Powercat

GIAC (GPEN) Gold Certification

Author: Mick Douglas, mick.douglas@gmail.com
Advisor: Richard Carbone

Accepted: February 21, 2015

Abstract:

Powercat brings the functionality and power of Netcat to all recent versions of Microsoft Windows. It accomplishes this goal by using native PowerShell version 2 components. This allows easy deployment, use, and little chance of being caught by traditional anti-virus solutions. Additionally, the latest versions of Powercat include advanced functionality that goes well beyond those found in traditional forms of Netcat. This paper will acquaint the reader with the original proof-of-concept version, developed by Mick Douglas; the current release, developed by Luke Baggett; and the outline of the planned future features that will be developed by anyone who wishes to join Team Powercat. Perhaps even you!

Powercat is freely available at Github.com:
https://github.com/besimorhino/powercat/blob/master/powercat.ps1

Acknowledgements

Powercat started as a proof-of-concept tool that I initially developed. However, the recent work has been done by the talented Luke Baggett. He has implemented almost all of the "Future Enhancements" first listed in the earliest versions of Powercat. In addition to Luke, I would like to thank my other coworkers at Black Hills Information Security for their support, encouragement, and constant challenges for me to always bring my "A game". Last, but certainly not least, I owe a special debt to John Strand, for believing in me well before I became a member of Black Hills Information Security.

I am blessed in both my professional and personal life. I know it, and for these wonderful gifts I am both humbled and grateful.

1. Introduction

Knowledgeable defenders are aware of the risks that attackers pose. In an attempt to halt malicious use, they will create segmented networks with various detection tools that are often signature based. This in turn, creates a demand for new attacker tools that can bypass signature detection while still being able to operate in highly segmented networks. As defenders learn about these new attack tools, attackers adjust their tools and techniques accordingly which creates yet another round of demand for "new and improved" attack tools that are eventually made. Thus, the lifecycle of attack and defend continues.

Defenders are continuing to get better at monitoring networks. As they do so, the attackers must become more stealthy, and able to navigate the environment in novel ways. While this allows defenders to gain the upper hand against less skilled attackers, this has the unintended consequence of pushing the most skilled attackers into areas of the network where they cannot be easily detected. This means that skillful attackers now have a distinct advantage. If an attacker is able to conduct successful reconnaissance and has a solid working knowledge of how a network and its component nodes are configured, they can effectively fly below the defender's radar. A perfect example of this is the trend where attackers make maximum use of the built-in features and functionality of the systems they have compromised. Two very powerful talks on this matter are "AT is the New Black" (Fuller and Gates, 2013) and "Living Off the Land" (Campbell and Graeber, 2013) which were given at DerbyCon 3.0. Both of these talks focused on using built-in OS commands as part of penetration testing. The bottom line is that by using built-in functionality, attack detection becomes exceptionally difficult. Because these are tools that are used as part of the OS stack, it is not practical to create a signature detection rule for them. Additionally, they are using expected ports and protocols; this makes it hard to identify malicious use from "normal." It is in this spirit that Powercat was developed.

Powercat is a PowerShell script that makes use of stock .Net components and PowerShell commandlets. It only uses items available to PowerShell version 2. This allows the script to run on any system from Windows 7 to Windows Server 2012. Until Microsoft officially deprecates the use of the PowerShell version 2 components used, Powercat will be able to work on any and all future Windows OS released without additional modification.

2. Powercat

2.1. Powercat History

Powercat was initially created as a "proof-of-concept" tool to determine if a PowerShell native backdoor listener and reverse shell (a.k.a. shell shovel) were practical. It was created by Mick Douglas during the Offensive Countermeasures class being taught at Black Hat USA 2014. The 0.1 version of Powercat was exceptionally limited in functionality. It could only perform listener or allow client connections which is the core of the Netcat family of applications. Its one redeeming feature was that it had the ability to natively send or receive a *cmd.exe* shell. One of the major drawbacks of this version of the tool was that it was using synchronous TCP sockets. The synchronous nature of the tool meant that Powercat had to make extensive use of while loops and sleep functions in order to determine if there was any output from either STDERR or STDOUT. This design lead to many performance issues. Specifically, from the user's standpoint, it was exceedingly slow. The command would have to complete before feedback of any kind could be sent to the Powercat user. This meant that if a command took a long time to run[1] the user would have to wait for the entire command to finish before getting any indication that the command was successfully received by the listener and executed. In spite of these limitations, it worked and was a welcomed tool used by the few who knew about it. Powercat's big breakthrough happened when Luke Baggett began actively working on the "Future Features" and implementing components of his own design into the tool. Luke's involvement in Project Powercat began shortly before DerbyCon 4.0, around September of 2014. Since this time, Mick Douglas has

[1] Some commands such as WMIC queries using WMI Query Language across an entire domain can take a long time to run. Depending on the number of nodes in the domain, complexity of the query, and network speed, it is reasonable to assume at least a half-hour run time. This feels *very* long when staring at the empty end of a remote command shell.

unfortunately not been able to contribute to the current development efforts. All enhancements discussed for the remainder of this paper have been implemented by Luke. It is Mick's hope to return to the development of this project soon.

2.2. Powercat Features

2.2.1. Modular Script

One of the first improvements that Luke Baggett made to Powercat was to make it a modular script. This ensures maximum portability, simplifies the evasion of PowerShell local execution restriction policies, and brings Powercat into alignment with other PowerShell penetration testing tools (such as the PowerSploit Framework). This flexibility has a minor cost in terms of use. Prior to running Powercat for the first time on a host, the module must be imported. This is done by using the Import-Module PowerShell commandlet.

```
Import-Module Powercat
```

From this point forward, until the PowerShell context is destroyed[2] (or the Powercat module is manually unloaded via the Remove-Module commandlet) the user can invoke Powercat at the PowerShell prompt:

```
Powercat [Powercat options]
```

Perhaps the greatest advantage to using this method is that PowerShell supports the loading of remote modules. This means that a user can have PowerShell retrieve the Powercat script for them:

```
IEX (New-Object
System.Net.Webclient).DownloadString("https://raw.githubuserconte
nt.com/besimorhino/powercat/master/powercat.ps1")
; powercat [options]
```

[2] PowerShell context is most often the PowerShell window that the script or module in which it is working. Closing the PowerShell prompt window or rebooting the system are just one of many ways the current working PowerShell context can be destroyed.

Mick Douglas, mick.douglas@gmail.com

While the above command is quite a bit to type, it would be reasonably assumed that a user would copy/paste this command. URL shortening services such as bit.ly[3] could make this a less arduous task if one is in a situation where they must type this URL in by hand. Finally, it should be stressed that this remote download technique can only work in situations where the host the user is attempting to invoke Powercat from has both Internet connectivity and the ability to access the remote resource. This technique is used in tools like Powercat and PowerSploit. Because of this, defenders should ensure that they have a properly configured web proxy. To offer maximum protection, the proxy should be configured to restrict access on an as-needed-basis for URLs, or at the very least domains, that serve a business use.

2.2.2. Asynchronous Communication

Currently, Powercat supports many features that give it a large degree of flexibility. The single biggest performance improvement was moving Powercat to an asynchronous communications model. This meant that one end of the wire (the listener or client) could receive any input from STDIN or output from STDERR or STDOUT, and the script would immediately perform the appropriate I/O update. This makes the user experience significantly more responsive. With a high speed and low latency network link, it would feel as if one is working with a normal – and local – *cmd.exe* shell (or the shell was invoked, e.g. PowerShell, *wmic*, *netsh*, etc.)

Powercat supports the use of either TCP or UDP sockets on any port, assuming the port is not already in use by another program, and the user or invoking process has permissions to the port. For instance, only elevated accounts, such as root on UNIX or administrator on Windows, are able to use any of the "low order" ports (ports 0-1024). This means that once the Powercat user has conducted a thorough review of the ports and protocols permitted between two systems, they should be able to bypass any router ACL or firewall rule using permitted ports. Furthermore, it also means that port sweeps via the

[3] Bit.ly is just one of many URL shortening services. This is accomplished by creating a shorter link which points to long or difficult URLs. When a user clicks a shortened link, the URL shortening service will send a 300 series status (a.k.a. redirection) code to the user's browser, causing the original link to load.

use of tools such as Nmap's Firewalk NSE script might be the only readily detectable indicator prior to the use of Powercat.

2.2.3. Interoperability with Other Applications

Of course, Powercat supports both client and listen (a.k.a. server) modes. Listen mode permits Netcat or Powercat clients to connect to the computer that is running the listener. This means that Powercat can be used to connect to any Netcat listener, or another Powercat instance listening on a remote host. It was a deliberate choice to allow other Netcat-like programs to interact with Powercat. It is hoped that this will speed its adoption and permit for a more flexible use of Powercat, in conjunction with the rest of its "relatives." Users have the flexibility to 'consume' Powercat network traffic as they see fit. This design choice should allow Powercat to be easily added to existing toolkits and further augment the methodologies of penetration testers without having to retool in order to better accommodate it.

2.2.4. DNSCat2 Protocol Compliance

Powercat has functional requirements that go well beyond compatibility and ease of adoption. Some of the major objectives behind the Powercat project are stealth, being able to get information out of networks that have tight egress controls, and allowing remote command and control through restrictive ingress/egress[4] filtering. To facilitate these three functional demands, Powercat has implemented the DNSCat2 protocol as specified by Ron Bowes (Bowes, 2015).

The DNSCat2 protocol provides a framework for creating clients or servers that are able to send data or command and control traffic over standard DNS traffic. For this technique to be viable, the Powercat user must have a domain for which they have control over the authoritative DNS server. Instead of running a standard DNS service daemon, the DNSCat2 user will run any DNSCat2 compliant server. In addition to the DNSCat

[4] Ingress filters are firewall rules that filter network traffic which is considered "inbound" or coming into your network from some other zone. Egress filters are for "outbound" traffic that has a destination not belonging to your zone. Through careful configuration of the rules governing both inbound and outbound traffic flows, network administrators control how the various points of a network can communicate.

program, there is an experimental Metasploit module that can be used. This module can be found at https://blog.skullsecurity.org/blogdata/dnscat-shell-win32.rb.

It is important to note that under normal use the DNSCat2 protocols do not create a direct connection over port 53. Instead, what is done is far more subtle, and in many instances a bit harder to track. The communication takes place over DNS record requests and responses that are encoded via various methods. This means for any network that allows external name resolution and recursion, the DNSCat2 protocol option will be a viable method to interact with remote systems, even in situations where there are no direct external outbound ports between the client system and the DNSCat listener. This remote access and control works in such a tightly controlled environment, because all traffic is happening over the DNS server infrastructure. After all, the ports and protocols must be permitted outbound from the DNS server if external (i.e. Internet) name resolution is to work. This technique makes one's DNS server infrastructure act as a proxy for the traffic or data the Powercat user choses to send over the connection.

2.2.5. Persistent Listener Built-in

Unlike the UNIX traditional Netcat implementation, Powercat has the ability to create "persistent" listeners. Those familiar with the Windows version of Netcat will note the similar functionality to the "listen harder" mode. The developers have opted to move away from the "-L" invocation method in the traditional Windows Netcat implementation. This was a deliberate choice, since PowerShell scripts and commandlets tend to be more verbose than their UNIX equivalents. By adopting a longer argument it is hoped that this 'feels' more akin to the native PowerShell commandlets. Additionally, Microsoft operating systems and their applications, including PowerShell, are not case sensitive (Microsoft, 2009). While it would be technically possible to create a command line parameter check that does accomplish case sensitivity on the listener parameter (capital L vs. lowercase l), it is not in keeping with standard Windows conventions and likely to confuse those who are not expecting this behavior.

The simple reason for having a persistent listener is that it creates a listener that will automatically restart itself if the process or connection is stopped for any reason. By having this built-in capability, it saves the user the effort of having to create a while loop, cron entry, or use some other facility to restart the listener process upon termination. This is a common task users of the traditional UNIX Netcat will likely be familiar with. While it is not that difficult to do any of those tasks, the developers' find these restart "hacks" to be irksome and sidestep this issue altogether by providing a method for restarting the listener without bothering the user. As handy as this capability is, not every listener will require this behavior. In fact, many will never need or use this; as such, this feature is an optional item that the user can select upon invoking it with the "–rep" argument (short for repeating). The command line syntax for that is as follows:

```
powercat -l –rep –p [port]
```

2.2.6. Relay Built-in

One significant advantage that Powercat brings to its users is built-in relays. Using traditional Netcat, one can create a pivot (think host-based proxy for Netcat traffic), but it is not exactly the most "user friendly" method. The "easiest" method for creating Netcat relays in UNIX requires the use of piped nodes. An example for creating a listener to client is shown below (Skoudis, 2014):

```
$ mknod backpipe p

$ nc -l -p [localport] 0<backpipe | nc [targetIP]
[port] | tee backpipe
```

While this is workable, it does generate some evidence that could be used as part of a forensics investigation. Specifically, the creation of a named pipe node file is odd. Not only does this command exist in the history file of the account used to create it, the file itself resides on disk in the directory the user was in when the mknod command was run. It is important to note that the file created, in the example above named 'backpipe', is a file type called a *named pipe*[5]. While named pipes can be of great use to systems administrators on UNIX machines, the author of this paper cannot think of a single reason a 'normal' user would need to create a named pipe. Because these file types are

[5] Named pipes are linkage files which allow inter-process communication between different running commands. Unlike the standard UNIX pipe character "|", named pipes will persist after a command completes execution. Additionally, named pipes permit communication between different hosts, which unnamed pipes traditionally cannot (Aoki, 2013).

so atypical, any incident responder with a modicum of UNIX familiarity should quickly zero in on this named pipe as needing further investigation no matter how innocuous a name it was given.

Powercat attempts to make things easier, and stealthier, for the user by natively including the relay functionality. Users do not have to create piped nodes or make use of odd command line redirects. Simply invoke Powercat as follows and the system this command was executed on will become a relay (proxy, pivot, etc.):

Client Relay Format:

```
powercat  -r <protocol>:<ip address>:<port>
```

Listener Relay Format:

```
powercat -r <protocol>:<port>
```

DNSCat2 Client Relay Format:

```
powercat  -r dns:<dns server>:<dns port>:<domain>
```

2.2.7. Self-Generating Payloads

One major drawback and area of concern is the monolithic nature of Powercat. All features and functions are in a single script. The current version of Powercat is over 900 lines of code and comments. It is expected to grow quickly as the code is built up as new features are introduced. While it is exceptionally convenient to have everything in one script, this does present several drawbacks. For a start, it is likely to be easier to detect. Were Powercat a static file, detecting it could be as simple as a single MD5 file checksum. This means it would be trivial for traditional signature-based tools to discover the script.

To avoid these issues, Luke Baggett implemented a payload generation feature in Powercat. By using the payload generator, Powercat will create a string that is a stripped down version of the script with only the code that is absolutely necessary to support the command line arguments that the user supplied:

```
powercat [listener or client options] -g
```

This 'lean and mean' version of Powercat now can potentially go places and be used in situations where the full version simply would not work. A hypothetical use case for this functionality could be a command injection flaw in a web application where the attacker is able to echo to a file one line at a time. By using this method, the attacker is able to "upload" the Powercat script up to the remote web server.

```
echo "some text" >> sample-file.txt
```

Attempting this sort of file transfer with the full version of Powercat would be exceptionally tedious, and could not be realistically accomplished without the use of some additional script to manage the line-by-line file transfer. By using the generated payload, again only containing the features one needs for a given invocation, the script would be significantly smaller in line count, thus making this hypothetical file transfer much more reasonable for the attacker.

2.3. Future Enhancements

Powercat is still a work in progress. Some additional features that are currently being tested but have not yet been released as "stable" include using SSL/TLS to mask data in transit, additional protocol encoding/enveloping schemes, and authentication modules.

2.3.1. SSL/TLS

SSL/TLS is a natural choice for helping to covertly move data or command shells. Since it provides the point-to-point encryption of data in transit, once this feature is available, it will allow users to hide their communications. Unless defenders are terminating all SSL/TLS traffic in every direction, Powercat's future SSL/TLS functionality will allow attackers a greater level of stealth than currently available to most versions of Netcat (Ncat by the Nmap project being a notable exception).

At the time of this writing, SSL library support native to PowerShell version 2 appears to be somewhat limited. There is ongoing research to determine the best method for invoking SSL/TLS encrypted communication. However, given the wealth of options the .Net framework offers PowerShell, it is only a matter of time until the necessary components are discovered. Several methods are available and it is only a matter of testing to determine if the features will work as needed for this script. While this "getting by" in a limited environment may be vexing for some developers, the author of this paper has taken a semi-defiant attitude toward those who claim something cannot be done. After all, at the time Powercat was first written, it was 'conventional wisdom' that PowerShell was not able to support asynchronous communication. The author of this paper takes great delight in that Luke Baggett was able to decisively and definitively prove everyone wrong.

2.3.2. User Authentication Methods

Shortly after the SSL/TLS features are implemented, an area of intense focus will be providing various authentication options. When conducting a penetration test, it is imperative that the network and systems being tested are not in any way harmed or have their security degraded. Doing so is not only sloppy work on the penetration tester's part, but this exposes the client to potential liability and needless risk that they otherwise would not have ever had to deal with. Yet many are doing just this when they are creating backdoor listeners in Netcat. If an attacker knows what port and IP address to interact with, they will get access. It is the author's opinion that having unauthenticated backdoor listeners is an unacceptable risk, and this practice should be terminated across the penetration testing industry as soon as practical.

In an attempt to make Powercat a 'safer' alternative, there are several developments already underway. With the implementation of SSL/TLS, Powercat will be able provide mutual authentication for both listener and client. Additional certificate-based login methods are being investigated. The goal is to implement something similar to a SSH key-based login. While certificate logins will be the preferred method for authentication, the developers understand that not everyone will adopt this higher security standard. In order to improve the current situation username and password-based authentication will be enabled as well.

2.3.3. File level encryption

As another method of providing protection to data in transit, different encryption methods can be used. If SSL/TLS is not selected by the user, they can choose to encrypt the data using other encryption methods including AES or ECC. While this mode could be considered for interactive shell use, it would be a very powerful protection mechanism for data moved via Powercat. One could think of this as file-level protection for traditional Netcat file push or pull methods.

2.3.4. Protocol-Based Hiding

With the exception of SSL/TLS protected traffic, Netcat style communication often stands out as being distinct from expected network data flows. So that this communication channel is better obscured, future versions of Powercat will include some form of protocol-enveloping. That is, the data stream will be sent over a specific data element inside a carrier protocol. Some of the protocols that the Powercat developers are currently investigating include HTTP and SMTP.

Techniques such as this already exist in many tools. For example, the freely available tool Naisho DeNusumu (Japanese for "hidden stealing" available for download at https://github.com/3nc0d3r/NaishoDeNusumu) by Adam Crompton, offers several techniques to conceal the movement of data in highly monitored networks. One of the exfiltration methods employed by Naisho DeNusumu is the use of web browser request cookie session ID values to conceal traffic. Since session IDs are typically long and appear to be random, this data field is a strong candidate for hiding encoded or encrypted communications.

By enveloping Powercat connectivity over expected ports and protocols, users of the tool will likely be able to remain beneath the radar of the defensive network monitoring team. The intention behind this functionality is to highlight how imperative it is for monitoring teams to deeply understand what constitutes "normal" network traffic.

2.3.5. Testing for IPv6

A final area of research to conduct will be the testing of Powercat over two different network addressing schemes. To date, all testing has been completed using IPv4. To ensure that Powercat is ready for the future, additional testing will be carried out to verify that Powercat is able to handle IPv6 tunneling over IPv4. This is better known as "Teredo" tunneling (Microsoft, 2003). This transition protocol will likely be used for a while as networks slowly move to IPv6. Finally, Powercat will need to be tested under native IPv6. Since Powercat is simply making function calls to existing libraries to handle the networking, it is highly likely that these tests will pass without significant issue. Cursory research on Microsoft's Technet site indicates that everything should work as expected (Microsoft, 2013).

2.4. Implications of Powercat

The breadth of functionality in Powercat mentioned earlier should serve as a wakeup call to those who practice network and system defense. Attackers for some time have been able to hide command and control or data transfers using typically expected protocols. Powercat's claim to fame is not one of functionality. Its chief advantages are twofold: it is freely available and it allows defenders to more accurately model the attacks they are likely to face. The developers hope that this availability affords defenders the opportunity to better study the techniques already employed by skilled attackers.

2.5. The Ethics of Creating Powercat

Powercat is a tool with many powerful features. How these capabilities are used will be entirely up to the individuals who use it. The developers' hope is that Powercat will be used responsibly, safely, and lawfully. However, it would be naïve to believe that this will always be the case.

It is curious that "hacking" tools are subjected to a higher level of criticism or scrutiny because they are perceived as somehow different from other applications. After all, a standard web browser can be used to launch a SQL injection attack against a web server, yet one rarely hears about complaints against browser developers. Microsoft's Office products can be used in the commission of crime. Software is simply a tool, a force multiplier. Whether it is a tool or a weapon is largely up to the wielder of the software itself.

However, creating a tool with such a strong potential for evil has given the developers pause. After careful consideration, the developers of Powercat firmly believe that defensive practitioners, researchers, and others defenders who are within the law require access and use of tools that allow the modeling of complex attacks. Without this capability, there is no effective means to adequately test their defenses against the realistic attacks and tactics their adversaries will use against their infrastructure. In short, the potential for good outweighs the potential for harm.

The Verizon Data Breach Report has clearly shown that criminals have already developed their online presence (Verizon et al, 2014). The public and widespread release of Powercat does little if anything for the attackers, but it could help open defenders' eyes to the grim reality as Dan Greer so eloquently put it at the 2014 RSA security conference:

> Whether in detection, control, or prevention, we are notching personal bests, but all the while the opposition is setting world records (Greer, 2014).

Finally, it is worth pointing out that for all the evasion techniques and capabilities Powercat has, there are no vulnerability exploitation capabilities nor is there any on the development roadmaps. Powercat has and always will be a tool to highlight what an attacker *could* do, not necessarily be the go to tool to be used by attackers. This is a fine point to be sure, but nevertheless an important one. Powercat could be used for evil, but this is unlikely as it is limited in its functionality and features as compared to currently available crimeware

While there is a certain "gallows humor" to the following quote from the TV Show "All in the Family," it does accurately sum up the author's feelings and is highly apropos on the matter of munitions grade software:

> Gloria: Did you know that sixty five percent of the people murdered in the last ten years were killed by handguns?
> Archie Bunker: Would it make you feel any better, little girl, if they was pushed out of windows (Bloom and Nicholl, 1972)?

3. Conclusion

Current defensive best practices make extensive use of network segmentation and signature detection. Combined, these two techniques significantly raise the level of difficulty for penetration testers. By utilizing Powercat, these two controls, which are common in almost any organization, can be negated by using a single tool. Since Powercat is based entirely on PowerShell version 2, it contains no binaries to trip

IDS/IPS, anti-virus, or file integrity monitoring to detect. Additionally, it does not, and will not ever require the user to install dependencies. This means that Powercat is a flexible and useful tool that should be seriously considered and studied by all penetration testers. It effectively allows one to test a variety of different attack scenarios.

Defenders must be aware of this tool and its implications. Many of the traditional defense technologies in use today (anti-virus, firewall, etc.) will not ever be able to mount effective protections against Powercat. In light of this new reality, it is incumbent on defenders to know what "normal" network traffic is. One must also be aware of what expected processes and scripts should be running on a given host under "normal" conditions. Defense can no longer simply focus on alerting to "known bad" problems. Instead, defense must pick up the gauntlet thrown and take a more active role. They must investigate for deviations from baselines and aggressively seek to understand why these variances are taking place. To do any less is a dereliction of duty!

As a member of Team Powercat, I invite you to download this PowerShell module, and with appropriate permission, see how your defenses stand up to its use. We would love to know what you think about this tool. Let us know if you run into any issues while using it. Chances are you will not be alone in experiencing these problems. You could be saving your friends from future frustrations. If you have any ideas on how to improve or extend Powercat, please let us know via Github pull requests. If you have the time, ability, and inclination to work with the code, we would be delighted to have your assistance. Finally, we would be eternally grateful if you, dear reader, helped spread the word about this tool and its capabilities. We firmly believe that defenders need to know and understand the technical capabilities that the attackers have. In the right hands, Powercat is uniquely qualified to help shape and create a significant portion of a realistic and comprehensive view into the technical risks that organizations are facing. Getting the word out about Powercat so others can more accurately measure their risk exposure is perhaps the greatest gift you can bestow to the developers.

4. References

Aoki O. (2013) Debian Reference. V2. [website] Debian Foundation. https://www.debian.org/doc/manuals/debian-reference/ch01.en.html#_named_pipes_fifos [Accessed 21, Feb. 2015]

Bloom, G and Nicholl, D (Writers), & Campbell, N (Director). (Original broadcast 9/16/1972). Archie and the Editorial [Television series episode]. In N. Lear (Producer), *All in the Family.* Los Angeles, CA: CBS Television City (Studio) Norman Lear/Tandem Productions (now a division of Sony Pictures Television) (Distributors)

Bowes, R. (2015) [website]Available at: https://wiki.skullsecurity.org/Dnscat#Protocol [Accessed 21, Feb. 2015]

Campbell, C. and Graeber, M. (2013). [DerbyCon Presentation] *Living Off the Land* Recording available at: https://www.youtube.com/watch?v=j-r6UonEkUw [Accessed 21, Feb. 2015]

Crompton, A. (2014) [Notacon Presentation] *Naisho DeNusumu: Stealing Secretly Exfiltration Tool Framework* Recording available at: https://www.youtube.com/watch?v=4K_-IUGqGdg [Accessed 21, Feb. 2015]

Gates, C. and Fuller, R. (2013). [DerbyCon Presentation] *AT is the New Black.* Recording available at: https://www.youtube.com/watch?v=_8xJaaQlpBo [Accessed 21, Feb. 2015]

Greer, D. (2014). [RSA Presentation] *We Are All Intelligence Officers Now* Transcript at: http://geer.tinho.net/geer.rsa.28ii14.txt [Accessed 21, Feb. 2015]

Lyon, G "Fyodor" (date not listed). [website] Ncat Introduction Available at: http://nmap.org/ncat/ [Accessed 21, Feb. 2015]

Microsoft, et al. (2003) [website] Teredo Overview Available at: https://technet.microsoft.com/en-us/network/cc917486.aspx [Accessed 21, Feb. 2015]

Microsoft, et al. (2009) [website] Using Cmdlets. Available at:

https://technet.microsoft.com/en-us/library/bb648607(v=vs.85).aspx [Accessed 21, Feb. 2015]

Microsoft, et al. (2013) [website] Net TCP/IP Cmdlets in Windows PowerShell https://technet.microsoft.com/en-us/library/hh826123.aspx [Accessed 21, Feb 2015]

Skoudis, E. (2014). 1st ed. [ebook] SANS, p.2. Available at: http://www.sans.org/security-resources/sec560/netcat_cheat_sheet_v1.pdf [Accessed 21, Feb. 2015]

Verizon, et al. (2014)Verizon Data Breach Report, 2014. 1st ed. [ebook] Verizon. Available at: http://www.verizonenterprise.com/DBIR/2014/ [Accessed 21, Feb. 2015]

Mick Douglas, mick.douglas@gmail.com