Coding For Incident Response: Solving the Language Dilemma

Incident responders frequently are faced with the reality of "doing more with less" due to budget or manpower deficits. The ability to write scripts from scratch or modify the code of others to solve a problem or find data in a data "haystack" are necessary skills in a responder's personal toolkit. The question for IR practitioners is what language should they learn that will be the most useful in their work? In this paper, we will examine several coding languages used in writing tools and scr...

CODING FOR INCIDENT RESPONSE:

Solving the Language Dilemma

GIAC (GSEC) Gold Certification

Author: Shelly Giesbrecht, headnerd@nerdiosity.com
Advisor: Mark Stingley

Accepted: July 22nd, 2015

Abstract

Incident responders frequently are faced with the reality of "doing more with less" due to budget or manpower deficits. The ability to write scripts from scratch or modify the code of others to solve a problem or find data in a data "haystack" are necessary skills in a responder's personal toolkit. The question for IR practitioners is what language should they learn that will be the most useful in their work? In this paper, we will examine several coding languages used in writing tools and scripts used for incident response including Perl, Python, C#, PowerShell and Go. In addition, we will discuss why one language may be more helpful than another depending on the use-case, and look at examples of code for each language.

1. Introduction

Lack of financial and staff resources are a chronic issue for incident responders. On any given day, in any given organization, members of computer security incident response teams ("CSIRT"s) can find themselves figuratively hip-deep in data. C-level types expect answers to urgent questions like: "What happened?", "What was taken?", "How did the attacker get in?" The ability to deliver these answers in a timely and accurate manner is directly related to the amount of resources available to a CSIRT. A 2014 survey of incident responders by the SANS Institute (Torres, 2014) suggests that many organizations are unable to staff a dedicated CSIRT due to budgetary issues and consequently rely on resources from other internal teams to bolster an investigation when, and if, an incident is detected. In the same survey, 39% of respondents stated they were unaware of any funds being allocated to Incident Response ("IR"), and 30% stated there were no funds allocated. This may mean that CSIRT teams or individual responders are without enterprise-grade technology to assist in their investigations, and without training dollars to advance their skill sets to offset tool deficiencies.

With cyber incidents on the rise, the central question is how can incident responders "do more with less"; that is how can they improve their ability to prevent, detect, and respond with small budgets and teams? In this paper, we will examine the benefits of learning to code for incident responders. In addition, we will look at coding languages frequently used in IR, the benefits and drawbacks of each language, and why one language might be used over another for a specific task. Lastly, we will review an example of a script or tool written in each language.

2. Coding for Incident Response

2.1. Why learn to code?

The obvious answer to why an incident responder would want to learn to code is to be able to code. While this sounds like oversimplifying the answer, prolific Digital Forensics and Incident Response ("DFIR") blogger David Cowen (2013) suggests that

Shelly Giesbrecht, headnerd@nerdiosity.com

learning to code will allow growth as a responder, increase efficiency, and reduce frustration. For most IR professionals, the journey to coding proficiency starts with a need or a problem that needs a solution. A common issue that learning to code may help to resolve is a budgetary one. Many smaller organizations simply do not have the budget to purchase commercial tools that would ease the burden of their IR staff, if they even have dedicated IR staff.

Another type of problem maybe a technical one, perhaps a need to automate a repetitive task, chain together a set of scripts or utilities, or find an IP address in a mountain of firewall log files. The ability to code enables a responder to work smarter, and either create or modify existing scripts or tools to meet their own requirements.

Lastly, with the flood of new artifacts being documented from DFIR research and investigations, it is almost certain that scripts or tools to extract, parse and validate those artifacts do not exist in either the commercial market or open-source community, or an existing tool may no longer adequately accomplish the job at hand.

Whatever the reason for choosing to pick up coding skills, the next question to face is what language to choose.

2.2. What language is the right choice?

Just like spoken languages, computer-programming languages each have their own idiosyncrasies, benefits, limitations, cadence, and syntax. Choosing the right language to learn can make the difference between success and failure, just like learning Spanish for a trip to Japan.

Toal (n.d.) describes categories of programming languages as follows:

- Machine languages — interpreted directly in hardware
- Assembly languages — thin wrappers over a corresponding machine language
- High-level languages — anything machine-independent
- System languages — designed for writing low-level tasks, like memory and process management
- Scripting languages — generally extremely high-level and powerful
- Domain-specific languages — used in highly special-purpose areas only
- Visual languages — non-text based

- Esoteric languages — not really intended to be used

A general understanding of these categories and their features is inherent to selecting the appropriate language to learn. A language may exist in more than one category; however, ability to use a feature of category does not make a language a good choice (Suh, 2002). More important to the decision of what language to use is the end goal. Carvey (2014) suggests that it is this goal, and the mode of getting to it that matters most. In other words, choose a language based on features that will best aid the path to successful completion of the defined objective. In the following sections, we will examine several languages that are popularly used in coding for incident response and discuss how each language may be well suited to the needs of a particular goal.

3. Coding Languages

3.1. Perl

3.1.1. History of Perl

Perl or the Practical Extraction and Report Language made its debut in 1987 on the 'comp.sources' UseNet newsgroup as a text processing language for Unix or Unix-like operating systems (Kuhn, 2001). The creator of Perl, Larry Wall wanted to bridge the gap between two concepts he referred to as "manipulexity" and "whipupitude".

Manipulexity refers to the "manipulation of complex things" by high-level languages like C (Wall, 2006). Most of the major operating systems used today including UNIX, Linux, and Windows are written in C or C-based languages. This is because C is highly structured, can be compiled on different platforms and the programs created are efficient ("C Language", n.d.). In contrast, *whipupitude* or the 'aptitude for whipping things up' describes the ease with which programmers could use utilities like AWK to create handy scripts and programs (Kuhn, 2001).

3.1.2. Why use Perl?

What would make you crazy enough to use Perl? We believe that many beginning programmers ask themselves that question when first attempting to learn to code in Perl. Over the course of its development, Perl became the language of choice for many coders and an object of fear for many others. The reasons for these opinions lie mainly in one key feature of Perl, which some consider an advantage and others, a limitation.

Within the syntax of Perl, it is possible to accomplish the same thing in a variety of ways. This is called the 'TIMTOWTDI' philosophy, or 'there is more than one way to do it' (Newkirk, 2014). For experienced programmers, the ability to write a string of code using syntax they like the most or know the best and simply enclosing the result in some braces is much preferable to being constrained to the more structured approach some of the later languages discussed take. On the other hand, it is suggested that beginners in coding will acquire bad habits when left to their own devices in the more laissez-faire style of Perl (Mikoluk, 2013).

Aside from the aforementioned TIMTOWTDI philosophy, Perl is widely considered low on the usability scale. In their study on the usability of Perl, Python and Tcl, Pfeiffer and Wang (2002) found that Perl suffered from a number of difficulties including challenging language constructs that made Perl harder to learn and use over time.

A clear advantage Perl has held over its competition for quite some time is the large library of modules written over the years and collected in the CPAN (Comprehensive Perl Archive Network) repository (Banerjee, 2012). The modules housed in CPAN allow a rank beginner or a seasoned developer to re-use code that others have contributed in their own work to solve virtually any problem programmatically.

Regular Expressions or 'RegEx' is another reason Perl continues to hold its popularity. RegEx gives coders the ability to define an expression that can describe one or more strings (Kuhn, 2001). In RegEx, special characters are used to assist in these definitions. For example, a pipe symbol, '|', is used to separate alternate characters or strings. The following simple expression indicates either "A" or "B":

```
A|B
```

This ability to describe strings while harnessing the string manipulation power of other tools like AWK and SED make Perl RegEx a dominant tool in data processing for incident responders and sysadmins alike.

In the realm of benefits, one more very important one comes to mind for Perl. Incident responders are frequently called upon to work on a variety of operating systems.

While Perl comes natively installed on most Unix/Linux distributions, it is also easily ported to Windows and Mac OS systems (Newkirk, 2014). This ubiquity allows responders the possibility of writing one script or tool that can be run across multiple platforms. Carvey (2007) suggests that one common theme across the practice of incident response is that there is no such thing as one type of incident or investigation. Perl's portability makes it a natural ally for a seasoned responder.

3.1.3. Perl in Action

There is no better tool to demonstrate the power of Perl than RegRipper by Harlan Carvey. RegRipper is an offline Windows registry analysis tool that automates the retrieval, parsing and translation of registry data, and the last-written timestamp for that data, into human readable output files (Carvey, 2010). As many responders do, we count RegRipper among the first line of tools used in an investigation. Statistics available on the project homepage for RegRipper on Google Code (2015) indicate the current version 2.8 has been downloaded 18817 times as of June 18, 2015. Registry hives can be analyzed using the rr.pl script on the command line or via the GUI using the rr.exe executable.

```
1  #! c:\perl\bin\perl.exe
2  #-----------------------------------------------------------
3  # Registry Ripper
4  # Parse a Registry hive file for data pertinent to an investigation
5  #
```

Figure 1. RegRipper Perl script 'rr.pl'

The power of RegRipper comes from the plug-ins, which are individual Perl scripts that each performs a different parsing operation (Carvey, 2010). A plethora of plug-ins are already available, but as the RegRipper project is totally open source, creating a new plug-in to fill a void or extract a newly discovered artifact from the registry is an option for any responder willing to contribute.

The USBSTOR registry key found at HKLM\SYSTEM\CurrentControlSet\Enum\USBSTOR\ and an entry is created or updated every time a USB device is connected (Barbara, 2012). Using RegRipper and the plug-ins written for USBSTOR, it is trivial to extract all devices that have been connected to a computer system and the last-written timestamp.

```
7790  ----------------------------------------
7791  usbstor v.20080418
7792  (System) Get USBStor key info
7793
7794  USBStor
7795  ControlSet001\Enum\USBStor
7796
7797  Disk&Ven_&Prod_&Rev_8.07 [Thu Jun 11 01:26:47 2015]
7798    S/N: 1405058600198840 [Thu Jun 11 01:26:47 2015]
7799      FriendlyName : USB Device
7800
7801  Disk&Ven_FLASH&Prod_Drive_SK_USB20&Rev_1.00 [Mon Sep 15 00:09:28 2014]
7802    S/N: 89900000AA04012700008C97&0 [Mon Sep 15 00:09:28 2014]
7803      FriendlyName : FLASH Drive SK_USB20 USB Device
7804
7805  Disk&Ven_Staples&Prod_Relay_UFD&Rev_1.26 [Fri Jun 19 15:39:29 2015]
7806    S/N: 2004432033078C0315F0&0 [Fri Jun 19 15:39:29 2015]
7807      FriendlyName : Staples Relay UFD USB Device
7808
7809  Disk&Ven_WD&Prod_My_Passport_0830&Rev_1065 [Fri May 29 20:40:37 2015]
7810    S/N: 575850314537345A324A5A41&0 [Fri May 29 20:40:37 2015]
7811      FriendlyName : WD My Passport 0830 USB Device
7812
7813  Other&Ven_WD&Prod_SES_Device&Rev_1065 [Fri May 29 20:40:37 2015]
7814    S/N: 575850314537345A324A5A41&1 [Fri May 29 20:41:09 2015]
7815
```

Figure 2. RegRipper output: USBSTOR entries

In an investigation, this information enables a responder to identify whether and when a particular device was connected to a system, and possibly connect the owner of the device to the investigation. All this is accomplished in mere seconds with the power of Perl.

3.2. Python

3.2.1. History of Python

Python has recently seen an upswing in its popularity that may lead new coders to believe it is a newly released language. While it is newer than Perl, Python was actually developed as a scripting language by Guido Van Rossum in 1989 and publically released in 1991 (Barbara, 2003). The development of Python diverged in 2008 with the release of 3.0, a new version that was wholly incompatible with the 2.x hitherto released (PSF, 2008). Similar to the schism between the coders of Perl and Python regarding superiority of their preferred language, a similar schism has developed between devotees of Python 2.x and 3.x. releases. Both versions have current releases as of 2015, with the 2.x version standing at 2.7.10 and the 3.x version at 3.4.3 (PSF, 2015).

3.2.2. Why use Python?

The popularity Python is currently enjoying is in large part to do with its high level of usability. The ease with which the constructs of Python can be learned and used

in comparison to Perl contributes to this usability (Pfeiffer & Wang, 2002). Similar to Perl, Python is considered a scripting language, but in contrast to the ability of Perl programmers to vary their syntax to achieve the same objective, Python implements very rigid conventions that must be adhered to for a successful outcome (Mikoluk, 2013). This benefits new learners of programming, as they are able to easily follow the multitude of online tutorials and use the same syntax and constructs.

A major complaint about Python is the speed of execution compared to other languages. This is said to be due, in large part, to Python being dynamically typed (type checking is done at runtime) as opposed to statically typed (type-checking is done during compilation) (Vanderplas, 2014). What this means is that a Python script or application must interpret its code while it executes. This requirement adds overhead to the overall processing and renders Python up to 1.5 times slower than other languages that are interpreted when compiled (Zaïane, 2007).

One feature of Python that is highly praised or highly decried depending on what camp of coders it originates from is Python's use of whitespace to mark blocks of code instead of, for example, curly braces. This is considered an advantage by some as it underlines the rule of having only one way to write Python (Levin, 2011). A script written in Python will simply not run if the indents are not lined up properly. The required attention to detail ensures that coders new to Python are unable to ignore the importance of proper syntax and subsequently, may learn quicker as they are not confused by varied methods to achieve the same goal. On the other hand, the idiosyncratic conventions Python employs means that translating code written in Python to other languages is not trivial.

Another outcome of the popularity of Python is the proliferation of modules being developed for Python in the same vein as Perl. This is evident particularly in the incident response and digital forensics field where Python modules are being showcased in academic papers and blogs. In his paper "Grow Your Own Forensics Tools: A Taxonomy of Python Libraries Helpful for Forensic Analysts", O'Connor (2010) discusses and gives examples of several Python libraries that could be used to build tools and scripts to assist forensic analysis. More recently, David Cowen (2015) has written a

series of blog posts using Python modules to extract, examine, and analyze a forensic image. The community-driven approach to building up these modules is similar to Perl and the number of libraries and modules available will soon rival CPAN.

3.2.3. Python in Action

There is perhaps no other Python project currently being used by incident responders that receives more attention than Volatility. Developed as an open source alternative for memory and volatile data analysis, Volatility was first released at BlackHat in 2007 and is said to be the 'most widely used memory forensics platform' (Volatility Foundation, 2013). As more and more cyber-attacks are using methods that never touch the physical file system, an advanced capability to analyze dynamic sources of data is imperative. Volatility provides this capability with a multitude of profiles for different operating systems and versions, as well as an entire library of plug-ins to examine volatile images for artifacts.

A good place to start for any incident responder when a system is suspected of being compromised is rogue processes. Volatility provides four main plug-ins for examining a system's running processes. First, pslist will list the processes of system and includes that start and exit time of a process, but does not display any hidden or unlinked processes (Hale-Ligh et al, 2014a). This can be useful when looking for process start times that do not coincide with system start-up time.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 pslist
```

Figure 3. Command to run pslist for a Windows 7 x64 system (Volatility Foundation, 2015).

Secondly, pstree shows the same list but in a tree format so that the parent-child relationships are visible. Similar to pslist, hidden and unlinked processes are not displayed (Hale-Ligh et al, 2014b). Pstree is very helpful to find a process that may be considered normal in pslist but is not linked to its expected parent process, and therefore, could be considered suspicious.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 pstree
```

Figure 4. Command to run pstree for a Windows 7 x64 system (Volatility Foundation, 2015b).

Next, psscan lists all processes seen by pslist and pstree, but can also uncover processes that are already terminated, or altered by a rootkit and hidden or unlinked (Hale-Ligh et al, 2014c). Any processes that appear in psscan that did not appear in pslist or pstree may be considered suspicious and require further investigation.

```
$ python vol.py --profile=Win7SP0x86 -f win7.dmp psscan
```

Figure 5. Command to run psscan for a Windows 7 x86 system (Volatility Foundation, 2015c).

Lastly, if the process is well and truly hidden, Volatility provides one more plug-in, psxview, that provides a comprehensive view of all currently available locations where processes could be listed. Each possible location for each process is marked "True" when evidence of the process is found, and "False" when a process is found to be missing for that location (Volatility Foundation, 2014). As below, the process 1_doc_RCData_61 would be considered suspicious as it was found in evidence in every location but where pslist displays (Hale-Ligh et al, 2014d).

```
$ python vol.py -f prolaco.vmem psxview
Volatility Foundation Volatility Framework 2.4
Offset(P)  Name                    PID pslist psscan thrdproc pspcid csrss session deskthrd
---------- -------------------- ------ ------ ------ -------- ------ ----- ------- --------
0x06499b80 svchost.exe            1148 True   True   True     True   True  True    True
0x0640ac10 msiexec.exe            1144 False  True   False    False  False False   False
0x005f23a0 rundll32.exe           1260 False  True   False    False  False False   False
0x0113f648 1_doc_RCData_61        1336 False  True   True     True   True  True    True
```

Figure 6. Command and output for psxview (Volatility Foundation, 2015d.)

The portability of Python allows Volatility to be use for IR triage easily and quickly on most major platforms. In addition, the smaller learning curve for Python as opposed to other coding languages allows many more people to contribute (and to feel they can contribute) to the Volatility plug-in library.

3.3. C#

3.3.1. History of C#

While "cool" is not what may initially come to mind when thinking about C# ("C-sharp"), the name C# was originally given was 'C-like Object Oriented Language' or 'Cool' (Hamilton, 2008). Currently in its fifth version, C# was originally released in 2001 by Microsoft with the intent of creating applications in .NET that were platform and runtime agnostic (Dillon, 2002). This means that programs written in C# could be run on virtually any architecture, although porting to Apple iOS or Android requires a utility called Xamarin, and for Linux, a framework called Mono (O'Brien, 2014). C# is considered to have stemmed from the C-based language family, although comparisons to Java are frequently made.

3.3.2. Why use C#?

C# seems an unlikely choice for incident response given that it is not considered a scripting language like Perl, Python, or PowerShell. However, there are some features of C# that could be considered advantages over the others discussed here depending on the objective of a responder. Unlike the coding languages described thus far, C# is a statically typed language which means it is interpreted when it is compiled (O'Brien, 2014). This means the overhead incurred when a script or application is interpreted at run-time like Python does not occur with C#, and therefore, it should perform a similar function faster.

In addition to the lower overhead at runtime, complied C# code meets the Common Type Specifications ("CTS") (Introduction to the C#, n.d.). This means that it can be integrated easily with the approximately twenty other .NET Framework languages that also meet CTS. This ability to integrate was extended even further with the release of C# version 4 and the introduction of dynamic keywords (Ochal, 2010). This new feature meant that statically typed C# was also able to interact with dynamic languages like Perl or Python.

The drawback to allowing dynamic-type functionality into C# code was that it limited or removed the advantages gained from being statically typed; namely, speed of execution (Allen, n.d.). We can assume that while there may be reasons when to use

dynamic keywords for C# coding integration, the benefits must outweigh the consequences.

3.3.3. C# in Action

When investigating a case of employee malfeasance where an individual may be accused of accessing or taking files they were not authorized to, a frequent denial heard by incident responders is "I have never even opened that directory". A key artifact is establishing whether this access was made, in addition to when, is assisted by Windows Registry keys found in the NTUSER.dat and UsrClass.dat hives called Shellbags (McQuaid, 2014). Shellbags are found in the following locations:

Windows XP:

- NTUSER.DAT\Software\Microsoft\Windows\Shell
- NTUSER.DAT\Software\Microsoft\Windows\ShellNoRoam
- NTUSER.DAT\Software\Microsoft\Windows\StreamMR

Windows 7:

- NTUSER.DAT\Software\Microsoft\Windows\Shell
- UsrClass.dat\Local Settings\Software\Microsoft\Windows\Shell

(Pullega, 2013)

Within the registry, shellbags are not human readable and therefore require a utility to extract and parse the data to make it usable for investigation. To achieve this,

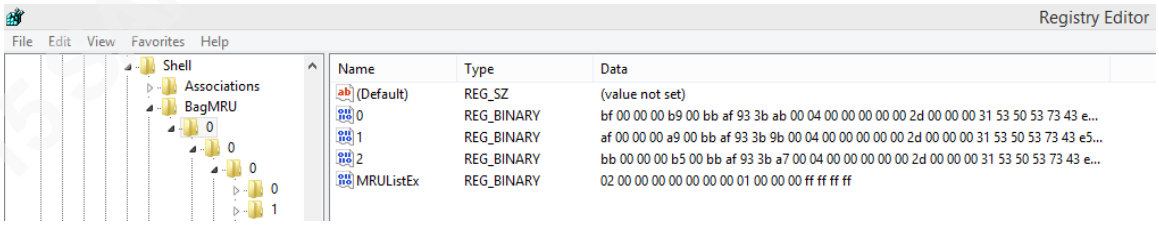

Figure 7. Shellbags registry key and values in NTUSER.dat in Windows 7

Eric Zimmerman created ShellBags Explorer (SBE). SBE was released in 2014 and provides an investigator with the means to view parsed data from the shellbags keys as if they were looking at the user in question's directory structure (Zimmerman, 2015).

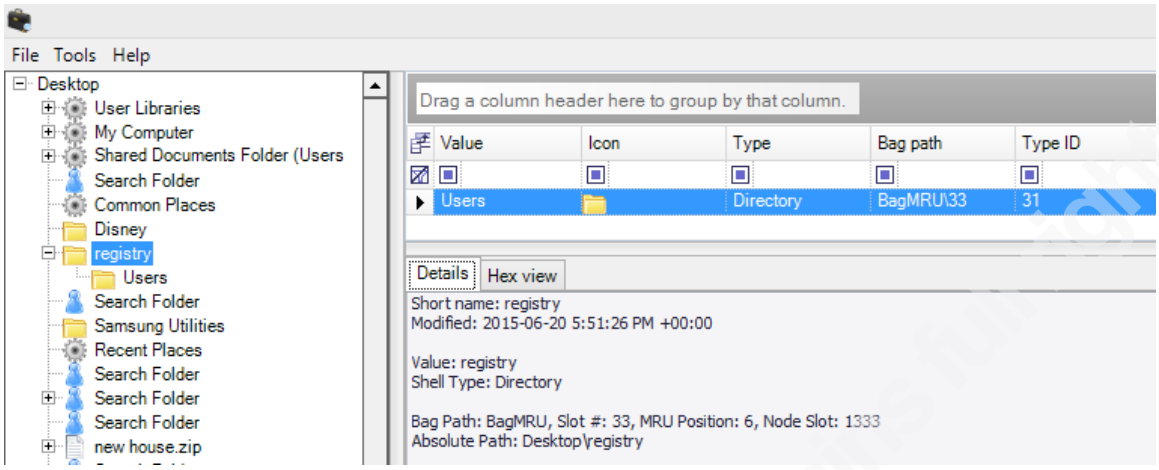

Figure 8. ShellBags Explorer output on live Windows 7 registry

The output from SBE displays the timestamp the entry was created (i.e.: first accessed by the user), as well as last modified and last accessed dates.

According to Zimmerman (2015a), a main driver for using C# in the development of SBE was the ability to create an attractive gui in a non-web-based medium. The result is a Windows Explorer-like interface that an experienced Windows user can navigate with ease.

3.4. PowerShell

3.4.1. History of PowerShell

When Microsoft wanted to update the commandline shell, PowerShell was the result. Originally named 'Monad', PowerShell was intended to replicate the ease and flexibility of shells available on UNIX and Linux systems, as well as provide a powerful scripting language that integrated seamlessly with the .NET Framework (Lee, 2005). Currently in version 5.0, PowerShell has become the de-facto standard for shell and scripting in the Windows environment.

3.4.2. Why use PowerShell?

The ubiquity of PowerShell in modern Windows operating systems is its most obvious strength. PowerShell is installed by default on Windows 7/Windows Server 2008R2 forward (MSDN, 2008). A language written by Microsoft for Windows suggests a virtual wealth of possibilities for incident responders to interact with target systems. In

absence of availability of other IR tools and scripts, PowerShell provides a responder with a native tool already in existence.

Now that we have PowerShell though, what can we actually do with it? The answer lies in what might be called the build-blocks of PowerShell. The name 'Monad' came from a philosophy called 'Monadism' that suggested that small bits of matter, of which a monad is the smallest, combine to create the world (Lee, 2005). PowerShell is comprised of a library of small commands called 'cmdlets' that enable the user to manage all facets of Windows systems (TechNet, 2014). Cmdlets can be written in any .NET compliant language including C# and are named in a very user-friendly noun-verb format (Lee, 2005). For example, a cmdlet in frequent use is 'GET-COMMAND'. When invoked in a PowerShell command shell, this cmdlet will display all available cmdlets for the version installed. These cmdlets can be used individually or in combination to allow an incident responder to interrogate any Windows system past Windows XP SP2.

Figure 9. PowerShell Get-Command output

A criticism of using PowerShell for incident response or digital forensics was made by Andrew Case on Twitter. Case (2014) suggested that since PowerShell scripts are still running on a live system, the API calls made could still be hooked by malicious processes. The implication is that the results would be invalid or skewed. The decision to use PowerShell for IR purposes may then depend on what is the objective of the responder (ie: Does the objective involve malware?) or possibly, the scope of the examination (ie: Could PowerShell be used across a large volume of systems to collect information that would be used to determine whether a deeper dive was required?).

Lastly, the biggest drawback of PowerShell is it is limited to Windows systems. While this isn't a problem when a organization is wholly a Windows 'shop', throwing in

a mix of other platforms means that incident responders will require a broader skillset that just PowerShell.

3.4.3. PowerShell in Action

The ability to be deploy tools and scripts quickly and efficiently in IR is imperative in order not to miss critical data or indicator of compromise ("IOC"). What if you had to perform IR activities across literally thousands of Windows systems on a daily basis? What if your network was under constant attack by external attackers as well as an internal red team? These are just some of the challenges that face the Incident Response team at Microsoft.

Kansa is a modular PowerShell-based IR framework written by Dave Hull that collects and analyzes data from Windows systems across an enterprise using PowerShell Remoting capabilities (Hull, 2014). Hull, as it happens, is an incident responder at Microsoft.

Figure 10. Running kansa.ps1

Data collection in Kansa is accomplished by running the main PowerShell script 'kansa.ps1' and specifying the modules and list of target hosts. Once the data is collected, analysis on the data is available. The caveat to the Analysis modules is that they were written to parse through data for multiple systems, from dozens to many thousands (Hull, 2015a). This means that analysis on one machine is not possible as many of the points of analysis rely on comparison. For example, the Kansa module 'Get-Autorunsc.ps1' leverages the SysInternals Autorunsc.exe to collect data on Auto Start Extension Points (ASEP) including ASEP hashes (Hull, 2015b). A step in the analysis compares the ASEP hashes from one system to others collected. The figure below shows that for ten domain controllers where data was collected, all ten returned the same seven ASEP hashes (Hull, 2014). In this case, a responder may consider this a non-finding, or

cnt	Image Path	MD5
10	c:\windows\system32\cpqnimgt\cpqnimgt.exe	78af816051e512844aa98f23fa9e9ab5
10	c:\hp\hpsmh\data\cgi-bin\vcagent\vcagent.exe	54879ccbd9bd262f20b58f79cf539b3f
10	c:\windows\system32\cpqmgmt\cqmgstor\cqmgstor.exe	60668a25cfa2f1882bee8cf2ecc1b897
10	c:\program files\hpwbem\storage\service\hpwmistor.exe	202274cb14edaee27862c6ebce3128d8
10	c:\hp\hpsmh\bin\smhstart.exe	5c74c7c4dc9f78255cae78cd9bf7da63
10	c:\msnipak\win2012sp0\asr\configureasr.vbs	197a28adb0b404fed01e9b67568a8b5e
10	c:\program files\hp\cissesrv\cissesrv.exe	bf68a382c43a5721eef03ff45faece4a

Figure 11. ASEP hashes collected from 10 domain controllers (Hull, 2014).

choose to investigate further. It is important to note that the only thing that is certain from the hashes returned is that they are identical to each other on all ten systems. There is not yet any indication of their legitimacy.

Taking into account the limitation of PowerShell as an IR language previously mentioned, we can assume that once data has been collected and analyzed, the usefulness of the data will depends on the problem being faced. With that said, we can also reasonably assume that the output from the Kansa analysis will be able to identify, with some degree of confidence, systems that display IOCs that require further investigation.

3.5. GO

3.5.1. History of Go

In terms of the languages discussed in this paper, Go can be considered the new kid on the block. Starting as a concept in 2007, Go was publically released in November 2009 (The Go Programming, n.d.). The impetus behind the development of Go was a language that would provide faster programming and compiling, better overall performance at run-time and an enjoyable user experience (Turner, 2010). Varghese (2014) suggests that what makes Go so unique is its focus on real-world, current applications instead of following the established practices of its predecessors, C# and Java. The intent then of Go is more in line with C#, as a language for building applications, as opposed to more scripting focused languages like Perl, Python or PowerShell.

3.5.2. Why use Go?

For the coding traditionalists, Go has many of the features its predecessors have including statically typed variables and garbage collection (Varghese, 2014). While this provides better runtime performance and memory management, Go may still seem to be another unlikely choice for the world of IR where agility is a key factor. Additionally, it has been suggested that library support in Go is still quite limited (Jenkins, 2014). This might be expected from a language that is not as well established as other we have previously reviewed. However, a number of key differentiators separate Go from more traditional C-based languages.

First, the syntax of Go is compact and considered relatively easy to learn (Forbes, 2013). This means that it is becoming more and more popular among newer coders looking for a more modern and succinct way to learn to code. In addition, Go was

developed with concurrency in mind. Concurrency describes the ability of Go to execute multiple processes simultaneously and is a core component of cloud-based application development (Asay, 2014). As multi-core or multi-processor architecture becomes the norm, the ability to take advantage of that processing power is a key consideration for developers. Go uses a combination of *goroutines* and *channels* to provide its own brand of concurrency. Goroutines use channels to move values and to communiciate with more security and effiency (Varghese, 2014).

Perhaps the most intriguing feature of Go is that despite its ties to statically typed languages for performance, it is suggested that the experience of coding in Go is similar to many dynamic languages (McAllister, 2012). The idea that a coding language could not only run quickly but also allow creativity and be fun to work with appears to be quite a novel concept. One of Go's core developers, Rob Pike (2012) maintains that while he originally thought C++ programmers would obviously move to Go, a more "expressive" language that retained many of C++'s features, in reality, they observed far more Python and similar language coders adopt Go as a means of acquiring better performance and concurrency while retaining creativity. It is this hybrid and flexible nature of Go that may see it become a more dominant choice for IR professionals.

3.5.3. Go in Action

In the section on PowerShell, we discussed Kansa, a Windows-centric tool for scanning systems across a large organization for signs of compromise. While extremely powerful, what if the organization in question has a very large contingent of Linux/Unix systems with some Mac OS thrown in for good measure?

Enter the Mozilla InvestiGator ("MIG") from Mozilla. MIG was introduced in 2013 by Julien Vehent as a fully open source project (Mozilla.org, n.d.). The intention for MIG was as a light-weight, portable and nimble incident response tool that could ingest IOCs from one or more systems, and query in near real-time for those same IOCs on any other systems enterprise-wide (Vehent, 2015). It may be considered an open-source competitor to commercial products like FireEye's Mandiant MIR or Bit9's CarbonBlack, and for organization with smaller budgets, could be an entry into threat detection across enterprise for minimal cost.

MIG functions using a centralized console to control agents installed on endpoints in an environment to interrogate those endpoints for interesting forensic artifacts. A Go package called a *module* can be utilized to perform specific functions on endpoints including inspecting files, scanning memory and conducting network status checks (Mozilla.org, n.d.-2).

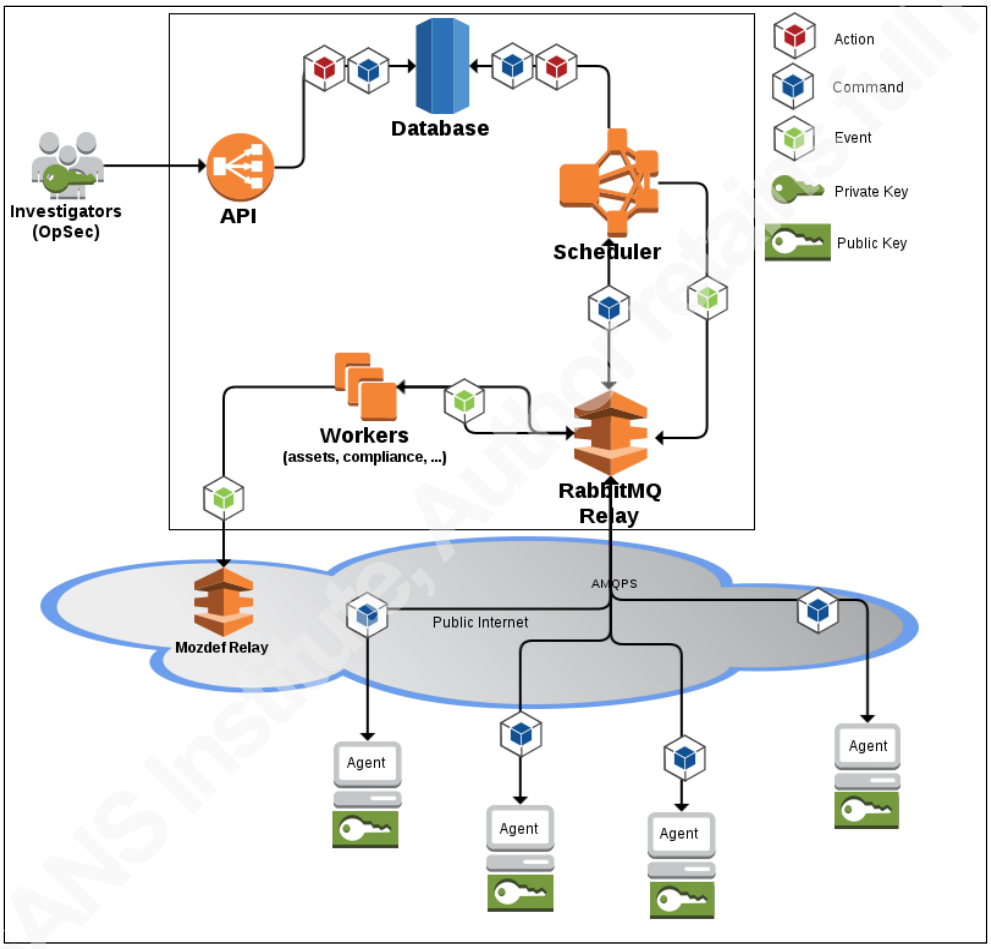

Figure 12. MIG High Level Architecture (Mozilla.org, n.d.-2)

One downside to using Go for modules is that Go is unable to load new modules dynamically at runtime, so all modules required must be compiled into the agent's binary before deployment (Mozilla.org, n.d.-3). This means that each time a new module is developed, a new agent binary would need to be compiled and deployed to all monitored systems. Obviously, this limits the nimbleness of MIG in situations where a module may need to be created or modified on the fly during an incident.

Shelly Giesbrecht, headnerd@nerdiosity.com

On the plus side, the agent is one static binary that has no dependencies (Vehent, 2015). This means that no additional software need be installed on an endpoint, and with platform support for Linux, Mac OS and Windows for most basic modules, MIG is highly portable.

Overall, MIG is still very new utility but with its open-source roots, the backing of Mozilla, and its very reasonable price, MIG is a viable option for small or large organizations with multi-platform environments and a need for fast detection on the endpoint.

4. Conclusion

For an incident responder, the difference between success and failure in an investigation can come down to the know-how they bring to the table. With more than 55% of incident responders lacking budgets for tools and technology (Torres, 2014), it is even more important than ever to be able to execute as precisely and quickly as possible. In this paper, we have examined the reasons why coding may be an important skill to learn and develop for incident response teams. Of those discussed, perhaps the single more important reason is simply that knowing how will make you a better responder. Even with commercial tools to aid in investigations, being able to script a solution to problem, having a command of what is going on "under the hood" and eventually, being able to devise a new utility to parse a yet undiscovered or little understood artifact should be considered a desired (if not mandatory) path for those on the road to IR proficiency.

In addition, we reviewed the history of some commonly used coding languages in Incident Response and Digital Forensics including Perl, Python, C #, PowerShell and Go. In discussing the relative merits and limitations of each language, the conclusion we can draw is that the choice of which language should be not based on mere preference for one over another, but should lay heavily in what the problem to be solved requires. While each language overlaps several others in the types of features it *can* do, it behooves us to look for the best code for the job.

With respect to text processing and manipulation, Perl is probably still the most likely choice for speed, flexibility, and performance but it lacks the usability and structure of

Python. On the other hand, for use in the development of more full-featured applications that execute quickly, and possibly include a GUI for a user-friendly experience, C# is the obvious candidate. However, Go is making gains in this area as it appeals to both traditional and modern coders.

Platform-specific issues may also be at the root of what language will work the best. For Windows environments where no other tools are available or allowed to be installed, or where the ability to interact natively with the operating system is critical, PowerShell will be a go-to selection. For multi-platform usage, a responder may likely decide on Python, or opt for the hybrid static/dynamic capabilities of the newer Go.

The tools we have discussed show the incredible depth and breadth of research and development going on within the Incident Response community. There is no doubt that they are all utilities that serious responders should already have in their arsenal or seriously consider adding. The one commonality between all these tools is that their developers openly encourage suggestions and discussion to enhance the functionality and usability, no matter what language being used.

A last word for future and current IR practitioners is that the path to proficiency lies in not only the betterment of ones' own skills, but in the enrichment of our practice in general. Learning to code will not only makes you a better responder, it will assist you in contributing more fully in the IR community at large.

Shelly Giesbrecht, headnerd@nerdiosity.com

5. References

Allen, Sam (n.d.). Dynamic. Retrieved from http://www.dotnetperls.com/dynamic

Asay, Matt (2014, March 13). Adoption of Google's programming language is rapidly gaining on Java and others [Blog post]. Retrieved from http://readwrite.com/2014/03/21/google-go-golang-programming-language-cloud-development

Banerjee, Asit (2012, November 9). Benefits of Perl Scripting [Blog Post]. Retrieved from http://asitbanerjee.blogspot.ca/2012/11/benefits-of-perl-scripting.html

Barbara, John J. (2012, August 10). Windows 7 Registry Forensics: Part 6 [Blog Post]. Retrieved from http://www.forensicmag.com/articles/2012/08/windows-7-registry-forensics-part-6

Carvey, Harlan (2007). Perl Scripting for Windows Security: Live Response, Forensic Analysis, and Monitoring [Kindle PC version]. Retrieved from Amazon.com

Carvey, Harlan (2010, May 10). About [Webpage]. Retrieved from https://regripper.wordpress.com/regripper/

Case, Andrew [attrc] (2014, March 13). I am wondering why many #dfir people seem so interested in 'PowerShell foresnics' [Twitter post]. Retrieved from https://twitter.com/attrc/status/444163636664082432

Cowen, David. (2013, July 10). Daily Blog #17: Milestones 11 And 12 Detailed [Blog Post]. Retrieved from www.hecfblog.com/2013/07/daily-blog-17-milestones-11 and-12.html

Cowen, David. (2015, February 19). Part 1 - Accessing an image and printing the partition table. In Automating DFIR - How to series on programming libtsk with python. Retrieved from http://www.hecfblog.com/2015/02/automating-dfir-how-to-series-on.html

Dillon, Shawn (2002, March 25). Why should you add C# to your skill set? [Blog post]. Retrieved from http://www.techrepublic.com/article/why-should-you-add-c-to-your-skill-set/1027622/

Donaldson, Toby (2003, April 25). Python as a First Programming Language for Everyone. Retrieved from https://www.cs.ubc.ca/wccce/Program03/papers/Toby.html

Forbes, Dennis (2013, July 23). The Most Powerful Feature of Go Is The Least Sexy [Blog post]. Retrieved from https://dennisforbes.ca/index.php/2013/07/23/the-most-powerful-feature-of-go-is-the-least-sexy/

Fulton, Scott (2006, June 14). TechEd 2006: Microsoft demonstrates first PowerShell release candidate. Retrieved from

Shelly Giesbrecht, headnerd@nerdiosity.com

http://www.tomshardware.com/news/teched2006-PowerShell-underlie-admin-guis,2959.html

Google Code (2015). [Table of available downloads and statistics June 18, 2015]. Google Code project for RegRipper. Retrieved from https://code.google.com/p/regripper/downloads/list

Hale-Ligh, M., Case, A., Levy, J., & Walters, A. (2014, July 28). The Art of Memory Forensics [Kindle PC version] Retrieved from Amazon.com

Hamilton, Naomi (2008, October 1). The A-Z of Programming Languages: C#. Retrieved from http://www.computerworld.com.au/article/261958/a-z_programming_languages_c_/?pp=2

Hull, Dave, (2014, July 18). Kansa: A PowerShell-based incident response framework [Blog post]. Retrieved from http://www.PowerShellmagazine.com/2014/07/18/kansa-a-PowerShell-based-incident-response-framework/

Hull, Dave, (2015a, April 23). Kansa [software]. Available from https://github.com/davehull/Kansa/blob/master/README.md

Hull, Dave, (2015b, April 29). Kansa: Autoruns data and analysis [Blog post]. Retrieved from http://trustedsignal.blogspot.ca/2014/04/kansa-autoruns-data-and-analysis.html

Jenkins, Tim (2014, March 6). How to Convince Your Company to Go With Golang [Blog post]. Retrieved from https://sendgrid.com/blog/convince-company-go-golang/

Kuhn, Bradley M. (2001, January 9). Background of Perl [Blog Post]. Retrieved from http://www.ebb.org/PickingUpPerl/pickingUpPerl_9.html

Kuhn, Bradley M. (2001, January 9). Regular Expressions [Blog Post]. Retrieved from http://www.ebb.org/PickingUpPerl/pickingUpPerl_7.html

Lee, Thomas (2005, November). Monad: The Future of Windows Scripting. Retrieved from https://technet.microsoft.com/en-us/magazine/2005.11.scripting.aspx

Levin, Mike (2011, January 5). Python Programming Language Advantages and Disadvantages [Blog Post]. Retrieved from http://mikelev.in/2011/01/python-programming-language-advantages/

McAllister, Neil (2012, January 3). 10 programming languages that could shake up IT [Blog post]. Retrieved from http://www.javaworld.com/article/2078503/mobile-java/10-programming-languages-that-could-shake-up-it.html

McQuaid, Jamie, (2014, August). Forensic Analysis of Windows Shellbags [Blog post]. Retrieved from http://www.magnetforensics.com/computer-forensics/forensic-analysis-of-windows-shellbags

Shelly Giesbrecht, headnerd@nerdiosity.com

Mikoluk, Kasia (2013, August 5). Perl vs Python: What's the Difference? [Blog Post]. Retrieved from https://blog.udemy.com/perl-vs-python/

Mozilla.org (n.d.). About Mozilla InvestiGator [Web page]. Retrieved from http://mig.mozilla.org/about.html

Mozilla.org (n.d.-2). Mozilla InvestiGator Documentation [Web page]. Retrieved from http://mig.mozilla.org/doc.html

Mozilla.org (n.d.-3). MIG Modules [Web page]. Retrieved from http://mig.mozilla.org/doc/modules.rst.html

MSDN (2008, October 28). PowerShell will be installed by default on Windows Server 08 R2 (WS08R2) and Windows 7 (W7)! [Blog post]. Retrieved from http://blogs.msdn.com/b/PowerShell/archive/2008/10/28/PowerShell-will-be-installed-by-default-on-windows-server-08-r2-ws08r2-and-windows-7-w7.aspx

MSDN(n.d.). Introduction to the C# Language and the .NET Framework Retrieved from https://msdn.microsoft.com/en-us/library/z1zx9t92.aspx

Newkirk, Al (2014, April 16). 10 Reasons to Never Use Perl, Ever [Blog Post]. Retrieved from https://engineering.tilt.com/never-ever-use-perl/

O'Brien, Gerry (2014, June 3). A brief history of C# [Transcript]. Retrieved from http://www.lynda.com/C-tutorials/brief-history-C/164452/176185-4.html

O'Connor (2010, April 1). Grow Your Own Forensics Tools: A Taxonomy of Python Libraries Helpful for Forensic Analysts [PDF document]. Retrieved from https://digital-forensics.sans.org/community/papers/gcfa/grow-forensic-tools-taxonomy-python-libraries-helpful-forensic-analysis_6879

Ochal, Zenon (2010, December 4). Dynamic Language Integration in a C# World [Blog Post] Retrieved from https://www.simple-talk.com/dotnet/.net-framework/dynamic-language-integration-in-a-c-world/

Pfeiffer, P. & Wang, L. (2002). A Qualitative Analysis of the Usability of Three Contemporary Scripting Languages: Perl, Python and Tcl. Retrieved from http://legacy.python.org/workshops/2002-02/papers/14/index.htm

Pike, Rob (2012, June 25). Less is exponentially more [Blog post]. Retrieved from http://commandcenter.blogspot.it/2012/06/less-is-exponentially-more.html

Pullega, Dan (2013, December 4). Shellbags Forensics: Addressing a Misconception (interpretation, step-by-step testing, new findings, and more) [Blog Post]. Retrieved from http://www.4n6k.com/2013/12/shellbags-forensics-addressing.html

Python Software Foundation (2008, December 3). Python 3.0 Release. Retrieved from https://www.python.org/download/releases/3.0/

Python Software Foundation (2015). [Software download page] Retrieved from https://www.python.org/downloads/

TechNet (2014, August 4). Scripting with Windows PowerShell. Retrieved from https://technet.microsoft.com/en-us/library/bb978526.aspx

Shelly Giesbrecht, headnerd@nerdiosity.com

The Go Programming Language. (n.d.). Retrieved from https://golang.org/doc/faq#history

Toal, Ray. Classifying Programming Languages [Blog Post]. Retrieved from http://cs.lmu.edu/~ray/notes/pltypes/

Torres, Alissa (2014, August). Incident Response: How to Fight Back [PDF document]. Retrieved from https://www.sans.org/reading-room/whitepapers/analyst/incident-response-fight-35342

Turner, James (2010, June 22). Does the world need another programming language? [Blog post]. Retrieved from http://radar.oreilly.com/2010/06/does-the-world-need-yet-anothe.html

Vanderplas, Jake (2014, May 9). Why Python is Slow: Looking Under the Hood [Blog Post]. Retrieved from https://jakevdp.github.io/blog/2014/05/09/why-python-is-slow/

Vehent, Julien (2015, May 29). MIG Mozilla InvestiGator: Distributed and Real-Time Digital Forensics at the Speed of the Cloud [PDF Document]. Retrieved from https://conferenc.hitb.org/hitbsecconf2015ams/materials/D2T2 - Julien Vehent - Mozilla InvestiGator.pdf

Volatility Foundation (2014). Retrieved June 18, 2015 from https://github.com/volatilityfoundation/volatility/wiki/Command%20Reference%20Mal#psxview

Volatility Foundation (2015a). Retrieved June 18, 2015 from https://github.com/volatilityfoundation/volatility/wiki/Command%20Reference#pslist

Volatility Foundation (2015b). Retrieved June 18, 2015 from https://github.com/volatilityfoundation/volatility/wiki/Command%20Reference#pstree

Volatility Foundation (2015c). Retrieved June 18, 2015 from https://github.com/volatilityfoundation/volatility/wiki/Command%20Reference#psscan

Volatility Foundation (2013). About [Web page] Retrieved from http://www.volatilityfoundation.org/#!about/cmf3

Wall, Larry. (2006, February 3). Present Continuous, Future Perfect [Blog Post]. Retrieved from http://perl.org.il/presentations/larry-wall-present-continuous-future-perfect/transcript.html

Zaïane, Osmar (2007, November 19). Python [PDF Document]. Retrieved from http://ugweb.cs.ualberta.ca/~c410/F07/410/presentations/ReportPython.pdf

Shelly Giesbrecht, headnerd@nerdiosity.com

Appendix

A. List of Languages

Language	Current Version	URL
Perl	5.22.0	https://www.perl.org/index.html
Python	2.7.10, 3.5.0b2	https://www.python.org/
C#	5	https://msdn.microsoft.com/en-us/library/z1zx9t92.aspx
PowerShell	5	https://technet.microsoft.com/en-US/scriptcenter/dd742419.aspx
Go	1.4.2	https://golang.org/

B. List of Tools

Tool	Language	Current Version	URL
RegRipper	Perl	2.8	https://regripper.wordpress.com/
Volatility	Python	2.4	http://www.volatilityfoundation.org/
ShellBagsExplorer	C#	5	http://binaryforay.blogspot.ca/p/software.html
Kansa	PowerShell	-	https://github.com/davehull/Kansa
Mozilla InvestiGator	Go	-	http://mig.mozilla.org/

Shelly Giesbrecht, headnerd@nerdiosity.com

Last Updated: December 11th, 2018

Upcoming SANS Training

Click here to view a list of all SANS Courses

SANS Sonoma 2019	Santa Rosa, CAUS	Jan 14, 2019 - Jan 19, 2019	Live Event
SANS Threat Hunting London 2019	London, GB	Jan 14, 2019 - Jan 19, 2019	Live Event
SANS Amsterdam January 2019	Amsterdam, NL	Jan 14, 2019 - Jan 19, 2019	Live Event
SANS Miami 2019	Miami, FLUS	Jan 21, 2019 - Jan 26, 2019	Live Event
Cyber Threat Intelligence Summit & Training 2019	Arlington, VAUS	Jan 21, 2019 - Jan 28, 2019	Live Event
SANS Dubai January 2019	Dubai, AE	Jan 26, 2019 - Jan 31, 2019	Live Event
SANS Las Vegas 2019	Las Vegas, NVUS	Jan 28, 2019 - Feb 02, 2019	Live Event
SANS Security East 2019	New Orleans, LAUS	Feb 02, 2019 - Feb 09, 2019	Live Event
SANS SEC504 Stuttgart 2019 (In English)	Stuttgart, DE	Feb 04, 2019 - Feb 09, 2019	Live Event
SANS Northern VA Spring- Tysons 2019	Vienna, VAUS	Feb 11, 2019 - Feb 16, 2019	Live Event
SANS Anaheim 2019	Anaheim, CAUS	Feb 11, 2019 - Feb 16, 2019	Live Event
SANS London February 2019	London, GB	Feb 11, 2019 - Feb 16, 2019	Live Event
SANS Dallas 2019	Dallas, TXUS	Feb 18, 2019 - Feb 23, 2019	Live Event
SANS New York Metro Winter 2019	Jersey City, NJUS	Feb 18, 2019 - Feb 23, 2019	Live Event
SANS Scottsdale 2019	Scottsdale, AZUS	Feb 18, 2019 - Feb 23, 2019	Live Event
SANS Secure Japan 2019	Tokyo, JP	Feb 18, 2019 - Mar 02, 2019	Live Event
SANS Zurich February 2019	Zurich, CH	Feb 18, 2019 - Feb 23, 2019	Live Event
SANS Riyadh February 2019	Riyadh, SA	Feb 23, 2019 - Feb 28, 2019	Live Event
Open-Source Intelligence Summit & Training 2019	Alexandria, VAUS	Feb 25, 2019 - Mar 03, 2019	Live Event
SANS Reno Tahoe 2019	Reno, NVUS	Feb 25, 2019 - Mar 02, 2019	Live Event
SANS Brussels February 2019	Brussels, BE	Feb 25, 2019 - Mar 02, 2019	Live Event
SANS Baltimore Spring 2019	Baltimore, MDUS	Mar 02, 2019 - Mar 09, 2019	Live Event
SANS Training at RSA Conference 2019	San Francisco, CAUS	Mar 03, 2019 - Mar 04, 2019	Live Event
SANS Secure India 2019	Bangalore, IN	Mar 04, 2019 - Mar 09, 2019	Live Event
SANS Bangalore January 2019	OnlineIN	Jan 07, 2019 - Jan 19, 2019	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced