ACALVIO

An Acalvio Use Case Technical White Paper

Three Minutes Until the Apocalypse

Three Key Questions Needing Answers Within Three Minutes When You Suspect a Breach

Using Deception and Endpoint Logs to Backtrack Command and Control

Improving SOC Triage Workflow with Prevention Failure Detection

Acalvio, Inc.
2520 Mission College Boulevard
Suite 110
Santa Clara, CA 95054

+1 408.931.6160
JOHN BRADSHAW
john@acalvio.com
www.acalvio.com

Table of Contents

THREE MINUTES UNTIL THE APOCALYPSE

An adversary has targeted your organization and commenced a campaign to breach your defenses, establish a foothold and begin to either gather up your proprietary information or encrypt it and hold you hostage until you pay the ransom demand. If your present security solutions provide any warnings, they will be sent along to the front pane of glass of your Security Operations Center (SOC). Here, it is up to your team of analysts performing triage of the alerts to separate the wheat from the chaff, look at the alert details, and determine whether the alert is telling them something impactful is occurring, or decide it can be ignored.

What could possibly go wrong? Plenty.

RACE TO THE BOTTOM (OF THE ALERT PILE)

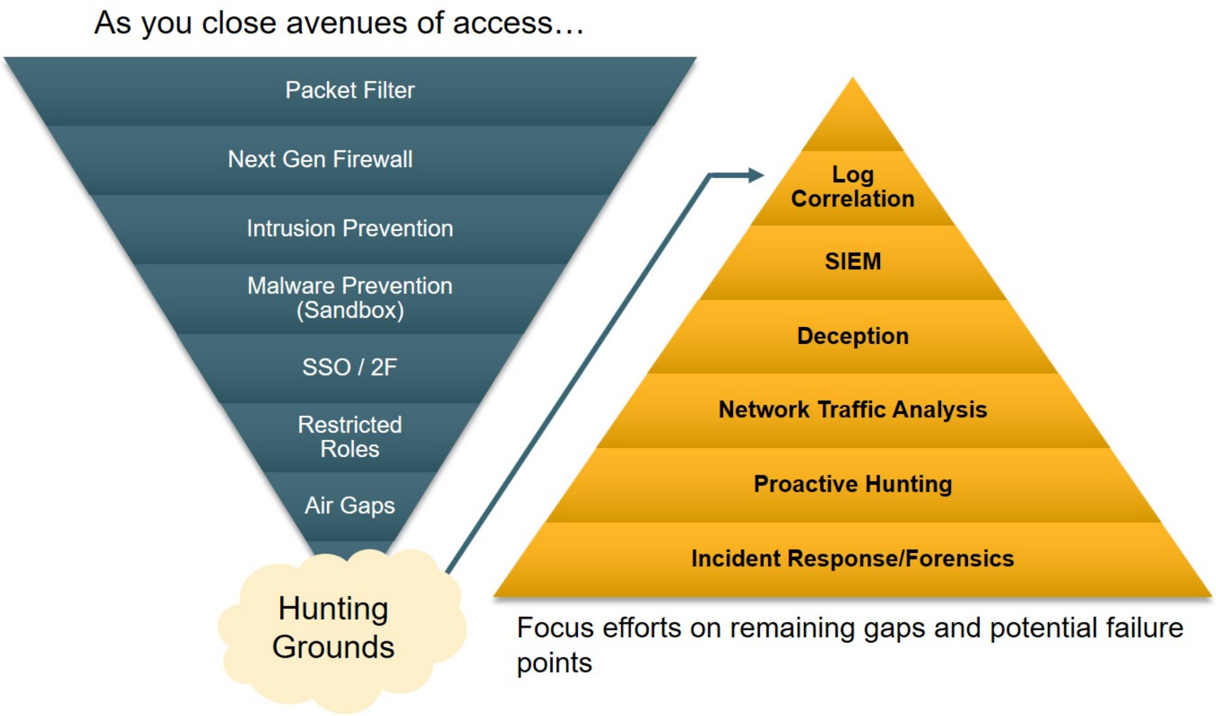
Most organizations utilize a Security Information Event Management (SIEM) solution in their SOC to aggregate, correlate and prioritize alerts presented to the frontline SOC Analyst. Initial triage of alerts is generally handled by a Level I Analyst – often the newest, and most inexperienced member of the team. With Network-based IDS often spitting out 40 events per second along with a myriad of other security solutions and operating / application logs feeding into the SIEM, it is a daunting task to keep up with the alerts on the screen. To further increase the pressure, SOC Analysts are usually expected to triage an alert in three minutes or less. Get it right, you live to triage another day; get it wrong, your stock price tumbles, people lose jobs and your company gets a ton of negative press.

Security Teams need to do three things to make their lives better:

- Refocus front-line triage on Prevention Failure Detection (PFD)
- Prioritize solutions that provide high-fidelity, but low-volume alerts during triage
- Enable correlations that answer three key questions every SOC Analyst must know when investigating alerts

PREVENTION FAILURE DETECTION

I first heard the term “Prevention Failure Detection” from a friend of mine, Tim Crothers, Vice-President of Cyber Security for Target Corporation. PFD refocuses your detection capabilities away from trying to detect ***everything*** happening in your environment and instead focus your detection efforts on where your prevention capabilities are most likely to fail. It also focuses your alerting, visualizations and SOC displays not on ***everything*** that has been seen, but only what your PFD workflows say is important. You don’t need pie charts and bar graphs telling you how many times your AV quarantined a file or your firewall blocked an access attempt – those distract your team away from ***the alerts that matter.*** You want to establish clear hunting grounds for your SOC and Incident Response (IR) teams to focus their efforts.



HIGH-FIDELITY ALERTS

If you knew an alert was a true positive every time it fired, how would that impact your workflows and decision process in handling that particular incident? High-Fidelity alerts essentially mean you can trust and act on the information contained within the alert. They also tend to be very low in volume (unless you're having a ***really*** bad day). There are not many solutions out there that can claim zero false positives (and I would be wary of any vendor that does make that claim!); however, let's consider how Deception Solutions rate when looking at Alert Fidelity and Alert Volume.

Deception and High-Fidelity

Deception-based solutions utilize decoys and misinformation to divert and delay an adversary giving the SOC / IR teams sufficient time to perform remediation before the adversary can complete his mission. Deception objects are not known to normal end-users and are white-listed against allowed vulnerability and IT Asset Discovery scanning systems in the organization – so no one should ever touch a deception decoy. Let's consider the possible ways a decoy could be touched:

- Network Misconfiguration – a scanner was missed in the whitelist or some other misconfiguration causes a system to attempt communications with a decoy
- Curious Insider – an end-user or system administrator pokes around outside of their normal duties, comes across a decoy and reaches out to see what the system is all about
- Malicious Insider – an end-user or system administrator is looking to steal information or cause disruption and stumbles across a decoy while looking for the crown jewels
- External Adversary – an adversary of varying skill level and resources has evaded your prevention layers and is now poking around inside your network

In all four cases, some type of action is required that demands immediate attention. The first two are not malicious in nature and will most likely involve different groups resolving the issue other than the security teams (most likely Network Operations for the first and Human Resources for the second). The last two are malicious and require immediate escalation and gathering of additional information to learn the full nature of the attack.

Deception and Low-Volume

Deception is a ***Breach Detection Solution.*** By that, we mean that Deception is not generally used to detect ***Intrusion Attempts*** or even ***Breach Attempts***. Deception is a great ***Prevention Failure Detection*** solution because it focuses detection capabilities on adversaries and malware that have ***already successfully bypassed your prevention capabilities.***

If we take a typical breach scenario, an adversary will spear-phish an end-user, get them to click on the malicious attachment or link, a payload gets downloaded and/or detonated on the end-user's system and command and control is established between the adversary and the compromised system.

Breach Accomplished. The Doomsday Clock starts ticking.

Many security solutions had to ***fail*** for this to happen. This first beachhead is not the mission of the adversary, they want your data or to disrupt your operations. They must establish additional beachheads, reach out to application and database servers, map out your organization's assets and determine what are likely targets. Most intrusion / breach attempts will be blocked by your prevention technologies - you aren't losing sleep over those. It is the ones that get through you need to lose sleep over - and this is where Deception solutions step in and present the adversary with inviting targets – targets that only an adversary should be touching.

Because of this, Deception alerts are few and far between (as I said earlier, unless you're having a ***really*** bad day!)

THREE KEY QUESTIONS EVERY TRIAGE ANALYST MUST ANSWER

Whenever I have seen an analyst investigating a network-based alert (alerts that only contain IP addresses, ports and service information) there are always three questions at the top of their minds – and if they had the answers would greatly speed up the time to triage as well as accuracy of any decisions made.

1. What is the endpoint user session responsible for causing this alert to occur (i.e.: Which user clicked on something they shouldn't have?)
2. What is the endpoint user session's process and parent process responsible for causing this alert to occur (i.e.: Was it a user's interactive program like Chrome or Firefox, or was it an underlying process that normally doesn't communicate like Explorer?)
3. Who else has this system communicated with in the past few minutes of this alert being generated?

Answering Question #1 lets me focus on what roles and permissions that user has so I can determine the potential extent of a breach (does this user have local admin rights? Is it a domain administrator?).

Answering Question #2 can tell me if this was a user-initiated action (John clicked a website in Chrome and downloaded a malicious payload) or an adversary-initiated action (Adversary through an injected svchost process downloaded mimikatz to the workstation). One is pre-breach/pre-detonation, the other is post-breach requiring a different level of urgency.

Answering Question #3 tells me if the adversary has laterally moved, can I identify potential command and control servers or if a malware detonation is spreading to other systems.

PREPARING FOR THE APOCALYPSE

Now that we know some key questions needing answers, we turn our attention to high-fidelity alerts that are focused around Prevention Failure Detection. We need to identify potential base and correlated events that will help us realize our vision. Once the logs, alerts and workflows have been identified, we can begin building out content that will make alert triage efficient and actionable.

Base Logs and Alerts

This paper will not address every potential log source and operating system; however, the concepts should be universally applied to your environment. For purposes of this paper, we will focus on Windows Workstations (because they are one of the most likely points of first breach) for our event source and ArcSight and Splunk (because that's what I know) for our SIEM correlation efforts (these concepts should apply to all SIEM platforms). As always, there may be more than one method to accomplish the goal, organizations should explore what methods fit best to their environments.

Event Logs

Log Source	Description	Key Fields Required
Windows Endpoint Log - Sysmon	Sysmon is a Microsoft Sysinternals tools that can be installed on Windows Workstations that provide additional logging capabilities. Sysmon will allow us to log network connections with associated process information. https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon	Network Tuple Process ID User Session
Windows Endpoint Log – Security	The Windows Security Event file contains log entries for all processes created (and terminated) on the endpoint. Process creation/termination auditing must be enabled in the security policy.	User Session Process Name Process ID Parent Process ID
High-Fidelity Network-based Alert	This can really come from any source so long as you consider it to be a high-fidelity alert that you always want correlated against the other event sources described above. For this paper we will be using Deception-based Network alerts from the Acalvio ShadowPlex solution.	Network Tuple

CORRELATING HIGH-FIDELITY ALERTS

The first step in defining solid correlation use cases is being able to define and understand the problem.

Let's review a (highly simplified) diagram of a typical breach by an adversary. The attacker sends a spear-phishing email to an end-user who opens the attachment or clicks on the link. The detonated process establishes persistence to ensure it will run if the system is rebooted or the user logs out/in. When this process runs, it will usually inject itself into a legitimate process and establish command and control (C&C) back to the adversary. From his remote system, the attacker will direct activities through the C&C channel and use tools that he downloads, or built-in system programs, to laterally move to other targets in the environment.

If I start with my high-fidelity alert and work backwards, I can establish the correlations required to provide answers to those three most sought-after questions:

Correlate:

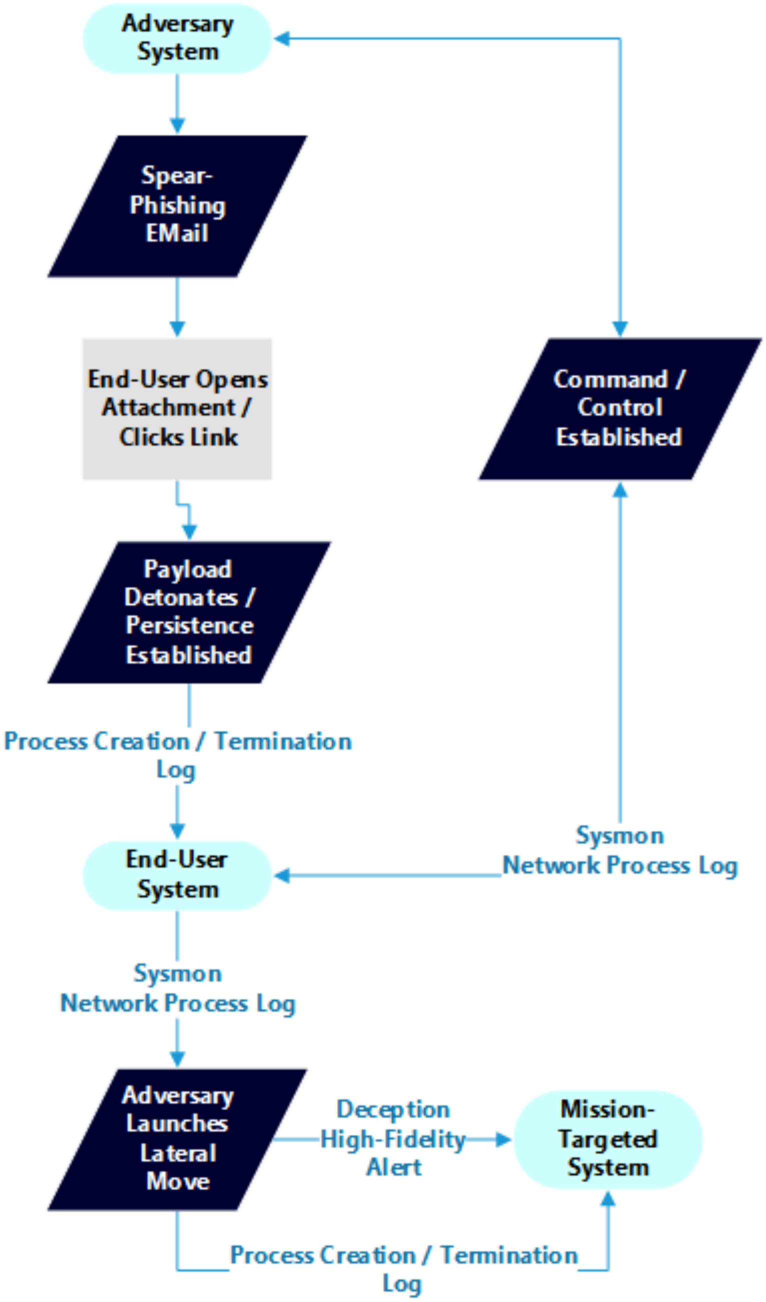
- Deception-based Alert
- Sysmon Network Process Log

Correlate:

- Sysmon Network Process (above)
- Process Creation Log

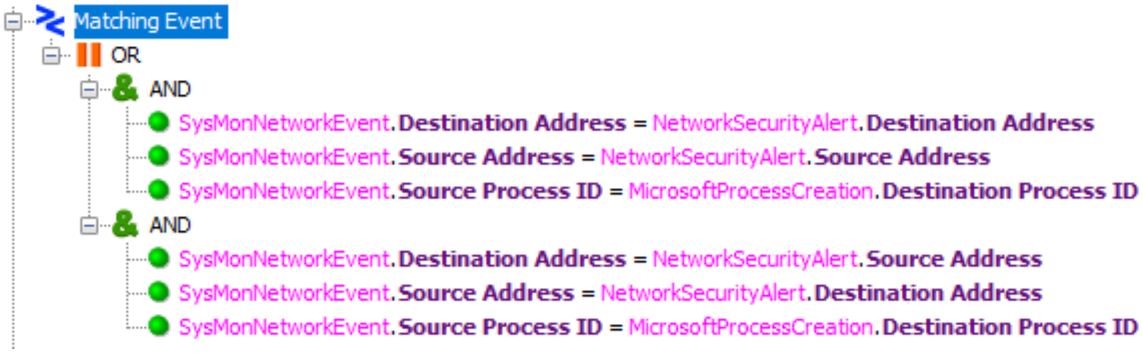
Correlate:

- Process Creation Log (above)
- Parent Process Creation Log


Adversary System
Spear-Phishing EMail
End-User Opens Attachment / Clicks Link
Command / Control Established
Payload Detonates / Persistence Established
Process Creation / Termination Log
Sysmon Network Process Log
End-User System
Sysmon Network Process Log
Adversary Launches Lateral Move
Deception High-Fidelity Alert
Mission-Targeted System
Process Creation / Termination Log

ARCSIGHT HIGH-FIDELITY BREACH ALERT CORRELATION EXAMPLE

Here is what the associated logic would look like in the ArcSight Common Condition Editor (CCE):



```
Matching Event
  OR
    AND
      SysMonNetworkEvent.Destination Address = NetworkSecurityAlert.Destination Address
      SysMonNetworkEvent.Source Address = NetworkSecurityAlert.Source Address
      SysMonNetworkEvent.Source Process ID = MicrosoftProcessCreation.Destination Process ID
    AND
      SysMonNetworkEvent.Destination Address = NetworkSecurityAlert.Source Address
      SysMonNetworkEvent.Source Address = NetworkSecurityAlert.Destination Address
      SysMonNetworkEvent.Source Process ID = MicrosoftProcessCreation.Destination Process ID
```

SPLUNK HIGH-FIDELITY BREACH ALERT CORRELATION EXAMPLE

Here is what the associated logic would look like in the Splunk Search Interface if sending the events in tagged-field format:

Deception and Matching Endpoint Events

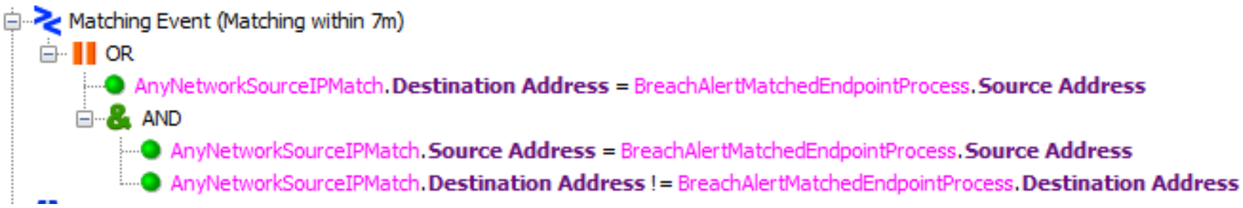
```
("NETWORK CONNECTION DETECTED" AND "MICROSOFT-WINDOWS-
SYSMON/OPERATIONAL" AND NOT "SOURCEISIPV6: TRUE") OR ("ACALVIO|SHADOWPLEX") |
REX "CEF:1\|ACALVIO\|.* SRC=(?<DECEPTIONSOURCE>\D{1,3}\.\D{1,3}\.\D{1,3}\.\D{1,3})" | REX
"CEF:1\|ACALVIO\|.* DST=(?<DECEPTIONDESTINATION>\D{1,3}\.\D{1,3}\.\D{1,3}\.\D{1,3})" | REX
"CEF:1\|MICROSOFT\|.* SRC=(?<SYSMONSOURCE>\D{1,3}\.\D{1,3}\.\D{1,3}\.\D{1,3})" | REX
"CEF:1\|MICROSOFT\|.* DST=(?<SYSMONDESTINATION>\D{1,3}\.\D{1,3}\.\D{1,3}\.\D{1,3})" | REX
"SPROC=(?<SOURCEPROCESS>\S.*?\ )[A-Z].*?\=?" |  REX
"CEF:1\|ACALVIO\|(.*?)\|(.*?)\|(.*?)\|(?<IDSALERT>.*?)\|" | EVAL
TESTSIP=COALESCE(DECEPTIONSOURCE,SYSMONSOURCE) | EVAL
TESTDIP=COALESCE(DECEPTIONDESTINATION,SYSMONDESTINATION) | TRANSACTION
TESTSIP TESTDIP MAXSPAN=5M | SEARCH "ACALVIO|SHADOWPLEX" SYSMON
EVENTCOUNT>1 | EVAL "COMPROMISED IP"=TESTSIP, "DECOY IP"=TESTDIP,
"COMPROMISED USER SESSION NAME"=SUSER, "COMPROMISED PROCESS
NAME"=SOURCEPROCESS | TABLE _TIME IDSALERT,"COMPROMISED IP", "DECOY IP",
"COMPROMISED USER SESSION NAME", "COMPROMISED PROCESS NAME"
```

RELATED COMMUNICATIONS FROM SUSPECT ENDPOINT

Now that I have a high-fidelity alert correlating against endpoint process logs, I also want to be able to gather up any other communications the compromised endpoint transmits. The goal is to help the analyst identify the attacker's C&C channel more effectively.

Matching Communications to Breach Alert

Here is an example of the correlation rule in the ArcSight CCE, we are aggregating all matching events within a seven-minute time window:



```
Matching Event (Matching within 7m)
  OR
    AnyNetworkSourceIPMatch.Destination Address = BreachAlertMatchedEndpointProcess.Source Address
    AND
      AnyNetworkSourceIPMatch.Source Address = BreachAlertMatchedEndpointProcess.Source Address
      AnyNetworkSourceIPMatch.Destination Address != BreachAlertMatchedEndpointProcess.Destination Address
```

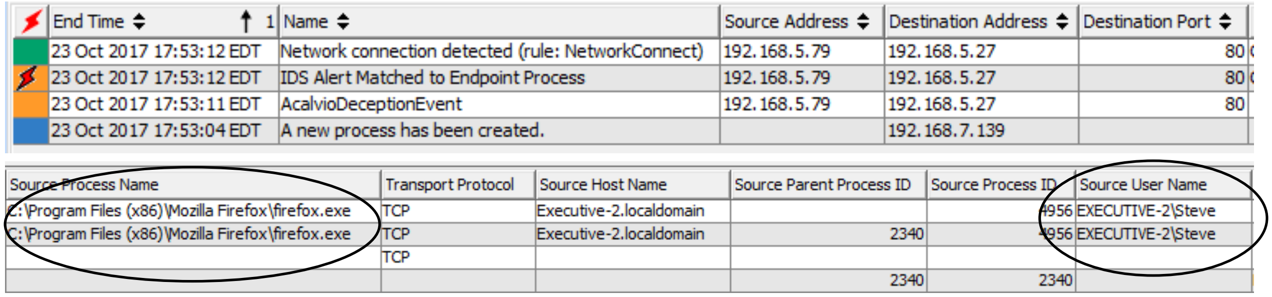
PUTTING IT ALL TOGETHER

With this correlation in place, the SOC Analyst has visibility into systems involved, program/process information, user session and the parent process ID responsible for the security alert:

QUESTION 1: WHO IS THE RESPONSIBLE USER AND PROCESS?

This correlation clearly tells me it is Steve’s user session running Firefox that reached out and touched the Deception Decoy.

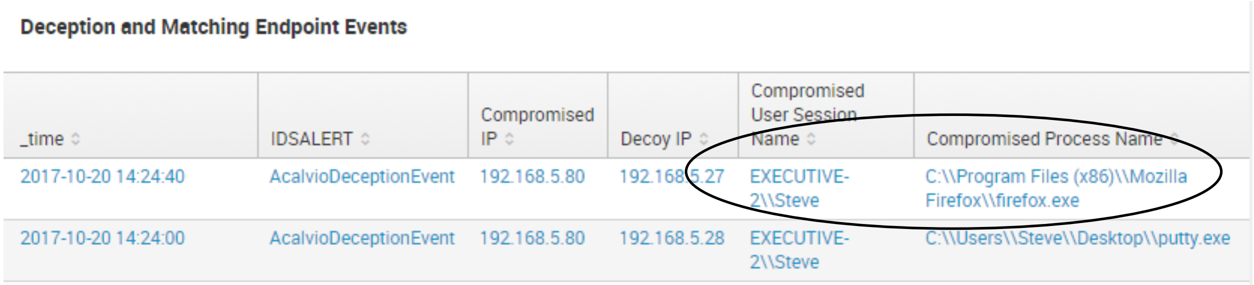
Correlation Alert in ArcSight



⚡	End Time	Name	Source Address	Destination Address	Destination Port
🟩	23 Oct 2017 17:53:12 EDT	Network connection detected (rule: NetworkConnect)	192.168.5.79	192.168.5.27	80
⚡	23 Oct 2017 17:53:12 EDT	IDS Alert Matched to Endpoint Process	192.168.5.79	192.168.5.27	80
🟧	23 Oct 2017 17:53:11 EDT	AcalvioDeceptionEvent	192.168.5.79	192.168.5.27	80
🟦	23 Oct 2017 17:53:04 EDT	A new process has been created.		192.168.7.139	

Source Process Name	Transport Protocol	Source Host Name	Source Parent Process ID	Source Process ID	Source User Name
C:\Program Files (x86)\Mozilla Firefox\firefox.exe	TCP	Executive-2.localdomain		4956	EXECUTIVE-2\Steve
C:\Program Files (x86)\Mozilla Firefox\firefox.exe	TCP	Executive-2.localdomain	2340	4956	EXECUTIVE-2\Steve
	TCP				
			2340	2340	

Correlation Alert in Splunk



Deception and Matching Endpoint Events

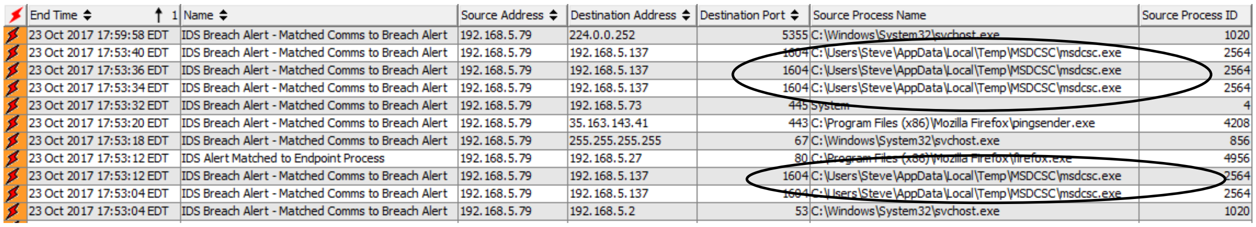
_time	IDSALERT	Compromised IP	Decoy IP	Compromised User Session Name	Compromised Process Name
2017-10-20 14:24:40	AcalvioDeceptionEvent	192.168.5.80	192.168.5.27	EXECUTIVE-2\\Steve	C:\\Program Files (x86)\\Mozilla Firefox\\firefox.exe
2017-10-20 14:24:00	AcalvioDeceptionEvent	192.168.5.80	192.168.5.28	EXECUTIVE-2\\Steve	C:\\Users\\Steve\\Desktop\\putty.exe

QUESTION 2: WHO ELSE DID THE COMPROMISED HOST RECENTLY COMMUNICATE WITH?

Correlating other communications around the time of the high-fidelity alert can yield a wealth of information. I see this workstation had approximately nine other systems it was communicating with at the time of the alert, one that stands out as highly suspicious.

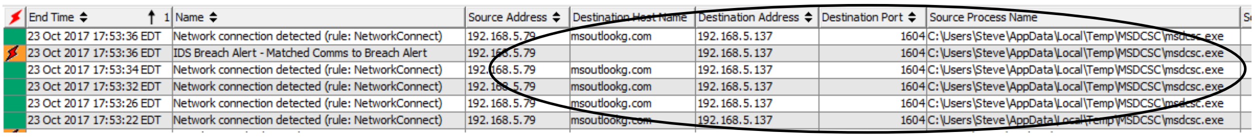
I see there is a process launched from the end-user's Local/Temp directory, while the other processes are all located in system directories. This process would get immediately flagged for investigation by the SOC Analyst.

Correlation Alert in ArcSight



⚡	End Time ⬍ ↑ 1	Name ⬍	Source Address ⬍	Destination Address ⬍	Destination Port ⬍	Source Process Name	Source Process ID
⚡	23 Oct 2017 17:59:58 EDT	IDS Breach Alert - Matched Comms to Breach Alert	192.168.5.79	224.0.0.252	5355	C:\Windows\System32\svchost.exe	1020
⚡	23 Oct 2017 17:53:40 EDT	IDS Breach Alert - Matched Comms to Breach Alert	192.168.5.79	192.168.5.137	1604	C:\Users\Steve\AppData\Local\Temp\MSDCSC\msdcsc.exe	2564
⚡	23 Oct 2017 17:53:36 EDT	IDS Breach Alert - Matched Comms to Breach Alert	192.168.5.79	192.168.5.137	1604	C:\Users\Steve\AppData\Local\Temp\MSDCSC\msdcsc.exe	2564
⚡	23 Oct 2017 17:53:34 EDT	IDS Breach Alert - Matched Comms to Breach Alert	192.168.5.79	192.168.5.137	1604	C:\Users\Steve\AppData\Local\Temp\MSDCSC\msdcsc.exe	2564
⚡	23 Oct 2017 17:53:32 EDT	IDS Breach Alert - Matched Comms to Breach Alert	192.168.5.79	192.168.5.73	445	System	4
⚡	23 Oct 2017 17:53:20 EDT	IDS Breach Alert - Matched Comms to Breach Alert	192.168.5.79	35.163.143.41	443	C:\Program Files (x86)\Mozilla Firefox\pingsender.exe	4208
⚡	23 Oct 2017 17:53:18 EDT	IDS Breach Alert - Matched Comms to Breach Alert	192.168.5.79	255.255.255.255	67	C:\Windows\System32\svchost.exe	856
⚡	23 Oct 2017 17:53:12 EDT	IDS Alert Matched to Endpoint Process	192.168.5.79	192.168.5.27	80	C:\Program Files (x86)\Mozilla Firefox\firefox.exe	4956
⚡	23 Oct 2017 17:53:12 EDT	IDS Breach Alert - Matched Comms to Breach Alert	192.168.5.79	192.168.5.137	1604	C:\Users\Steve\AppData\Local\Temp\MSDCSC\msdcsc.exe	2564
⚡	23 Oct 2017 17:53:04 EDT	IDS Breach Alert - Matched Comms to Breach Alert	192.168.5.79	192.168.5.137	1604	C:\Users\Steve\AppData\Local\Temp\MSDCSC\msdcsc.exe	2564
⚡	23 Oct 2017 17:53:04 EDT	IDS Breach Alert - Matched Comms to Breach Alert	192.168.5.79	192.168.5.2	53	C:\Windows\System32\svchost.exe	1020

If I pivot on that correlation rule to see the underlying events, I get further detail about the host involved. I highly suspect I am looking at the process and host the adversary is using for Command and Control of the compromised endpoint.



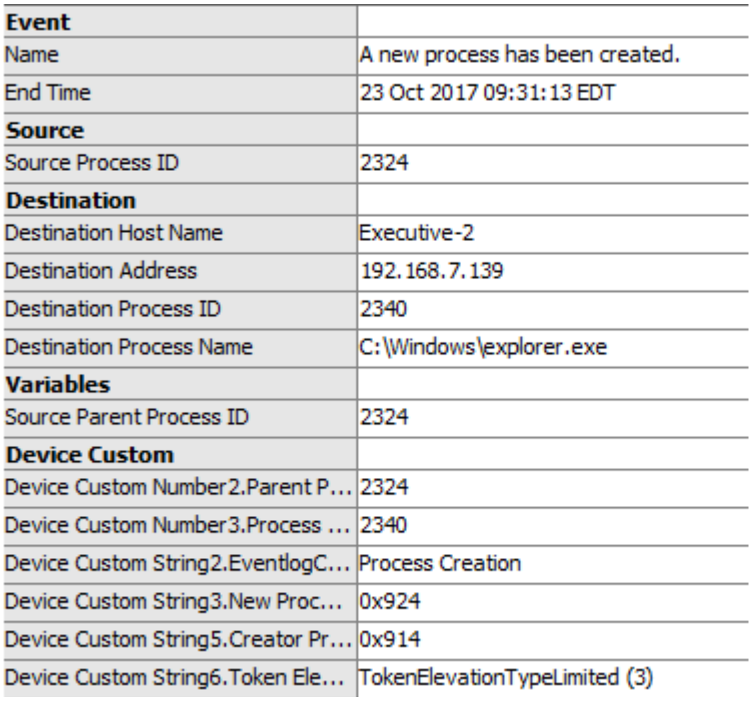
⚡	End Time ⬍ ↑ 1	Name ⬍	Source Address ⬍	Destination Host Name	Destination Address ⬍	Destination Port ⬍	Source Process Name	S
🟩	23 Oct 2017 17:53:36 EDT	Network connection detected (rule: NetworkConnect)	192.168.5.79	msoutlookg.com	192.168.5.137	1604	C:\Users\Steve\AppData\Local\Temp\MSDCSC\msdcsc.exe	
⚡	23 Oct 2017 17:53:36 EDT	IDS Breach Alert - Matched Comms to Breach Alert	192.168.5.79		192.168.5.137	1604	C:\Users\Steve\AppData\Local\Temp\MSDCSC\msdcsc.exe	
🟩	23 Oct 2017 17:53:34 EDT	Network connection detected (rule: NetworkConnect)	192.168.5.79	msoutlookg.com	192.168.5.137	1604	C:\Users\Steve\AppData\Local\Temp\MSDCSC\msdcsc.exe	
🟩	23 Oct 2017 17:53:32 EDT	Network connection detected (rule: NetworkConnect)	192.168.5.79	msoutlookg.com	192.168.5.137	1604	C:\Users\Steve\AppData\Local\Temp\MSDCSC\msdcsc.exe	
🟩	23 Oct 2017 17:53:26 EDT	Network connection detected (rule: NetworkConnect)	192.168.5.79	msoutlookg.com	192.168.5.137	1604	C:\Users\Steve\AppData\Local\Temp\MSDCSC\msdcsc.exe	
🟩	23 Oct 2017 17:53:22 EDT	Network connection detected (rule: NetworkConnect)	192.168.5.79	msoutlookg.com	192.168.5.137	1604	C:\Users\Steve\AppData\Local\Temp\MSDCSC\msdcsc.exe	

QUESTION 3: WHAT IS THE PARENT PROCESS FOR THE RESPONSIBLE COMMUNICATION?

As we see in Question 1, the Parent Process ID of the process that generated the alert is 2340. I can pivot in my logs searching for the Process Creation event for this Process ID to find out which process is most likely injected with malicious code. Bear in mind this process could have been created when the user session originally started, Active Lists in ArcSight and additional correlation techniques could help locate this faster, but are a topic for another day.

This pivot shows me that the Explorer.exe process is the parent process responsible for creating the task that reached out and touched my Deception Decoy.

Correlation Alert in ArcSight



Event	
Name	A new process has been created.
End Time	23 Oct 2017 09:31:13 EDT
Source	
Source Process ID	2324
Destination	
Destination Host Name	Executive-2
Destination Address	192.168.7.139
Destination Process ID	2340
Destination Process Name	C:\Windows\explorer.exe
Variables	
Source Parent Process ID	2324
Device Custom	
Device Custom Number2.Parent P...	2324
Device Custom Number3.Process ...	2340
Device Custom String2.EventlogC...	Process Creation
Device Custom String3.New Proc...	0x924
Device Custom String5.Creator Pr...	0x914
Device Custom String6.Token Ele...	TokenElevationTypeLimited (3)

WHAT REALLY HAPPENED

So, did the correlated events present an actual representation of how the attack really happened? Let's find out.

To create the attack, I used the DarkComet Remote Access Trojan (RAT) to take command and control of my "victim" system. I attached the RAT in an email and sent it to my target user (Steve).

When Steve opened the attachment, the RAT detonated and established a C&C link back to the control system (msoutlookg). The DarkComet program planted itself in Steve's Local/Temp directory using the name MSDCSC.EXE. Through DarkComet's Remote Desktop capability, I launched FireFox from the user's desktop (Explorer.exe) and reached out to an HTTP Deception Decoy – generating the high-fidelity alert.

As you can see, all these activities were correlated and captured providing clear context to the analyst as to what happened. And it all started from a single high-fidelity alert.

CONCLUSION

It is not enough for organizations to keep pumping all types of security events into SIEMs and hoping they get correlated and prioritized appropriately for the Level I Analyst. The triage process needs to focus on Prevention Failure Detection utilizing high-fidelity alerts combined with use case focused correlations that answer the key questions accurately and efficiently. Knowing the user session involved in the breach, processes responsible for communications, and other network communications involving a breached system are critical to rapidly isolating and remediating the compromise.

Utilizing Deception-based alerts with endpoint logs, SIEM can deliver on its capability to correlate alerts that matter.

The Apocalypse has been averted.

ABOUT ACALVIO

Acalvio provides Advanced Threat Defense (ATD) solutions to detect, engage and respond to malicious activity inside the perimeter. The solutions are anchored on patented innovations in Deception and Data Science. This enables a DevOps approach to ATD, enabling ease of deployment, monitoring and management. Acalvio enriches its threat intelligence by data obtained from internal and partner ecosystems, enabling customers to benefit from defense in depth, reduce false positives, and derive actionable intelligence for remediation.

www.acalvio.com

2520 Mission College Boulevard, Suite 110,

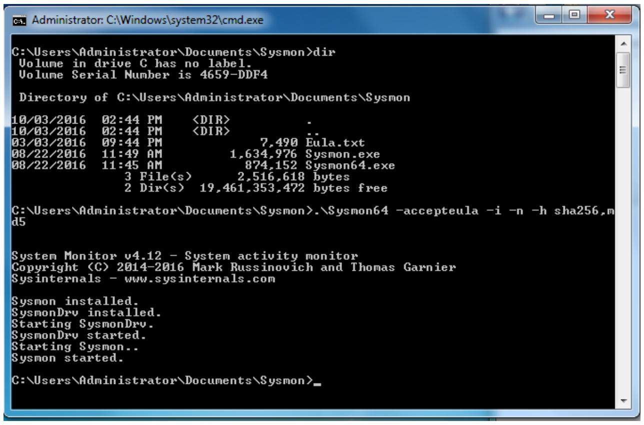
Santa Clara, CA 95054

USA: 1-408-931-6160

India: 91-80-4176-4500

APPENDIX A: WINDOWS ENDPOINT CONFIGURATION

Download and install the Sysmon utility using the following configuration parameters ***sysmon64 –accepteula –i –n –h sha256,md5***. Depending on your requirements, you may choose to select different file hash options, or none at all; however, the "-n" option is the most important for this use case as it logs network connections made on the endpoint. For more information and to download this tool, please visit: https://technet.microsoft.com/en-us/sysinternals/sysmon

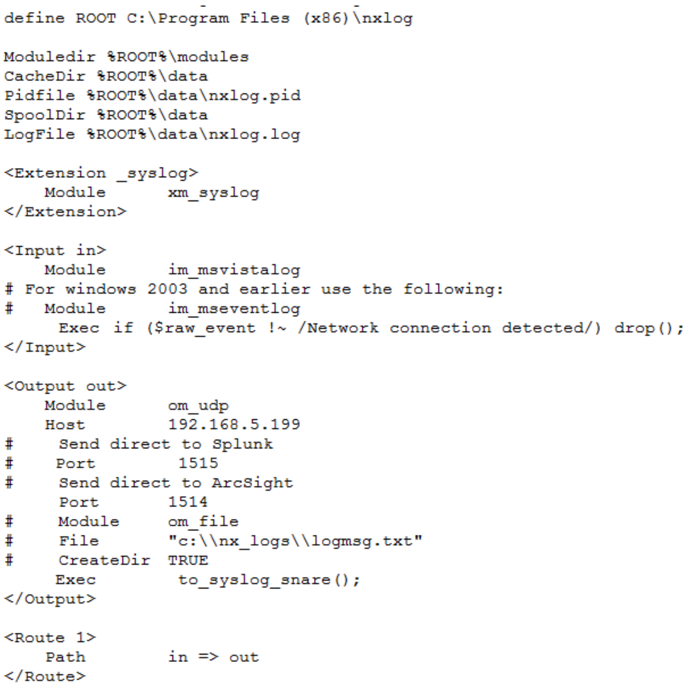


```
C:\Users\Administrator\Documents\Sysmon>dir
 Volume in drive C has no label.
 Volume Serial Number is 4659-DDF4

 Directory of C:\Users\Administrator\Documents\Sysmon

10/03/2016  02:44 PM    <DIR>          .
10/03/2016  02:44 PM    <DIR>          ..
03/03/2016  09:44 PM             7,490 Eula.txt
08/22/2016  11:49 AM         1,634,976 Sysmon.exe
08/22/2016  11:45 AM           874,152 Sysmon64.exe
               3 File(s)      2,516,618 bytes
               2 Dir(s)  19,461,353,472 bytes free

C:\Users\Administrator\Documents\Sysmon>.\Sysmon64 -accepteula -i -n -h sha256,m
d5


System Monitor v4.12 - System activity monitor
Copyright (C) 2014-2016 Mark Russinovich and Thomas Garnier
Sysinternals - www.sysinternals.com

Sysmon installed.
SysmonDrv installed.
Starting SysmonDrv.
SysmonDrv started.
Starting Sysmon..
Sysmon started.

C:\Users\Administrator\Documents\Sysmon>_
```

Next, setup your syslog endpoint configuration. I used NXLog (https://nxlog.co/) for this example. Use the following configuration file as an example to configure NXLog to read the Sysmon Operational logs and only send the Network Connection Detected events to your Syslog receiver.

This configuration file instructs NXLog to read from the MS Event Log using the EventLog API for Windows 2008/Vista and later, and send matching event logs in SNARE format over UDP Port 1514 to the system at 192.168.5.199.

NOTE: You may have reasons to send other Sysmon events, this Use Case paper is focused only on the Network Connection events.



```
define ROOT C:\Program Files (x86)\nxlog

Moduledir %ROOT%\modules
CacheDir %ROOT%\data
Pidfile %ROOT%\data\nxlog.pid
SpoolDir %ROOT%\data
LogFile %ROOT%\data\nxlog.log

<Extension _syslog>
    Module      xm_syslog
</Extension>

<Input in>
    Module      im_msvistalog
# For windows 2003 and earlier use the following:
#   Module      im_mseventlog
     Exec if ($raw_event !~ /Network connection detected/) drop();
</Input>

<Output out>
    Module      om_udp
    Host        192.168.5.199
#     Send direct to Splunk
#     Port       1515
#     Send direct to ArcSight
      Port      1514
#     Module    om_file
#     File      "c:\\nx_logs\\logmsg.txt"
#     CreateDir TRUE
     Exec        to_syslog_snare();
</Output>

<Route 1>
    Path        in => out
</Route>
```

APPENDIX B: ARCSIGHT SYSLOG SUB-PARSER

Current limitations in the ArcSight Microsoft Windows Unified Connector prevent it from reading the Sysmon Event Log on endpoints, therefore I also use NXLog to generate the necessary event steam when using ArcSight solutions. To ensure the Sysmon Network Connection events are properly parsed (they will not be if you use the default Intersect Alliance SNARE parser) I create a Syslog Regex FlexConnector sub-parser to normalize the events. Direct your NXLog output to a Syslog SmartConnector and add the following sub-parser to your SmartConnector configuration.

SDK Filename: microsoft.suibagent.sdkrfilereader.properties

Ensure “microsoft_syslog” is in front of snare_syslog for the agents[0].customsubagentlist parameter in the agent.properties file. I generally remove any subagent parsers that I am not using to keep things clean.

```
# FlexAgent Regex Configuration File
do.unparsed.events=true

regex=(\\S+)\\t(\\d+)\\t(\\S+)\\t(\\d+)\\t(.*)\\t(\\d+)\\t(\\S+)\\t(\\S+\\t\\S+\\t\\S+
\\t\\S+\\t)(Network connection detected \\(rule\:
NetworkConnect\\))\\t\\t(.*)\\\:\\s\\sUtcTime\\\:\\s(\\d+\\-\\d+\\-
\\d+\\s\\d+\\\:\\d+\\\:\\d+\\.\\d+)\\s\\sProcessGuid\\\:\\s(\\S+)\\s\\sProcessId\\\:\\
s(\\d+)\\s\\sImage\\\:\\s(.*)\\s\\sUser\\\:\\s(.*)\\s\\sProtocol\\\:\\s(\\S+)\\s\\sIni
tiated\\\:\\s(\\S+)\\s\\sSourceIsIpv6\\\:\\s(\\S+)\\s\\sSourceIp\\\:\\s(\\S+)\\s\\sSou
rceHostname\\\:\\s(.*)\\s\\sSourcePort\\\:\\s(\\d+)\\s\\sSourcePortName\\\:\\s(.*)\\s\
\sDestinationIsIpv6\\\:\\s(\\S+)\\s\\sDestinationIp\\\:\\s(\\S+)\\s\\sDestinationHostn
ame\\\:\\s(.*)\\s\\sDestinationPort\\\:\\s(\\d+)\\s\\sDestinationPortName\\\:\\s(.*)\\
t(\\d+)

token.count=28
token[0].name=mseventLog
token[0].type=String
token[1].name=syslogHeaderNumber
token[1].type=String
token[2].name=mseventLogType
token[2].type=String
token[3].name=syslogEventNumber
token[3].type=String
token[4].name=mseventTimestamp
token[4].type=String
token[5].name=mseventID
token[5].type=String
token[6].name=mseventLogType2
token[6].type=String
token[7].name=mseventUnneededData
token[7].type=String
token[8].name=mseventName
token[8].type=String
token[9].name=mseventName2
token[9].type=String
token[10].name=mseventUTCTime
token[10].type=String
token[11].name=mseventProcessGUID
token[11].type=String
token[12].name=mseventProcessID
```

```
token[12].type=Integer
token[13].name=mseventProcessName
token[13].type=String
token[14].name=mseventUserName
token[14].type=String
token[15].name=mseventProtocol
token[15].type=String
token[16].name=mseventInitiated
token[16].type=String
token[17].name=mseventSourceIPv6
token[17].type=String
token[18].name=mseventSourceIP
token[18].type=String
token[19].name=mseventSourceHostName
token[19].type=String
token[20].name=mseventSourcePort
token[20].type=Integer
token[21].name=mseventSourcePortName
token[21].type=String
token[22].name=mseventDestinationIPv6
token[22].type=String
token[23].name=mseventDestinationIP
token[23].type=String
token[24].name=mseventDestinationHostName
token[24].type=String
token[25].name=mseventDestinationPort
token[25].type=Integer
token[26].name=mseventDestinationPortName
token[26].type=String
token[27].name=mseventJunkField
token[27].type=String

event.deviceVendor=__stringConstant("Microsoft")
event.deviceProduct=__stringConstant("MS Sysmon")
event.deviceCustomIPv6Address2=__stringToIPv6Address(__ifTrueThenElse(__contains(mseve
ntDestinationIP,"\:"),mseventDestinationIP,""))
event.sourceHostName=mseventSourceHostName
event.sourceUserName=mseventUserName
event.sourceProcessName=mseventProcessName
event.name=mseventName
event.sourceProcessId=mseventProcessID
event.destinationServiceName=mseventDestinationPortName
event.sourceAddress=__oneOfAddress(__ifTrueThenElse(__contains(mseventSourceIP,"."),ms
eventSourceIP,""))
event.externalId=mseventID
event.transportProtocol=mseventProtocol
event.destinationPort=mseventDestinationPort
event.deviceCustomIPv6Address1=__stringToIPv6Address(__ifTrueThenElse(__contains(mseve
ntSourceIP,"\:"),mseventSourceIP,""))
event.sourcePort=mseventSourcePort
event.sourceServiceName=mseventSourcePortName
event.deviceEventClassId=mseventLogType
event.destinationAddress=__oneOfAddress(__ifTrueThenElse(__contains(mseventDestination
IP,"."),mseventDestinationIP,""))
event.destinationHostName=mseventDestinationHostName
```