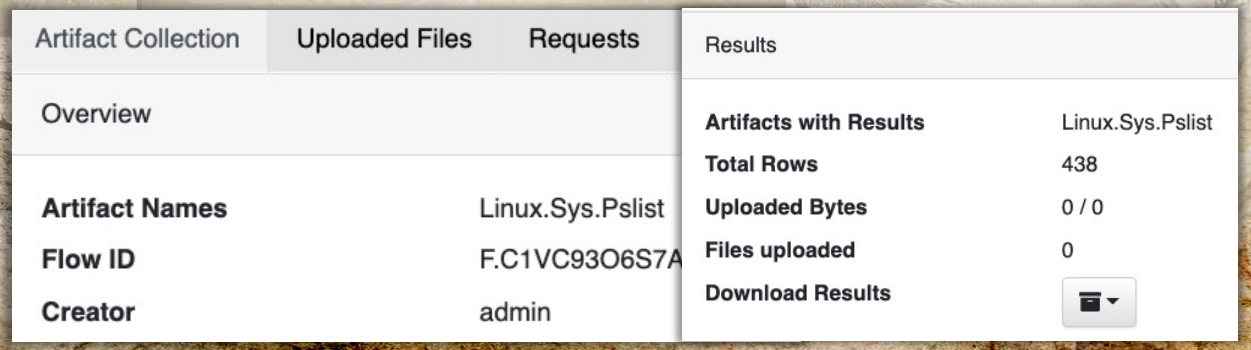
https://docs.velociraptor.app/docs/forensic/volatile/

Targeted Collection

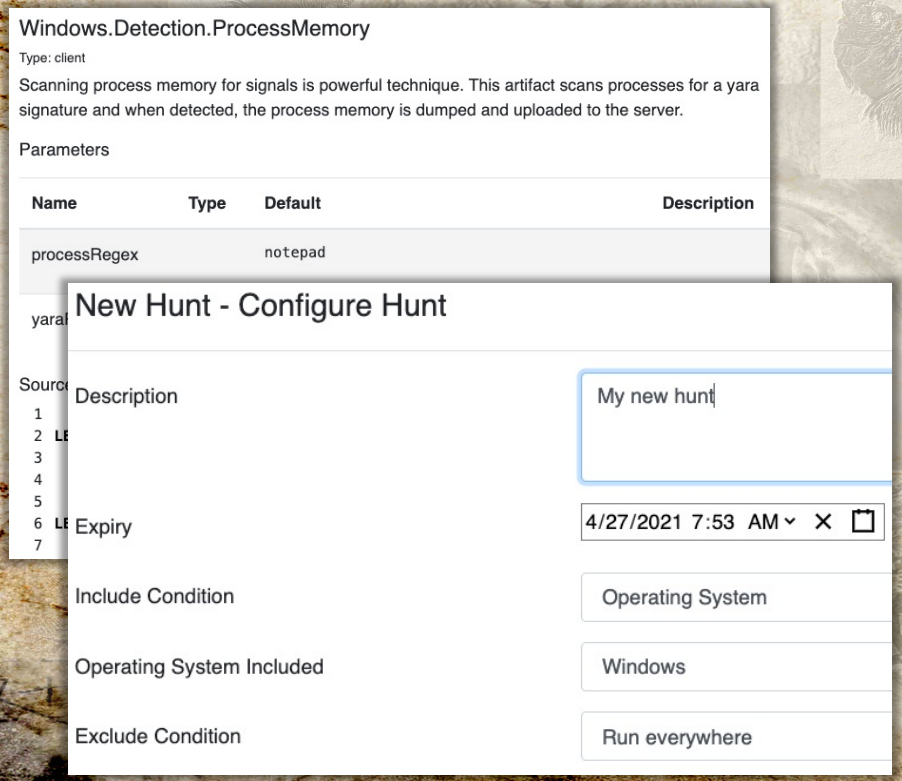
Artifacts can be selected for a single client on an ad-hoc basis

Multiple artifacts can be collected at a time

Files can be uploaded for further investigation



Hunting



Schedule a hunt across all or a subset of clients

Clients will enroll in the hunt as they come online

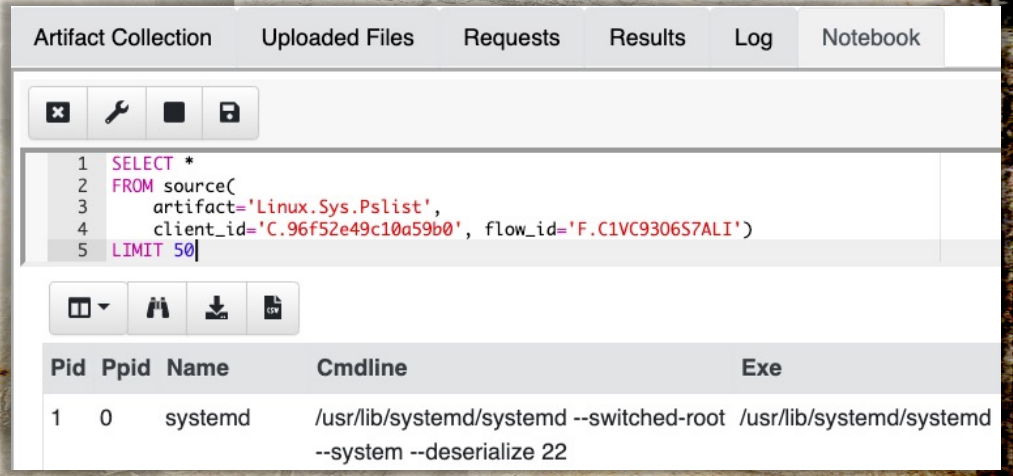
Stack results from hunts to identify outliers or commonalities

Notebooks

Similar concept to Jupyter notebooks

Test queries before creating artifacts

Post-process collection/hunt results



Detection

Client Event Artifacts

Detect when something happens on a client

Server Event Artifacts

Detect when something happens on the server

Server.Alerts.TheHive.Alert

Type: server_event

by Wes Lambert - @therealwlambert

Create a TheHive alert when monitored artifacts complete with results. Much of this was borro
https://gist.github.com/scudette/3a32abd19350c8fe3368661c4278869d

It is recommended to use the Server Metadata section to store credentials, instead of having t

Parameters

Name	Type	Default
TheHiveURL		https://mythehive
TheHiveKey		
VeloServerURL		https://myvelo
ArtifactsToAlertOn		.
DisableSSLVerify	bool	True

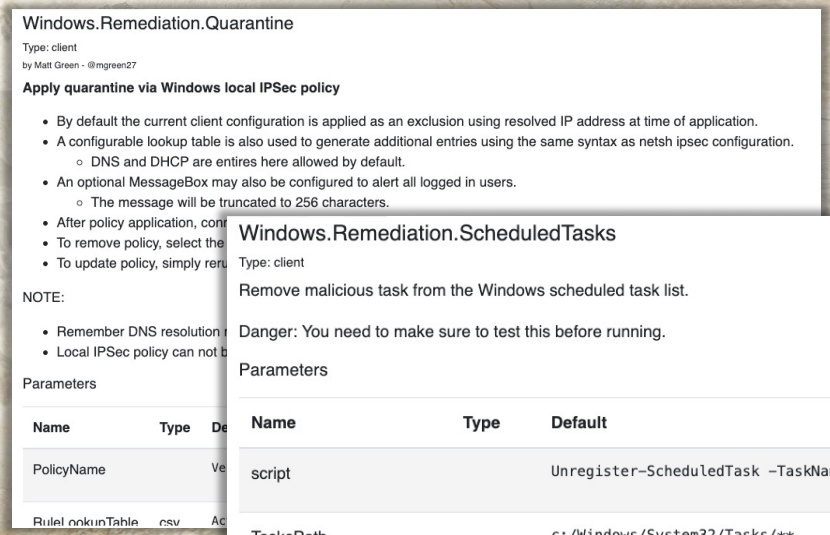
Remediation

Kill processes

Remove scheduled tasks

Isolate hosts

Run arbitrary commands



Windows.Remediation.Quarantine

Type: client

by Matt Green - @mgreen27

Apply quarantine via Windows local IPSec policy

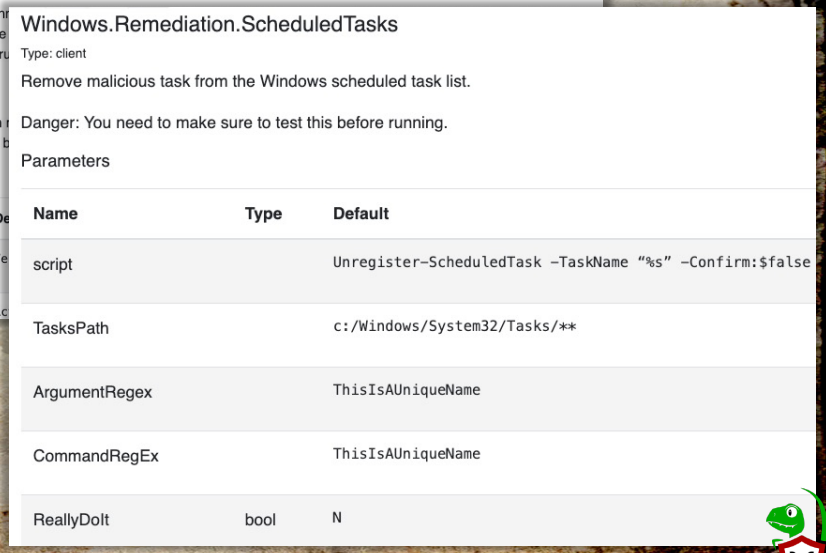
- By default the current client configuration is applied as an exclusion using resolved IP address at time of application.
- A configurable lookup table is also used to generate additional entries using the same syntax as netsh ipsec configuration.
  - DNS and DHCP are entires here allowed by default.
- An optional MessageBox may also be configured to alert all logged in users.
  - The message will be truncated to 256 characters.
- After policy application, cont
- To remove policy, select the
- To update policy, simply reru

NOTE:

- Remember DNS resolution
- Local IPSec policy can not b

Parameters

Name	Type	De
PolicyName		Ve
RuleLookupTable	csv	Ac



Windows.Remediation.ScheduledTasks

Type: client

Remove malicious task from the Windows scheduled task list.

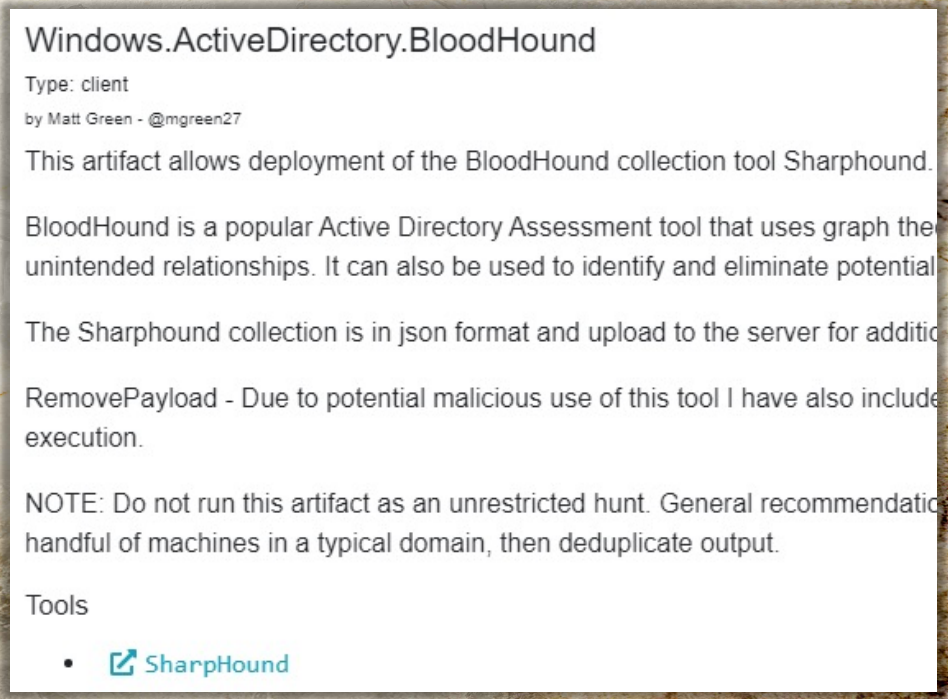
Danger: You need to make sure to test this before running.

Parameters

Name	Type	Default
script		`Unregister-ScheduledTask -TaskName "%s" -Confirm:$false`
TasksPath		`c:/Windows/System32/Tasks/**`
ArgumentRegex		`ThisIsAUniqueName`
CommandRegEx		`ThisIsAUniqueName`
ReallyDoIt	bool	N

Tools

- Velociraptor can utilize external tools
- Pull down required tools when artifact is run
- Build collector binary that includes tools



Automation

gRPC-based API, with ACL-based access

Automate response actions from Python, or a SOAR tool

- Hunt for files/hashes/processes provided from an alert from a security tool
- Isolate affected hosts

```
pip install pyvelociraptor
pyvelociraptor -c api.config.yaml “SELECT * from info()”
```

Start Digging!

Get latest binary from:

https://github.com/Velocidex/velociraptor/releases

Run:

./velociraptor gui

```
pstar@aquarius:~/velo-testing$ chmod +x velociraptor && ./velociraptor gui
[INFO] 2021-04-20T12:54:35Z
[INFO] 2021-04-20T12:54:35Z
[INFO] 2021-04-20T12:54:35Z
[INFO] 2021-04-20T12:54:35Z
[INFO] 2021-04-20T12:54:35Z
[INFO] 2021-04-20T12:54:35Z
[INFO] 2021-04-20T12:54:35Z Digging deeper!                  https://www.velocidex.com
[INFO] 2021-04-20T12:54:35Z This is Velociraptor 0.5.8 built on 2021-04-11T22:09:54Z (e468f54c)
[INFO] 2021-04-20T12:54:35Z No embedded config - you can pack one with the `config repack` command
[INFO] 2021-04-20T12:54:35Z Env var VELOCIRAPTOR_CONFIG is not set
[INFO] 2021-04-20T12:54:35Z Loading config from file /tmp/server.config.yaml
[INFO] 2021-04-20T12:54:35Z No valid config found - will generare a new one at /tmp/server.config.yaml
[INFO] 2021-04-20T12:54:35Z Starting Frontend. {"build_time":"2021-04-11T22:09:54Z","commit":"e468f54c"
[INFO] 2021-04-20T12:54:36Z Increased open file limit to 999999
[INFO] 2021-04-20T12:54:36Z Starting Journal service.
[INFO] 2021-04-20T12:54:36Z Starting the notification service.
[INFO] 2021-04-20T12:54:36Z Starting Inventory Service
[INFO] 2021-04-20T12:54:36Z Loaded 250 built in artifacts in 53.12641ms
[INFO] 2021-04-20T12:54:36Z Starting Label service.
[INFO] 2021-04-20T12:54:36Z Selected frontend configuration localhost:8000
[INFO] 2021-04-20T12:54:36Z Starting Client Monitoring Service
[INFO] 2021-04-20T12:54:36Z Reloading client monitoring tables from datastore
[INFO] 2021-04-20T12:54:36Z Creating default Client Monitoring Service
[INFO] 2021-04-20T12:54:36Z Initial user admin not present, creating
```

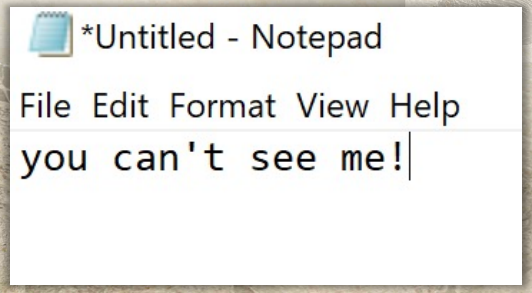
BLUE TEAM CON

Exercise 1: Scanning Process Memory

Process memory can be scanned using YARA – let's see how!

Invoke **notepad.exe**, and type the following:

"you can't see me!"



https://velociraptor.velocidex.com/digging-into-process-memory-33c60a640cdb

Exercise 1: Scanning Process Memory

Start a new notebook, and paste the following:

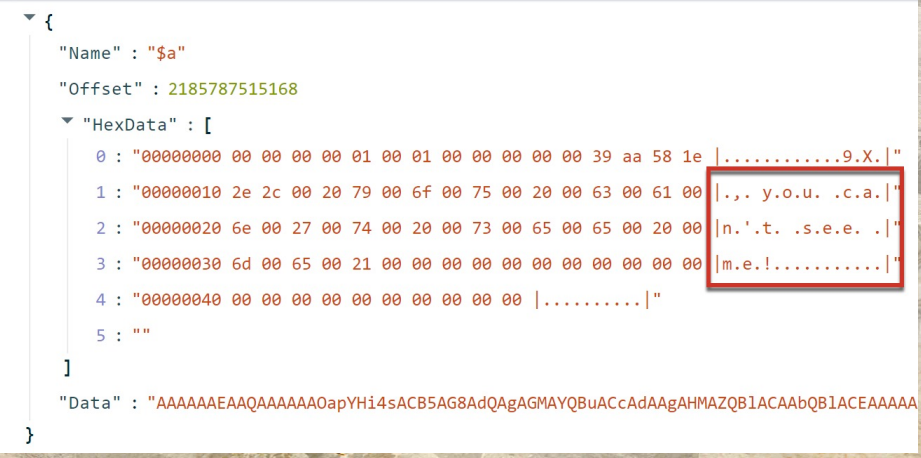
```
SELECT * FROM foreach(
row={
   SELECT * FROM pslist()
   WHERE Name =~ "notepad"
}, query={
  SELECT Pid, Name, String FROM yara(
     accessor="process",
     files=format(format="/%d", args=Pid),
     context=20,
     rules='rule x {strings: $a="you can\'t see me!" wide nocase condition: $a}'
  )
})
```

Select all processes containing “notepad” in the name

For each notepad process, use the pid to scan using the provided YARA rule

Exercise 1: Scanning Process Memory

We can see you!

Pid	Name	String
6992	notepad.exe	

While a trivial example in this case, the ability to quickly scan process memory can be quite powerful!

We could also apply this to an artifact, and hunt across our entire fleet if we wanted.

(Windows.Detection.ProcessMemory)

Exercise 2: Suspicious Service Installation

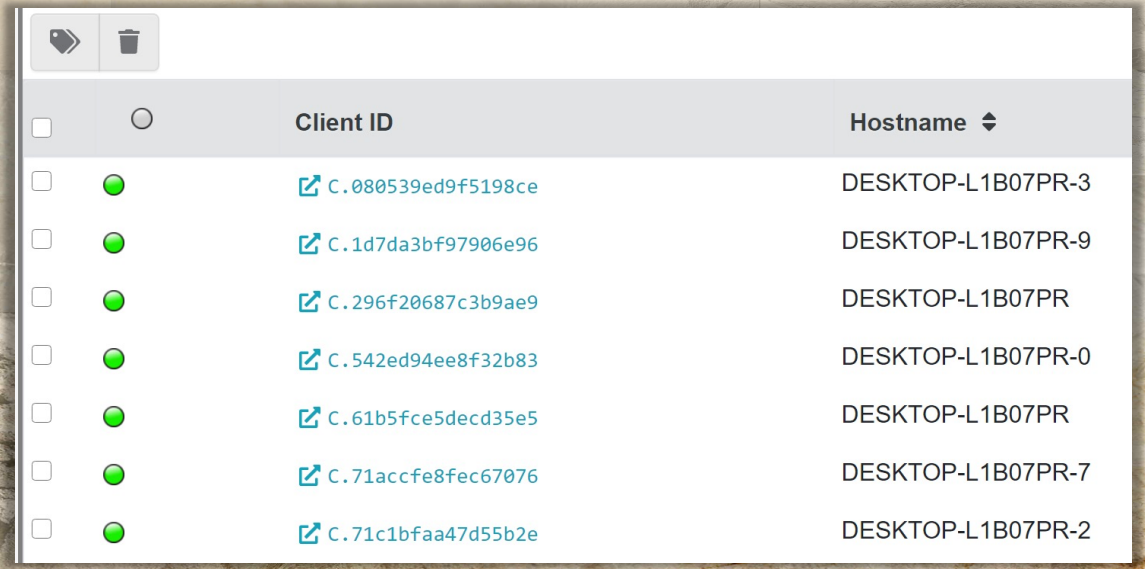
We can use Velociraptor to determine what services are installed, and stack those services across hosts to quickly and easily identity outliers or commonalities

First, we'll start a pool of clients with the following:

```
./velociraptor.exe --config filestore\client.config.yaml pool_client  --number 10 --writeback_dir filestore
```

Exercise 2: Suspicious Service Installation

From the Clients menu, we can now see multiple clients joined to the server:



Client ID	Hostname
C.080539ed9f5198ce	DESKTOP-L1B07PR-3
C.1d7da3bf97906e96	DESKTOP-L1B07PR-9
C.296f20687c3b9ae9	DESKTOP-L1B07PR
C.542ed94ee8f32b83	DESKTOP-L1B07PR-0
C.61b5fce5decd35e5	DESKTOP-L1B07PR
C.71accfe8fec67076	DESKTOP-L1B07PR-7
C.71c1bfaa47d55b2e	DESKTOP-L1B07PR-2

Exercise 2: Suspicious Service Installation

We can start a hunt with the **Windows.System.Services** artifact to get a list of installed services on the hosts, then post-process the results in a notebook with the following VQL:

```
SELECT Name, Status, PathName, UserAccount, Created, count() as COUNT
FROM source()
GROUP BY Name,
ORDER BY COUNT
```

The results show that the count for each service is the same (no outliers)

Exercise 2: Suspicious Service Installation

Next, we'll install a malicious service with the following:

```
sc.exe create backdoor binpath="c:\Windows\Notepad.exe"
```

Then, we'll start two more clients to simulate the malicious service being installed on only them:

```
./velociraptor.exe --config filestore\client.config.yaml pool_client  --number 12 --
writeback_dir filestore
```

Exercise 2: Suspicious Service Installation

The new clients will join the pre-existing hunt as they come online

After post-processing the hunt results in a notebook, through stacking we can see there are two clients that contain the “malicious” backdoor service

Windows.System.Services

2021-08-28 17:28:12 U

Name	Status	PathName	UserAccount	Created	COUNT
backdoor	OK	c:\Windows\Notepad.exe	LocalSystem	2021-08-28T17:22:07.9073883Z	2
AJRouter	OK	C:\Windows\system32\svchost.exe -k LocalServiceNetworkRestricted -p	NT AUTHORITY\LocalService	2019-12-07T09:16:04.9289101Z	13
AppIDSvc	OK	C:\Windows\system32\svchost.exe -k LocalServiceNetworkRestricted -p	NT Authority\LocalService	2019-12-07T09:16:04.9289101Z	13

Thanks for attending!

Twitter:
@therealwlambert (Wes Lambert)
@velocidex (Mike Cohen)
@mgreen27 (Matthew Green)

Github:
https://github/velocidex/velociraptor

Articles/Docs:
docs.velociraptor.app
medium.com/velociraptor-ir

Discord:
https://docs.velociraptor.app/discord