572.5

Encryption, Protocol Reversing, OPSEC, and Intel

SANS

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

572.5

Encryption, Protocol Reversing, OPSEC, and Intel

SANS

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

FOR572.5 | Advanced Network Forensics and Analysis

Encryption, Protocol Reversing, OPSEC, and Intel

Authors:
Phil Hagen, Lewes Technology Consulting, LLC
phil@lewestech.com | @philhagen
Mat Oldham
mat.oldham@gmail.com | @roujisecurity

http://twitter.com/sansforensics

This page intentionally left blank.

SANS DFIR

DIGITAL FORENSICS & INCIDENT RESPONSE

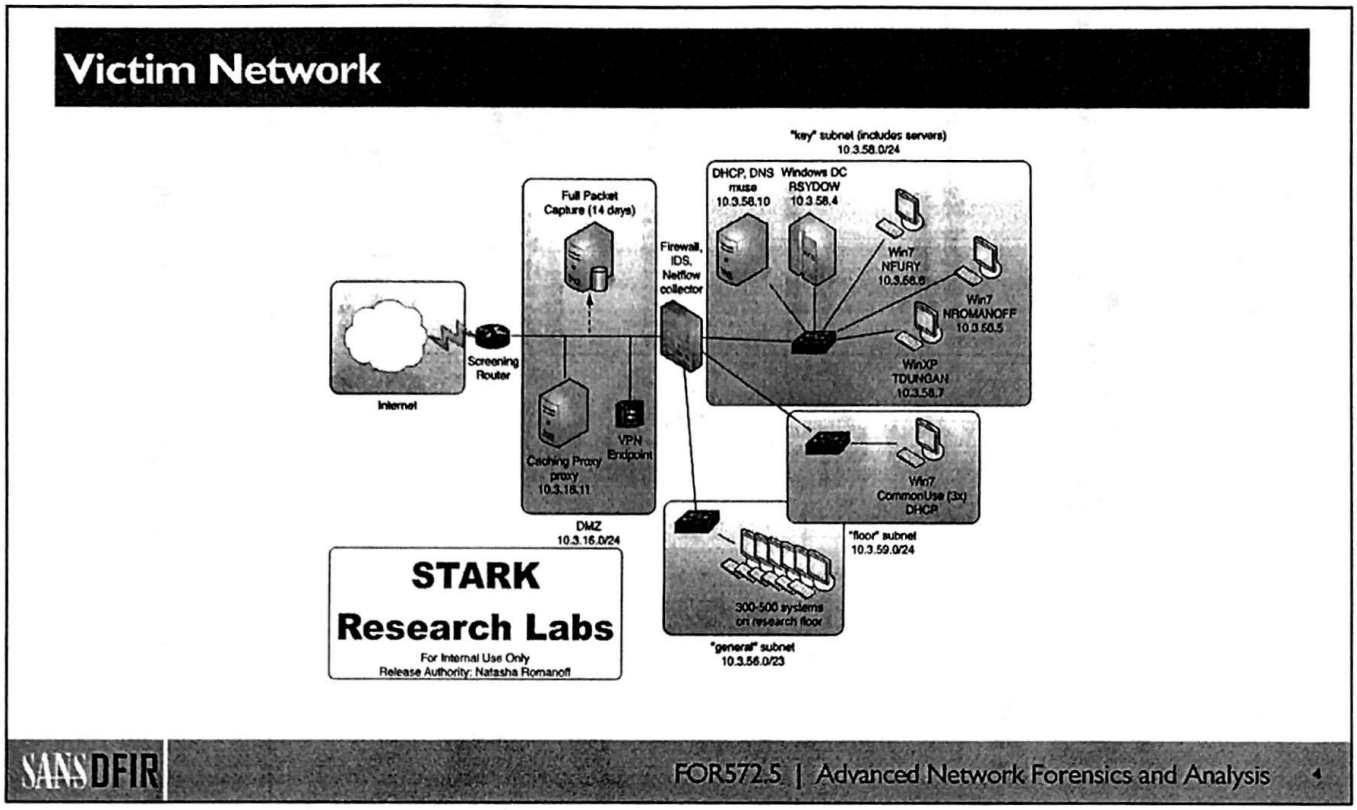
OPERATING SYSTEM & DEVICE IN-DEPTH

- FOR408 **Windows Forensics**
- FOR518 **Mac Forensics**
- FOR526 **Memory Forensics In-Depth**
- FOR585 **Advanced Smartphone Forensics**

INCIDENT RESPONSE & ADVERSARY HUNTING

- FOR508 **Advanced Incident Response**
- FOR572 **Advanced Network Forensics and Analysis**
- FOR578 **Cyber Threat Intelligence**
- FOR610 **REM: Malware Analysis**
- SEC504 **Hacker Tools, Techniques, Exploits, and Incident Handling**
- MGT535 **Incident Response Team Management**

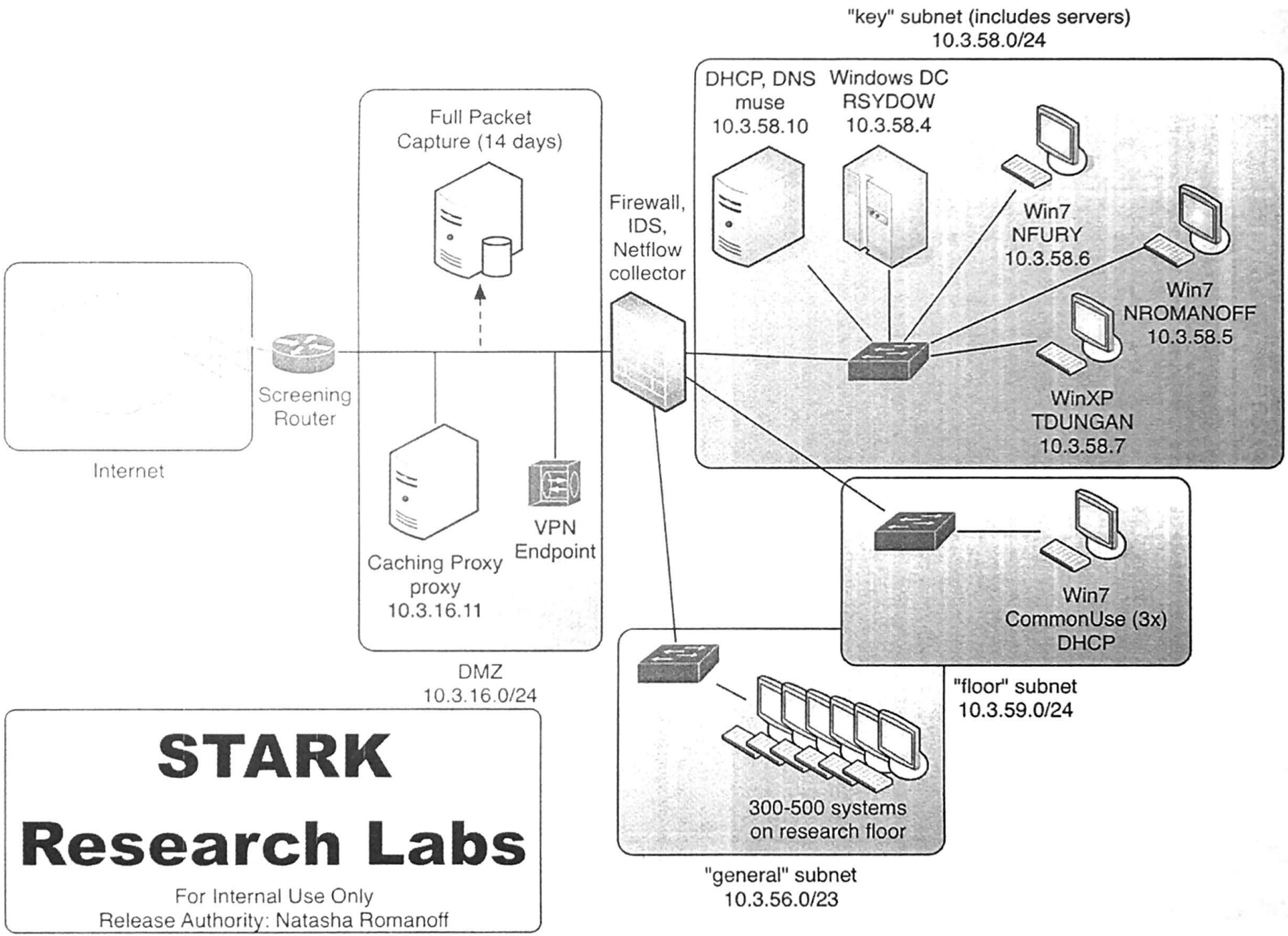
@sansforensics | sansforensics | dfir.to/DFIRLinkedInCommunity | dfir.to/gplus-sansforensics | dfir.to/MAIL-LIST



The SRL network is representative of many medium-sized enterprises, with several major internal segments separated by a single firewall, but very little network protection between hosts once on the inside of that perimeter.

Key systems we are aware of (not an exhaustive list) include:

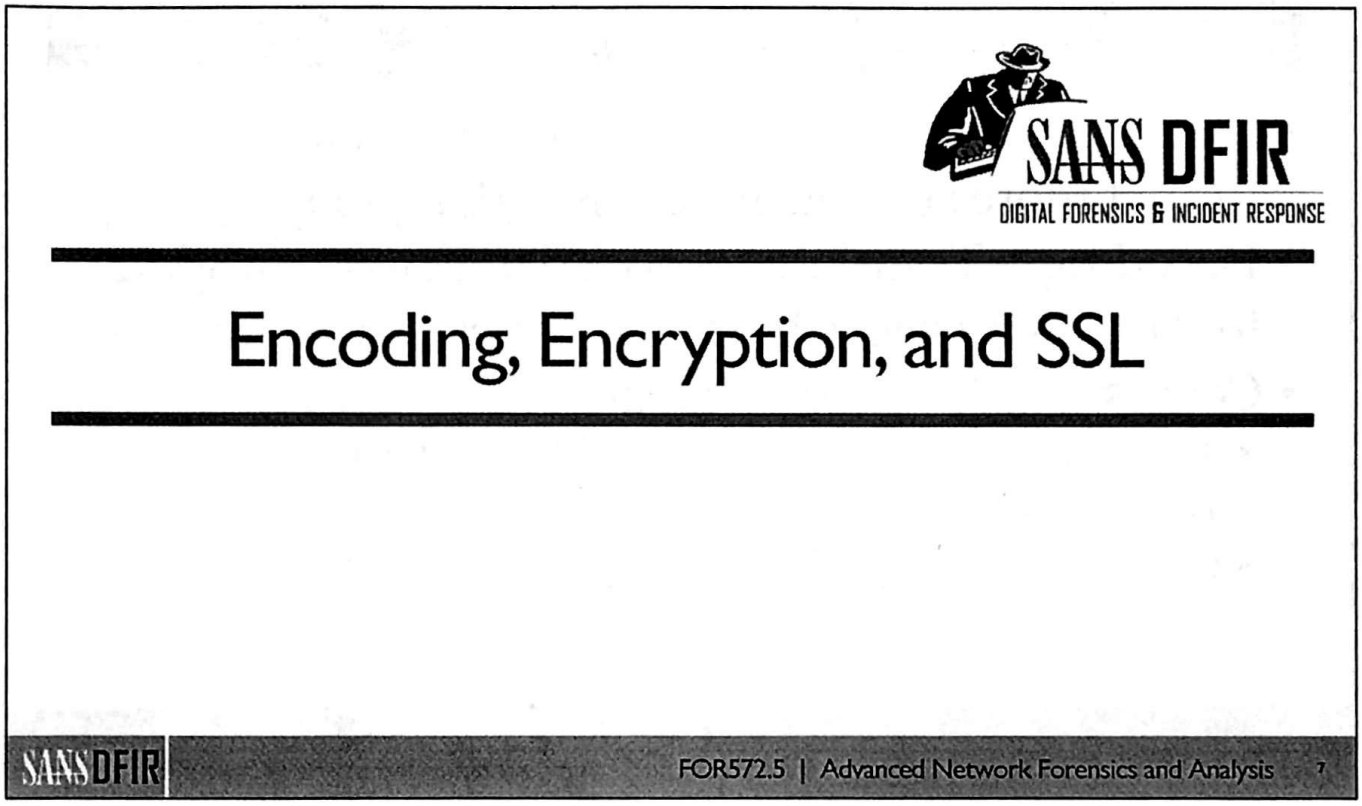
Host	IP Address	Role
Router	10.3.16.1	Inside interface of Internet screening router
Firewall	10.3.56.1/23 10.3.58.1/24 10.3.59.1/24 10.3.16.99/24	General administration and users Key users & SHIELDBASE servers subnet Production floor systems External (DMZ) interface
VPN	10.3.16.5/24	VPN Concentrator external interface
Proxy	10.3.16.11/24	Caching proxy for outgoing HTTP
Sniffer	10.3.56.254/24	Tcpdump rotating buffer. Approx 14 days of storage
Webserver	10.3.16.3/24	External presence of SRL
rsydow	10.3.58.4	SHIELDBASE Domain Controller – Windows 2008r2
tdungan	10.3.58.7	Tim Dungan's R&D workstation
nromanoff	10.3.58.5	Natasha Romanoff's workstation
nfury	10.3.58.6	Nick Fury's workstation



Encryption, Protocol Reversing, OPSEC, and Intel

- Encoding, Encryption, and SSL
- Lab 12: SSL Inspection
- Man-In-The-Middle
- Network Protocol Reverse Engineering
- Lab 13: Identify Undocumented Protocol Features
- Investigation OPSEC and Threat Intel
- Lab 14: Mini-Comprehensive Investigation
- Capstone Challenge Preparation

This page intentionally left blank.



This page intentionally left blank.

Encoding

- Encoding – Translates a piece of input data into format more suitable for transport/storage
- Decoding – Translates encoded data to its original format—message is the same as before encoding
- Covers a wide range of needs:
  - Text encoding: ASCII, UTF-7, UTF-8, UTF-16
  - Binary encoding: Base64
  - Compression: MPEG2, MPEG4, MP3, H.264

Developers and security professionals often mistakenly use the terms "encoding" and "encryption" interchangeably. Although both operate on a given piece of data, the end results are significantly different. When misused, this error can cause serious security issues.

Encoding is the process of translating a piece of data into a more suitable, usually publicly available format. Decoding is the process of translating the encoded data back to its original format. Encoding is used in many different ways within our day-to-day activities:

- Textual representation of data, such as ASCII, UNICODE, UTF-7, UTF-8, UTF-16
- Translating unprintable binary data into a printable format, such as Base64/MIME encoding
- Various compression algorithms: MPEG3, MPEG4, MP3, H.264

Technically, an encoding algorithm is known as a "cipher" that is used to generate an encoded message called "ciphertext". The decoded equivalent text is referred to as "plaintext". While technically correct terms, we will use the terms "input message" and "output message" instead to avoid confusion with encryption, which will be addressed in a moment.

Base64 Encoding (1)

- Translates binary data to printable ASCII characters
- Uses 64 output characters:
  - A-Z, a-z, 0-9, ("+" and ("/" or "-"))
  - May use "=" for padding

```
--Apple-Mail=_F387EA08-6CDF-4EFE-B115-E37BE6319094
Content-Type: application/pdf;
        name="LTC Invoice 2015.0079.pdf"
Content-Transfer-Encoding: base64

JVBERi0xLjQKJcfsj6IKNSAwIG9iago8PC9MZW5ndGggNiAwIFIvRmlsdGVyIC9GbGF0ZURlY29k
ZT4+CnN0cmVhbQp4nKlYWXMcRQx+318xVfCwQ7Gdvg/ecAiHKxxxtooHigfHhMTgg8QOqfx71KfU
afvnirMQvOB5U0yGc2bl5JRmetpern5b/zBvJDNKOrG+mjeCmcCtWv87g54JoLe+xtVz1D1DhZez
...
UgovSUQgWzxDNjYzMTM5NENDNDhBQTI1NjNGMEU5OTkyOTQwRDg1Nj48QzY2MzEzOTRDQzQ4QUEy
NTYzRjBFOTk5Mjk0MEQ4NTY+XQo+PgpzdGFydHhyZWYKMTQ2NTMKJSVFTDYK
--Apple-Mail=_F387EA08-6CDF-4EFE-B115-E37BE6319094
...
```


Base64 encoding is one of the most common encoding formats used on the Internet. Base64 encoding is commonly used to encode binary data that needs to be transferred or stored in a textual (usually ASCII) data format. This ensures the data remains intact during storage and transmission. Some examples of base64 encoding include encoding attachments associated with e-mails within the MIME format, storing complex data within XML structures, and HTTP transactions.

At a high level, base64 encoding takes 3 bytes of input data and encodes it into 4 bytes of output data using a predetermined alphabet. The alphabet consists of a total of 64 "printable" characters:

- 26 uppercase Latin/English letters (A-Z)
- 26 lowercase Latin/English letters (a-z)
- 10 numerical digits (0-9)
- 2 special characters ("+" and "/"), however the "-" may be used interchangeably with the "/" because some fields may contain base64 data as well as filenames/paths or URI strings

Because this adds one extra byte of output message size for every three bytes of the input message, the overall output message size will be roughly 137% of the input. In addition, if the number of bits of the input message is not divisible by 24, additional padding is added to the encoded message. This is typically handled by adding one "=" to the end of the base64-encoded message for each missing byte. Examples of this are below:

- "SANS FOR572 ROCKS!" => "U0FOUyBGT1I1NzIgUk9DS1Mh"
- "SANS FOR572 ROCKS" => "U0FOUyBGT1I1NzIgUk9DS1M="
- "SANS FOR572 ROCK" => "U0FOUyBGT1I1NzIgUk9DSw=="

Note, however, that the padding is not fundamentally required for the decoding algorithm, because the number of missing bytes would be trivial to calculate based on the size of the encoded message. Some base64 implementations omit the padding entirely.

References:
http://for572.com/98ug3

Base64 Encoding (2)

- Alphabet

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Character	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Index	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
Character	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

Index	52	53	54	55	56	57	58	59	60	61	62	63
Character	0	1	2	3	4	5	6	7	8	9	+	/

- Example encoding

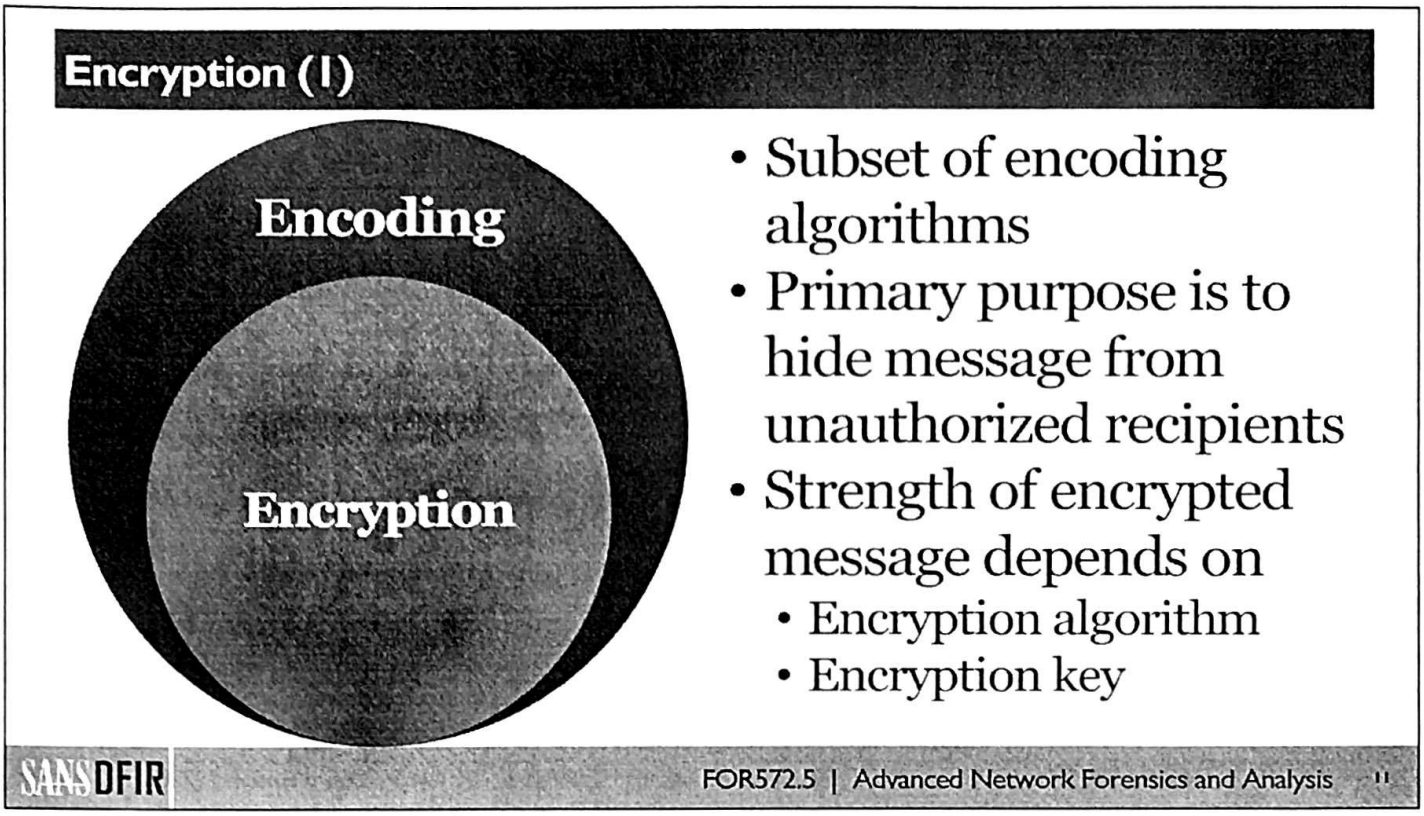
Text	S								A								N								S							
ASCII Hex	0x53								0x41								0x4E								0x53							
Decimal	83								65								78								83							
Binary	0	1	0	1	0	0	1	1	0	1	0	0	0	0	0	1	0	1	0	0	1	1	1	0	0	1	0	1	0	0	1	1	0	0	0	0
Output Index	20								52								5								14							
Base64	U								0								F								O							

As an example, say we want to convert the string "SANS" into a base64 format with a standard base64 alphabet. For each character, we first translate the text value into its corresponding ASCII hex representation. For example, the "S" character in ASCII hex has a value of 0x53. This is further decoded into the corresponding binary representation of 1s and 0s that turns the "S" into the bit string "01010011". Because the alphabet consists of 64 characters and each character in the output message requires just 6 bits to represent one of its 64 characters (i.e., 2^6), we take the first six bits (010100) and translate that to a corresponding number, which, in this case, is 20. The value 20 is the index into the base64 output alphabet that then corresponds to the "U" character. Therefore, the first value of our base64 encoded string is the "U".

There are still 2 bits (11) leftover that were not used as part of the translation to the "U" character. However, we need a total of 6 bits to represent the base64 output characters; so, we take the remaining two bits of the first input character and append the first four bits of the second character, an "A", which gives us the next six-bit value (110100). This translates to the number 52, which is then used as an index into the base64 output alphabet—the character "0".

This process continues as above for each character, until we reach the second "S" character from the input message. Because the output alphabet requires six bits, we pad the final two bits of the input character "S" with zero bits, resulting in a "110000". This maps to the "w" character in the base64 output alphabet. Decoding the output message back to its original format would simply use the same process in reverse.

As you may have recognized, successfully encoding and decoding messages using this algorithm fundamentally depends on using the specified base64 alphabet. If different characters were used, or the standard character set was jumbled in some way, only the parties that know that alphabet mapping would be able to decode the message. Some attackers have used the base64 algorithm and character set with a nonstandard mapping as a sort of rudimentary **encryption** algorithm, because the decoding process requires secret information—or a key—consisting of that mapping.



As mentioned prior, encoding is typically used to translate data into a format that is more appropriate for a given transport or storage method. A subset of encoding algorithms is used for encryption, where the purpose of the encryption algorithm is to secure the contents of message between sender and receiver. Although all encryption algorithms fall into the larger set of encoding algorithms, not all encoding algorithms are encryption algorithms.

Encryption is the process of disguising a message in such a way to hide its contents from unauthorized readers. The technical terms "plaintext" and "ciphertext" still apply in this arena and have the same definitions. However, in this case, the specialized form of encoding process is most accurately called "encryption", and the reverse—getting plaintext back—is called "decryption". The encryption process consists of multiple components:

- The message (M) is the data that is to be hidden from unauthorized users.
- A key (K) is a unique value used by the encryption/decryption algorithms to generate a more secure ciphertext.
- The encryption algorithm (E) is the process used to convert the message into ciphertext.
- The decryption algorithm (D) is the process used to convert the ciphertext back to a plaintext message.

This essentially allows for the following:

- D(E(M)) = M (Read: the decryption of the encrypted message M is the original message M)

Determining the strength of the encrypted message depends primarily on two factors:

- The strength of the encryption key
- The strength of the encryption algorithm

Encryption (2)

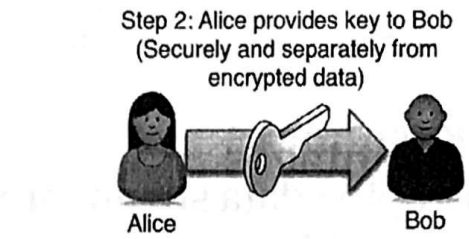
- Strength of algorithm
  - Depends on public knowledge of the algorithm
  - Depends on likelihood of mathematical collisions
- The overall strength of the key
  - Depends on the length of the key
  - Depends on the randomness of the key

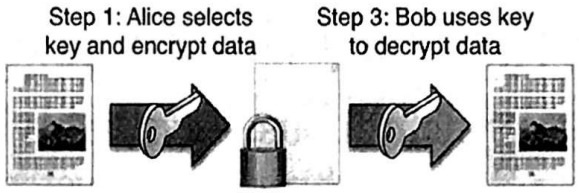
The strength of the encryption/decryption algorithm
Custom algorithms that are known only to the sender and receiver of the messages are known as restricted algorithms. The strength of an encrypted message is based on how many individuals know the algorithm the message was encrypted with. If someone leaves the group or the algorithm is compromised, all users of the group must change the algorithm used. This method is commonly used in low-security applications. This method is also typically defeated easily through reverse engineering of the applications developed.

The strength of the key
In cases where the algorithm is known by users outside the group, a key is used to add an extra layer of security. This is common for mass-produced applications because this allows for the algorithm to be scrutinized for weaknesses. The strength of a key is typically determined by the size of the key and randomness of the key. When multiple keys can produce the same output, a collision occurs.

Symmetric Key Encryption (1)

- Single "private" key is used for both encryption and decryption
- Strength: Computationally cheap
- Weakness: 100% depends on key security
- Two categories:
  - Stream ciphers
  - Block ciphers





There are two primary types of key-based algorithms: symmetric and asymmetric.

In a symmetric encryption algorithm, the same key is used by all users to encrypt and decrypt messages. Due to the shared key, typically symmetric key algorithms are less computationally expensive, which is a symmetric key algorithm's biggest strength.

However, the biggest drawback to the security of this type of encryption system is that the security of the system is wholly dependent on the security of the key. Both the sender and receiver must agree on the key; however, if the key is ever exposed, the encryption system is open to attack. If one individual from the group leaves, every member must change their key. Even if the group does change the key, all previous communications using the previous key value can still be decrypted by any party with knowledge of the old key value.

To demonstrate symmetric key encryption, assume Alice wants to send Bob an encrypted message. Alice types the message she wants to send to Bob and encrypts it with a password (or encryption key). Alice sends the message to Bob via e-mail and calls him to tell him the password. Bob then uses the password to decrypt and read Alice's message.

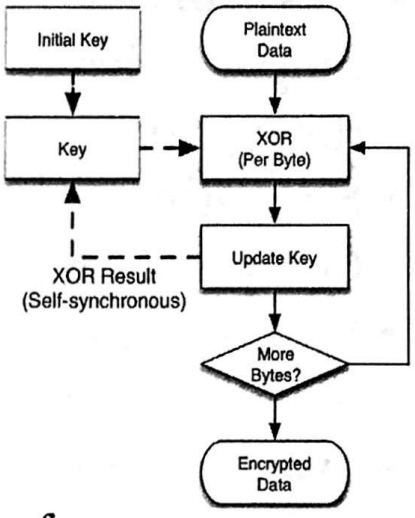
However, any party that acquires the key at any time would be able to decrypt all messages ever encrypted with that key value—so key security is absolutely paramount.

There are two specific types of symmetric encryption algorithms:

- Stream ciphers: Algorithms that operate on a single byte of the plaintext message at a time
- Block ciphers: Algorithms that operate on a group of bits or bytes at a time

Stream Ciphers: Encrypt One Byte at a Time

- Advantages
  - Ideal for variable length
  - Keystream adds randomness
- Disadvantages
  - Corrupted data stream breaks cipher
  - Slower without specialized hardware
- Two major cipher modes
  - Synchronous: Keystream independent of message
  - Self-synchronous: Keystream based on ciphertext



Stream ciphers are encryption algorithms that operate on one byte or character at a time. Stream ciphers typically start with a key called the seed. The key is then used to generate a keystream of a fixed or variable size. The keystream is then used to either encrypt or decrypt the message one byte at a time. During encryption or decryption, the keystream may be updated. Common examples of stream ciphers include the German Enigma, Wireless Communications (WEP), RC4, and several high-sensitivity communication systems.

There are two popular ways to generate the keystream:

- A keystream is generated independently of the message (this is called synchronous cipher text).
- A keystream is derived from the previous N number of bits (this is called self-synchronous cipher text).

Overall there are several advantages to stream ciphers:

- They are ideal for variable length messages.
- They are less susceptible to cryptanalysis attacks, because each portion of the message is encrypted with a different portion of the key.

There are several disadvantages as well:

- A transmission error with a stream cipher breaks the rest of the transmission. This requires both data and keystream to be resynchronized to continue the stream cipher.
- Typically, because a stream cipher operates on an individual character, the overall process is slower. However, stream ciphers can typically be implemented in specialized hardware to significantly increase the encryption and decryption process.

There are two major stream cipher modes: synchronous and self-synchronous cipher text.

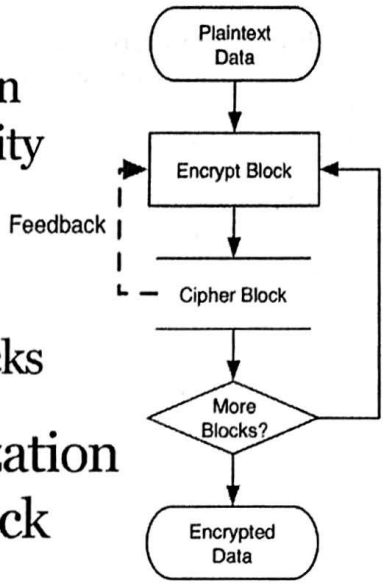
In synchronous cipher text, the keystream is generated independently of the message. To decrypt a message, the attacker would not only need to intercept the message, but they also must have knowledge of the keystream used to encrypt the message. A good example would be the German Enigma Machine during World War II. The German Enigma Machine required both the transmitter and receiver to reset their machines during a set period of time. The

transmissions between two machines required the same exact settings (rotors, wiring, and plugs) that represented the key to be consistent between the two machines. As the sender transmitted messages to the receiver, a keystream was generated based on the rotation of the rotors for each character that was typed over a given time period.

In self-synchronous cipher text, the keystream is generated based on prior message bytes. This has an added value over synchronous cipher text where any modification of any part of the stream will affect decryption of the rest of the stream. Essentially, this provides a "tamper-evident" functionality. The flowchart depicts a self-synchronous encryption algorithm, this one using a chained XOR encryption technique. Given the plaintext data and an initial key, the first byte of data is XOR'd with the initial key. The result of this operation is then used as the key to encrypt the next character of data. For the entire length of the message, each subsequent byte of data is XOR'd with the result of the previous operation. Because the keystream (i.e., the list of bytes used as a key for each XOR operation) is generated based on the prior message, the entire algorithm is self-synchronous.

Block Ciphers: Encrypt a Block of Data at a Time (1)

- Advantages
  - Feedback loop improves randomization
  - Can validate integrity and confidentiality
- Disadvantages
  - Identical blocks produce same results
  - More susceptible to cryptanalysis attacks
- Initialization vectors add randomization
- Feedback modifies key for each block



Block ciphers are encryption algorithms that operate on a fixed-length block of data at a time. Typically, block size is a power of two—most commonly 64 or 128 bits. In most cases, the cipher block used to encrypt the block of data is the same size. When the available input data is shorter than the cipher block, padding—usually `0x00` bytes—is added to the end of the input data to match the necessary cipher block size.

A block cipher's main advantage over a stream cipher is twofold:

- When encrypting or decrypting a message that is longer than the cipher block length, the message must be partitioned into segments that are the same length as the cipher block. Additional concepts adapted from stream ciphers, such as input vectors and feedback, help increase the overall strength of the cipher.
- Certain cipher block algorithms have built-in modes of operation that can check both the integrity of the data being encrypted or decrypted in addition to its confidentiality. Stream ciphers typically don't have this ability.

Block ciphers do have some weaknesses, however:

- Identical blocks of data being encrypted typically produce the same results.
- They are more susceptible to cryptanalysis attacks because the key may be the same for all blocks of plaintext, resulting in duplicate inputs creating duplicate outputs.

Block ciphers typically implement one of several different types of modes of operation. A mode of operation essentially modifies how the block cipher algorithm works from one block to another, thus adding randomization to the data.

To address shortcomings in a standard block-cipher algorithm, designers have implemented two modes of operation that offer significant improvements: initialization vectors and feedback.

An initialization vector is a random or pseudorandom block of data that is used to "initialize" a cipher block or as part of a mode of operation. Because a basic block cipher does nothing more than encrypt or decrypt a block of data based on a key, without any modifications, the same block of data would always be encrypted the same way, thus subjecting the cipher to cryptanalysis attacks. An initialization vector adds a factor of randomness to the overall algorithm.

Feedback is the process of the initialization vector and/or "cipher" block changing after each encryption or decryption operation. There are several different modes of feedback that are beyond the scope of this talk. However, it is important to understand that after the encryption operation, either the initialization vector or cipher block is modified. This operation implements stream-cipher-like capabilities into the block cipher algorithm.

The diagram depicts the process of encrypting data with a block cipher that uses an initialization vector and a feedback loop:

- An initialization vector is applied to a cipher block to generate an initial set of random or pseudorandom data.
- A plaintext message is partitioned into one or more blocks of data.
- The block of data is processed through the encryption algorithm generating encrypted text.
- Upon completion, the cipher block is updated based on the feedback or mode of operation determined by the user.
- The next block of data is processed through the algorithm and encrypted with the new cipher block.

Public Key Encryption

- Advantages
  - Private keys stay safe, public key shared
  - Provides authentication using same keys
- Disadvantages
  - More computationally intensive than symmetric key encryption
  - Susceptible to MITM attacks
- Typical use case
  - Use asymmetric to exchange key material
  - Use shared key material for symmetric

Asymmetric encryption systems are more commonly known as "public key encryption" systems.

Public key encryption is a system that uses two separate keys, one of which is kept secret and one of which is shared publicly. Although there are two different keys, they are based on a mathematical relationship such that one key can encrypt a message while the other key can decrypt the message. In public key encryption, the public key can be shared with other users, while the private key must remain secret. This is public key encryption's biggest strength—users do not need a secure means of exchanging key material before using the encryption system.

Another key strength is that these same keys can be used in the opposite direction to provide message authentication. That is, a digest (usually a hash of the message, along with some additional values) that is encrypted with a private key can only be decrypted by the corresponding public key. This provides the recipient with the assurance that the message was sent by a party with the private key material that is mathematically associated to the known public key.

However, there are several weaknesses associated with public key encryption. Due to the unique nature of the keys, they are more computationally costly than their counterparts within symmetric key encryption. Although longer keys reduce the likelihood of a brute force attack on the system, the longer key increases the computational costs associated with the encryption system. In addition, like any encryption system, public key encryption is also susceptible to Man-In-The-Middle (MITM) attacks, because all parties must still have a high degree of trust in the public key system. (This is why many public keys are signed by multiple parties, which increases the "web of trust" needed for proper cryptographic exchange.)

Common examples of public key encryption systems include SSL/TLS, SMIME, and PGP/GPG.

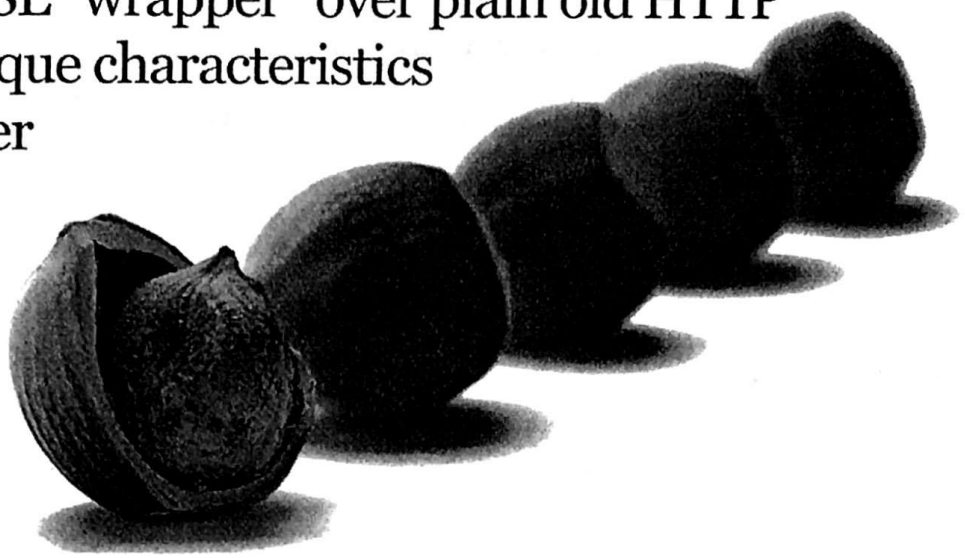
Symmetric and asymmetric encryption are frequently used together to mitigate the mathematical complexity of such a system. Both parties may use the more expensive asymmetric encryption to exchange a small message, without significant computational impact. The message that is exchanged in this phase becomes a symmetric key used for the remainder of the conversation.

The diagram here depicts an example of public key encryption:

- Alice sends Bob her public key.
- Bob generates a message and encrypts it with Alice's public key.
- Bob sends the encrypted message to Alice.
- Alice unencrypts Bob's message using her own private key.

HTTPS, in a Nutshell

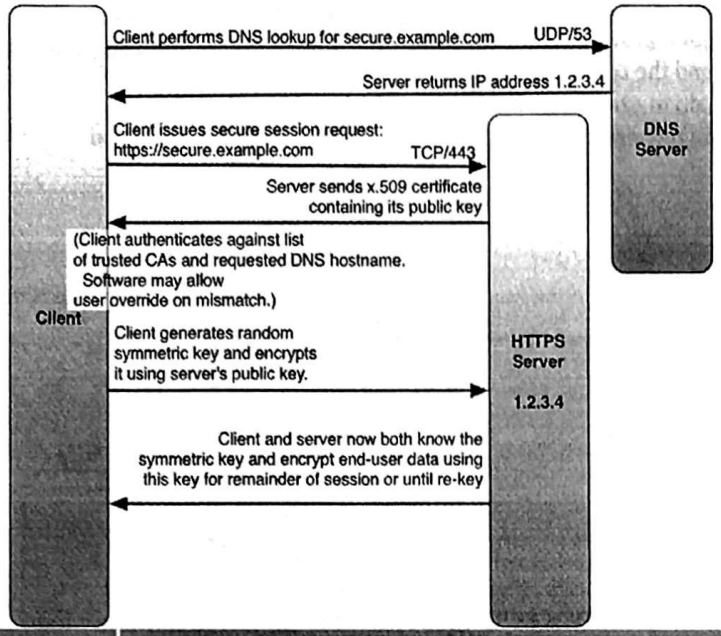
- Just an SSL "wrapper" over plain old HTTP
- Some unique characteristics to consider



We previously spent a good deal of time on HTTP itself—the good news is that for the most part, all of what we just covered also applies to encrypted HTTP. HTTPS is basically the same HTTP you know and love—just wrapped into an encrypted connection. However, by virtue of the encryption, it can't be easily examined without special tools and techniques. Despite that potentially work-stopping hurdle, there is still a good deal to learn from just external observation of an HTTPS connection.

In this module, we'll primarily focus on the differences in how HTTP behaves between typical unencrypted mode and encrypted HTTPS. We'll also briefly touch on some of the more useful artifacts that can still be captured from HTTPS communications—even if we can't observe the content itself.

Basic SSL Process



- Negotiation handshake followed by secure communication over encrypted channel
  - In-conversation renegotiation as well

After identifying the IP address for the intended hostname via DNS or a local cache, the client opens a TCP socket to the intended destination—in this case, what appears to be an HTTPS connection via TCP/443. The SSL exchange begins with a negotiation called a handshake followed by the client and server passing data between each other through the encrypted channel. The initial handshake must occur in plaintext, yielding some important data points for inspection, profiling, and analysis. Of course all communications after the connection is set up occur within the encrypted channel. Either party can request an SSL/TLS renegotiation at any time, at which point the initial process repeats. However, the renegotiation occurs over the existing secure channel, and is therefore not available for casual inspection.

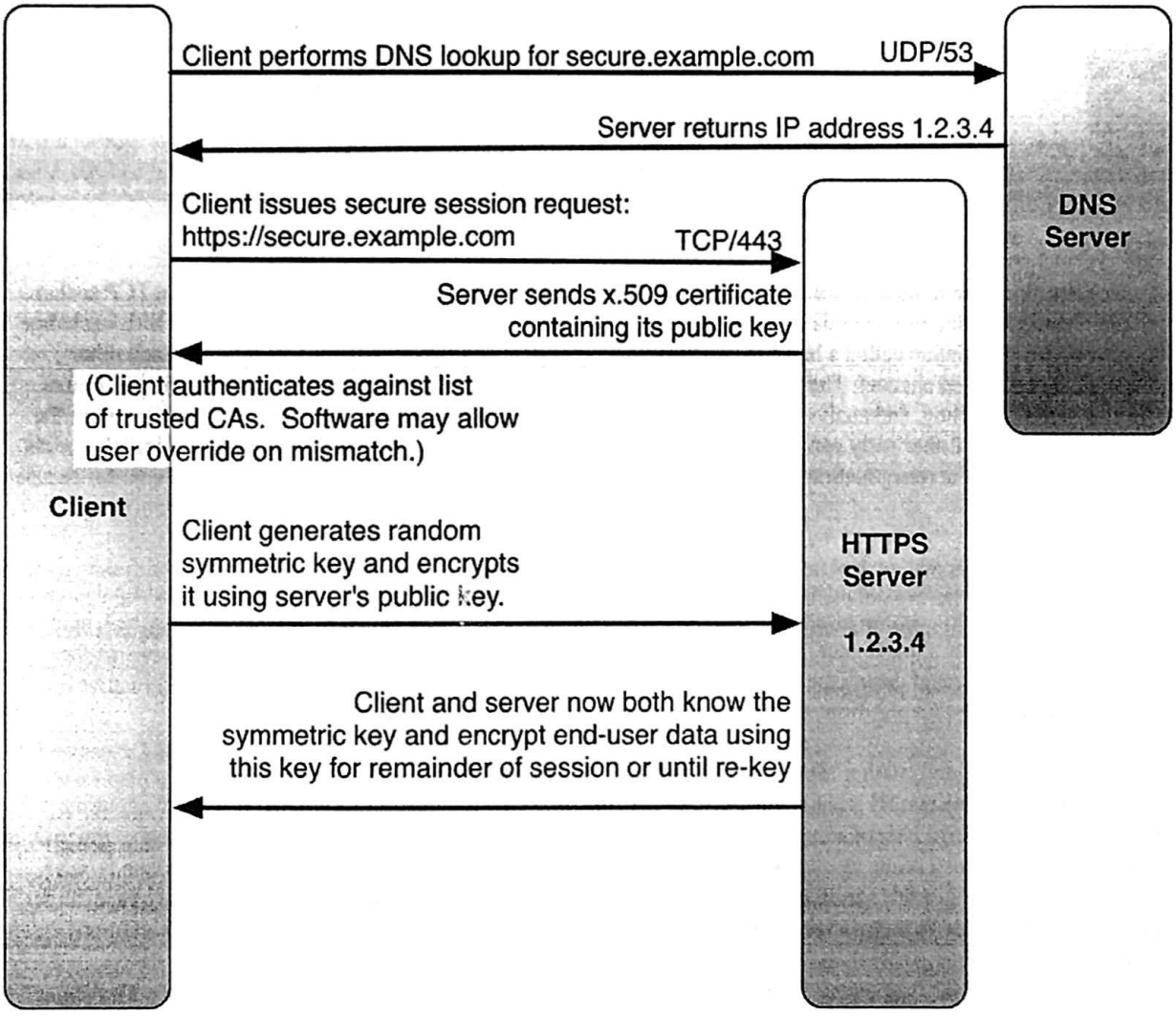
During the typical exchange depicted in the diagram, the client and server exchange a number of useful fields in the plaintext portion of the communication. The initial client request is called the "Client Hello" message. The Client Hello packet includes a list of encryption algorithms (called cipher suites) and compression mechanisms that the client is willing to accept during the handshake. Often, these values can indicate the client operating system or SSL libraries, because certain platforms offer a known set of cipher specs during this phase.

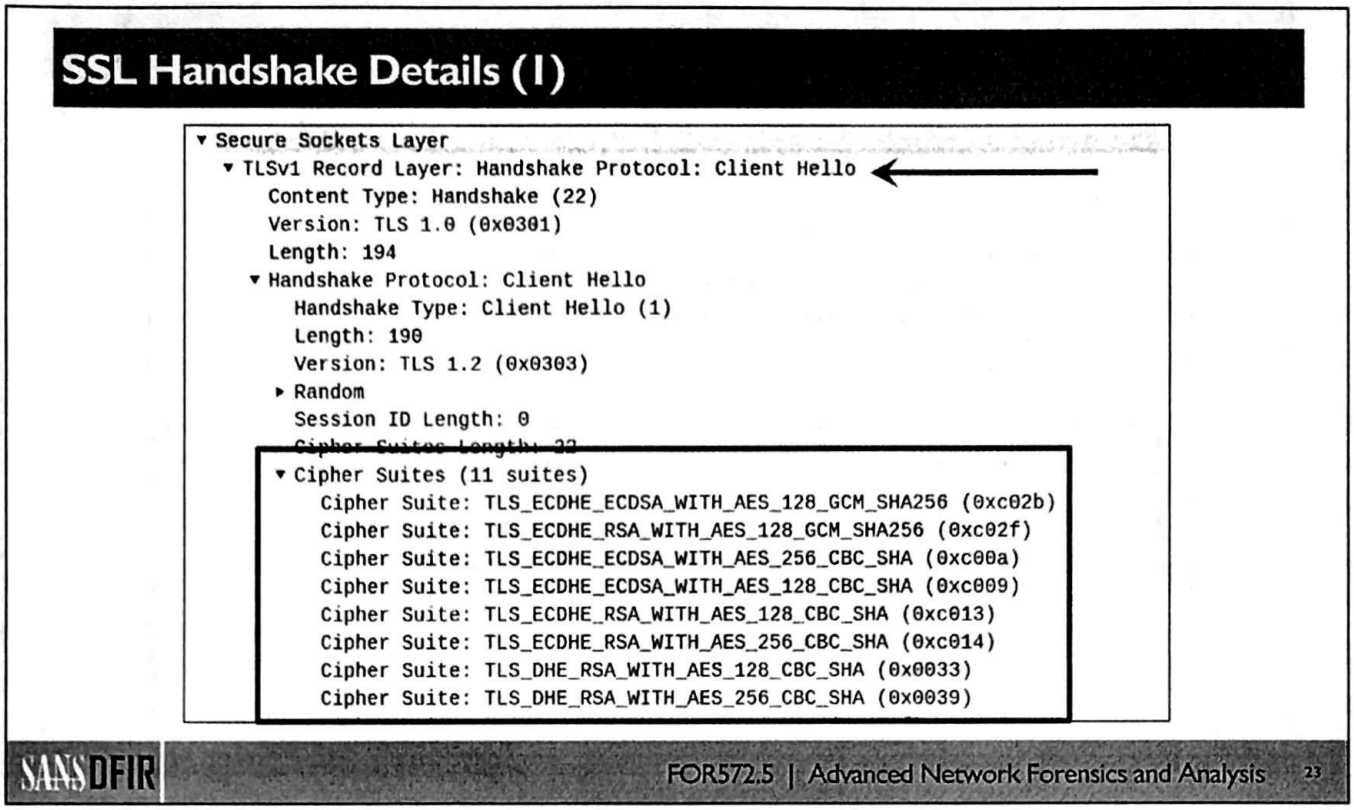
The server then responds with a "Server Hello" message and selects one of the client's supported cipher specs and compression mechanisms. By design, the server selects the most highly preferred cipher suite from the client list against its own preferred list of suites, resulting in the most preferred mutually usable cipher suite between the two parties. The server also sends its own SSL certificate. This certificate contains several very important data points: the "common name", or DNS hostname, the certificate is certifying is an important piece of information that can be used in conjunction with knowledge of the DNS request to ensure a match or isolate the appropriate NetFlow records.

The certificate may also include several parent certificates that have signed the server's own certificate. The client uses these values to validate the server's offered certificate against its own core list of "trusted" Certificate Authorities. This evaluation process also involves the client's confirmation that the common name being certified matches the hostname that it originally requested via DNS. Generally, if the client cannot validate the certificate at

this stage, it displays the familiar warning messages asking whether the user wants to accept the untrusted certificate. (Of course, users may very well just click "OK" to proceed anyway…)

Whether automatically or through the user's "manual override", after the certificate is validated, the client continues the negotiation using a public key encryption process. With this process completed, the connection becomes fully encrypted after a few more packets and the connection is considered opaque to the analyst. Once the connection is encrypted, there is little more we can do to examine the contents of the communication. Even subsequent re-keying requests, which either side can request arbitrarily, are accomplished within the encrypted channel.





Now we'll look at portions of the first few packets exchanged as they appear in Wireshark's packet details pane.

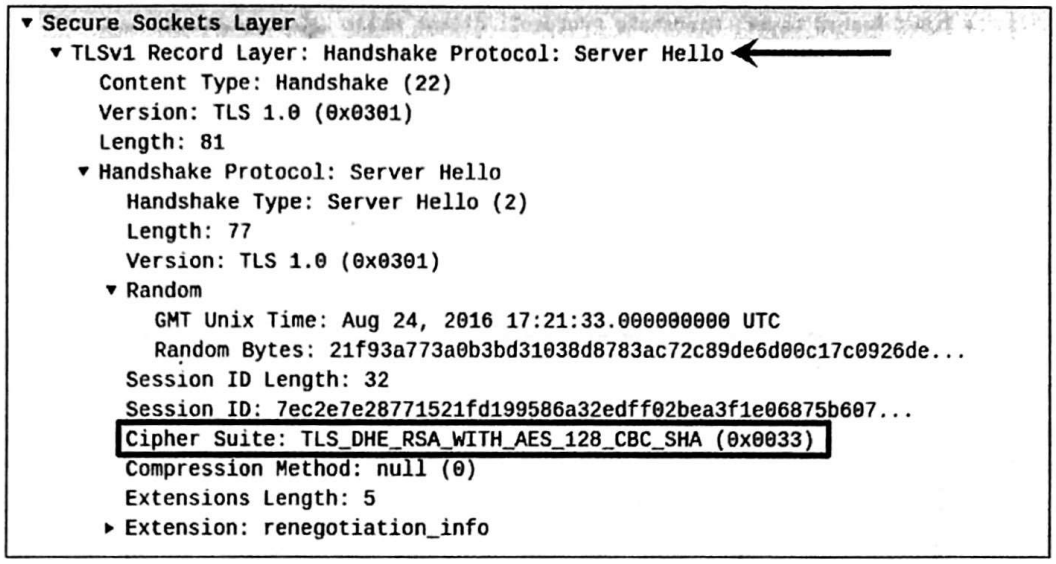
This screen shows the Client Hello message. We have framed the first few cipher suites the client is willing to support. As you will see in the next exercise, these can be used to profile the SSL library the client software uses in some cases, and it is often valuable to collect this list when examining suspect SSL connections. These values are passed in traffic as two-byte values as shown in parenthesis in the screenshot. The SSL RFCs[1][2] specify the meaning of each, and Wireshark displays this in the GUI. On the other hand, `tshark` only represents these as their 16-bit integer values.

Although not reflected in the screenshot, there is a similar list of supported compression algorithms and various SSL extensions in this message that can also be used to more specifically profile the client's SSL libraries.

References:

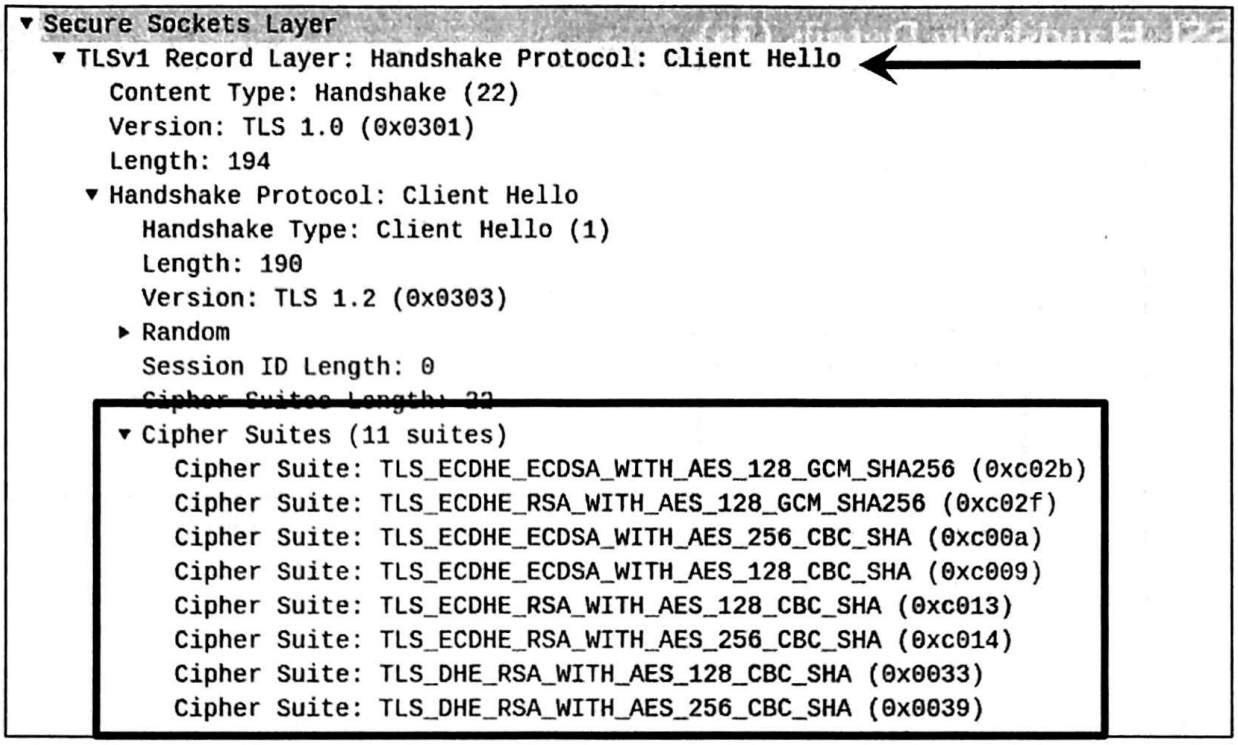
[1] `http://for572.com/jz0dt`
[2] `http://for572.com/3jusg`

SSL Handshake Details (2)

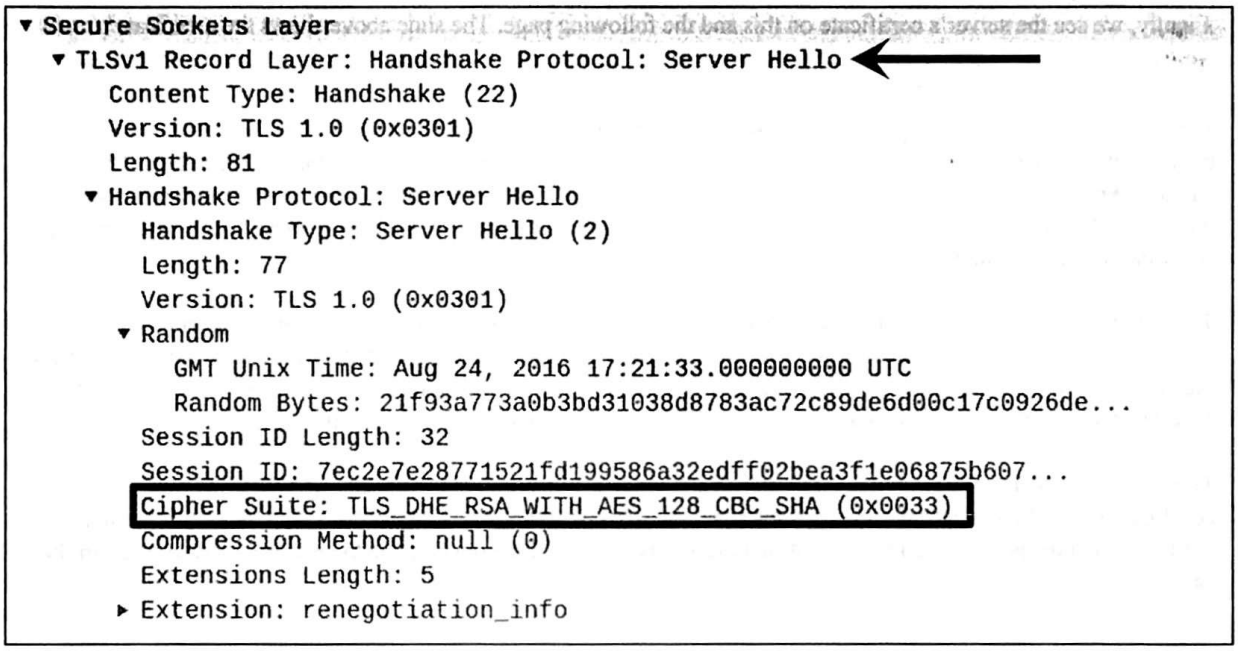


Here we see the server's reply, the Server Hello message. As we have highlighted, the server chose one of the client's supported ciphers, selecting the most preferable on its own list that the client offered in the previous message.

Also shown here is that the server has opted not to use any compression algorithm for the remainder of this handshake.

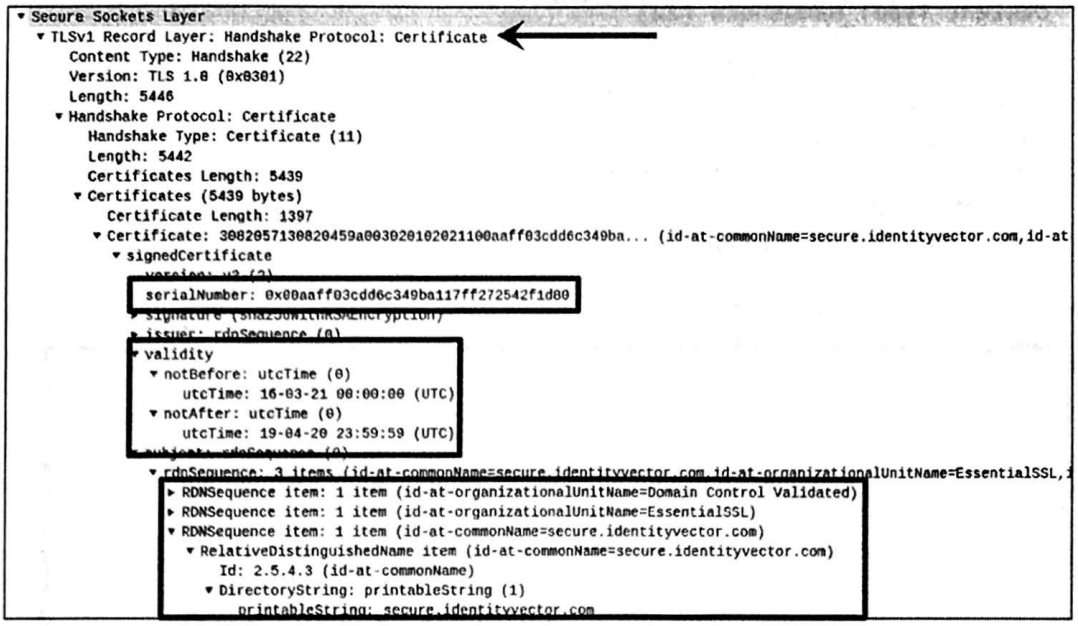


Handshake Details (1)



Handshake Details (2)

SSL Handshake Details (3a)



Finally, we see the server's certificate on this and the following page. The slide above shows the certificate's unique serial number, which we can use to associate encrypted communications that may be with many different servers.

We have also highlighted the period during which the certificate is considered valid. At some point, your web browser has displayed the warning that a certificate is no longer valid. That warning is the result of the browser's validation of these fields. This artifact may be of particular interest when evaluating the adversaries' actions within the scope of your investigation. For a custom certificate used in a campaign against only one environment, this may show the overall timeline of the attack.

The data framed at the bottom of the slide is the subject or "common name" of the certificate—a host named "`secure.identityvector.com`". This string may appear in any of several different fields, depending on how the certificate was constructed and who issued it. The analyst may need to walk through the contents of the ASN.1-encoded packet details in order to determine what field names were used for this specific certificate.

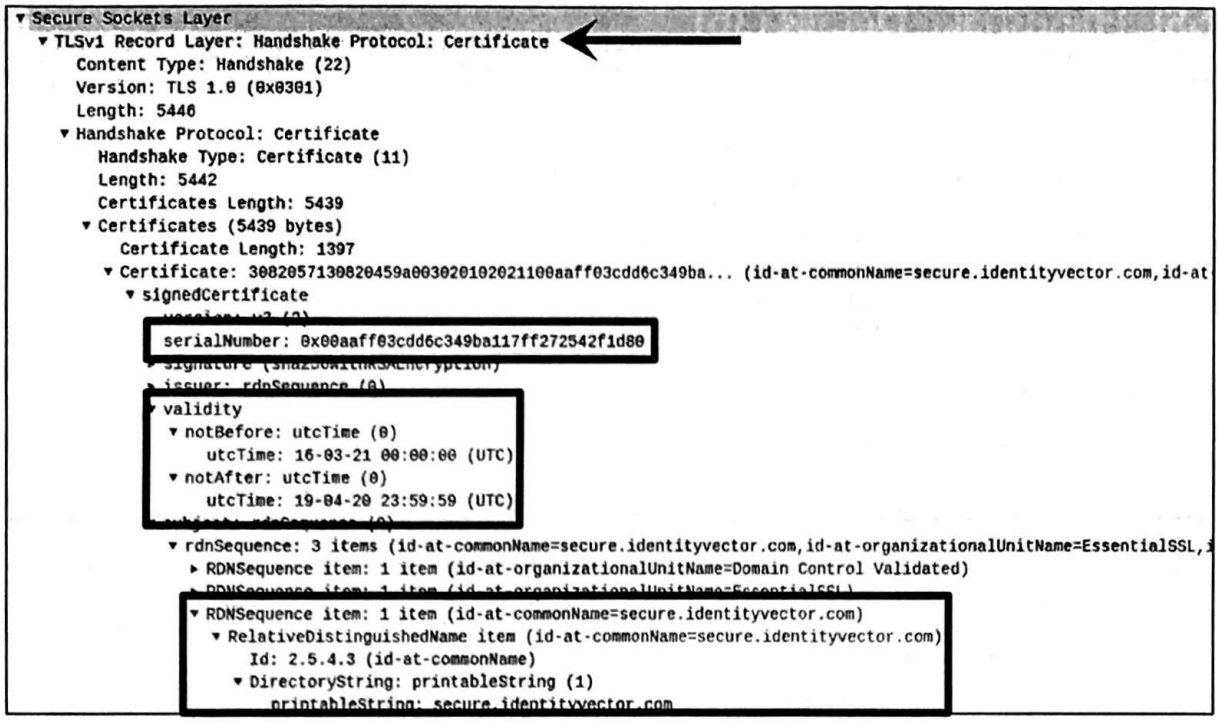
There are some additional strings in this certificate that further identify its useful parameters. In this case, the certificate was validated based only on domain ownership, rather than an "Extended Validation" certificate that requires much deeper ownership verification before it is issued. "Essential SSL" is the issuer's product name in this case.

SSL Handshake Details (3b)

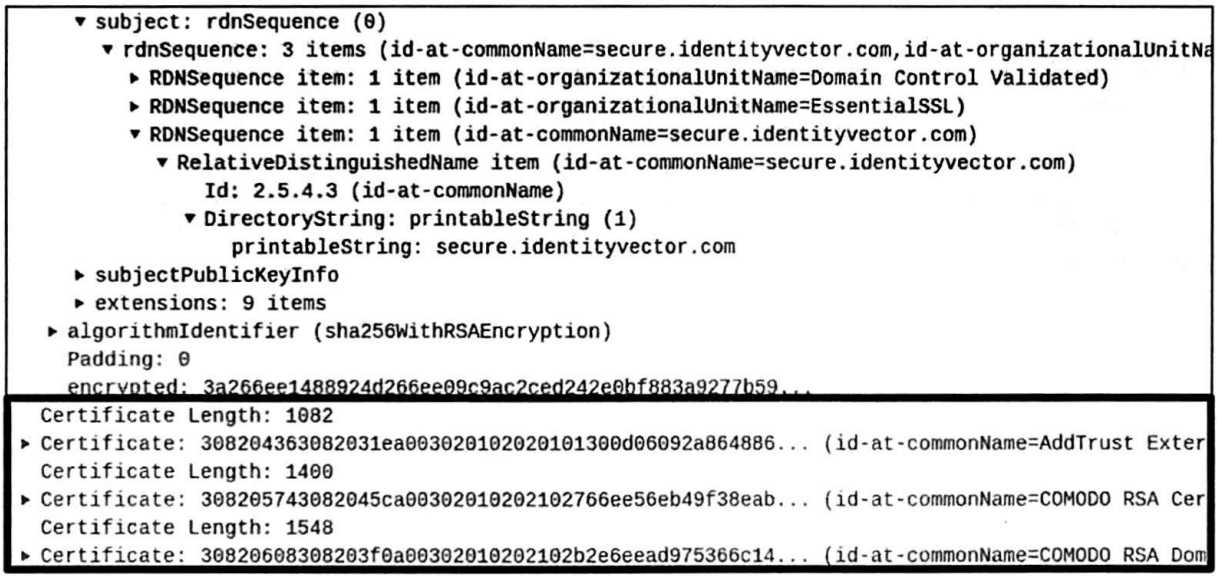
```
    ▾ subject: rdnSequence (0)
      ▾ rdnSequence: 3 items (id-at-commonName=secure.identityvector.com,id-at-organizationalUnitNa
        ▸ RDNSequence item: 1 item (id-at-organizationalUnitName=Domain Control Validated)
        ▸ RDNSequence item: 1 item (id-at-organizationalUnitName=EssentialSSL)
        ▾ RDNSequence item: 1 item (id-at-commonName=secure.identityvector.com)
          ▾ RelativeDistinguishedName item (id-at-commonName=secure.identityvector.com)
              Id: 2.5.4.3 (id-at-commonName)
            ▾ DirectoryString: printableString (1)
                printableString: secure.identityvector.com
    ▸ subjectPublicKeyInfo
    ▸ extensions: 9 items
  ▸ algorithmIdentifier (sha256WithRSAEncryption)
    Padding: 0
    encrypted: 3a266ee1488924d266ee09c9ac2ced242e0bf883a9277b59...
    Certificate Length: 1082
  ▸ Certificate: 308204363082031ea00302010202010300d06092a864886... (id-at-commonName=AddTrust Exter
    Certificate Length: 1400
  ▸ Certificate: 308205743082045ca00302010202102766ee56eb49f38eab... (id-at-commonName=COMODO RSA Cer
    Certificate Length: 1548
  ▸ Certificate: 30820608308203f0a00302010202102b2e6eead975366c14... (id-at-commonName=COMODO RSA Dom
```


In addition to the certificate for the "`secure.identityvector.com`" host itself, this message includes three additional certificates. These are certificates for Certificate Authorities, or CAs, that have assured the validity of the "`secure.identityvector.com`" hostname itself.

These are called "intermediate certificates", or "chained certificates". It is not uncommon for multiple chained certificates to be delivered with a server's own certificate because these are used to a core certificate that the client trusts in its root CA store.



Handshake Details (3a)



Handshake Details (3b)

HTTPS "Gotcha"

- "Keep-Alive" connections allow multiple request/response pairs over a single socket
  - Provided by HTTP itself (not HTTPS-specific)
  - Complicates byte counts over session lifetime
- Consider a 156,267-byte HTTPS connection observed in NetFlow evidence
  - Single file, approximately 152KB?
  - Two files, approximately 76KB each?
  - Some other combination?

When analyzing an encrypted communications channel, one of the useful pieces of metadata is the amount of data transferred. This methodology might be useful in estimating the amount of data an attacker has stolen from a victim's network, for example. However, remember that the underlying protocol in this case, HTTP, provides the "Keep-Alive" capability, which allows multiple request/response exchanges in a single socket. This feature was first devised to improve efficiency with regard to normal TCP handshaking—saving a few dozen bytes per connection. You can imagine that there is a significant improvement with SSL, which requires hundreds or thousands of bytes to establish the connection!

Although it certainly improves efficiency, it complicates volume analysis because there is no way to identify how many exchanges were conducted over a single SSL session, nor the amount of data for each exchange.

Differences from HTTP (1)

- Name-based virtual hosting and transparent proxying are both challenging
  - Allows many hostnames to share one IP address
  - "Chicken and egg" problem: Can't send "`Host:`" header until secured, no SSL without telling server the hostname
- Exceptions
  - Wildcard certificates "*.example.com"
  - RFC6066: Server Name Indication
- Dynamic certificate generation possible
  - Explicit proxy setting or "Bump-Server-First" trick

The nature of the encapsulating SSL protocol presents some other challenges that cause the enclosed HTTP traffic to diverge from its typical behavior when unencrypted. Understanding these differences can be extremely useful in finding investigative value in the aspects of HTTPS traffic that we can still observe without decrypting the contents.

Possibly the most pervasive difference is that "name-based virtual hosting" is not generally available for HTTPS connections. Name-based virtual hosting allows an HTTP server to provide content for hundreds or even thousands of web server hostnames using a single IP address on the server. This is possible because the HTTP/1.1 protocol specification requires that the client include the hostname from which they are requesting a resource as an HTTP request header named "Host". An example is below:

```
GET /funny_cat_video_1986120463.flv HTTP/1.1
Host: www.youtube.com
...
...
```

However, to perform an SSL handshake, the server must know the hostname the client is requesting, so it can send the appropriate certificate. The HTTP 1.1 specification states that the hostname is to be included as a header item, which is only passed after the SSL handshake has been completed. This classic "Catch 22" has historically required that a server use only one SSL certificate per listening port per IP address. From our perspective, this is actually quite helpful! It means that if we examine the certificate provided during that SSL handshake exchange—from captured traffic or by (appropriately) interrogating the server—we can identify the hostname the client likely requested. This information can be a useful lead in searching out similar behavior on other clients, mining DNS query logs, and more.

There are a few exceptions to the "one-to-one" rule, however. Perhaps the most commonly seen example is that of "wildcard" certificates. These are granted on a domain-wide or subdomain-wide basis, so named because the certificate's subject starts with an asterisk. These are often seen with content distribution networks or platforms, as well as with private Certificate Authorities used in corporate environments.

A relatively new exception is the January 2011 RFC6066. This RFC broadly defines a series of extensions to the latest iterations of the TLS protocol (itself a "child of SSL"). Among the various extensions it contains is the Server Name Indication, or SNI. When using the SNI extension, the client includes the hostname of the server it is attempting to contact as a part of the pre-encrypted SSL handshake process. By having this data available, the server can determine the correct certificate to present for each new SSL session—hence, SSL name-based virtual hosting becomes possible. Although all modern servers and most modern browsers support this extension, adoption has been slow due to the pervasiveness of antiquated software (cough, cough… Windows XP… and some other old vampire code clients and libraries).

There is a related difference between behavior of HTTP and HTTPS traffic when it comes to transparent proxying of web traffic. If you recall our walk-through from earlier in the class, transparent proxying takes place when a client's web requests are diverted to a proxy server by means of a NATting firewall or similar routing device. This enforces proxying even for devices or applications that are not configured with or aware of the environment's proxy settings.

When the client's outgoing request is unencrypted (i.e., straight HTTP), the proxy can freely inspect the request to broker the exchange between the client and the intended server. However, because the interception occurs at layer four (TCP traffic is redirected), the proxy is unaware of the DNS hostname the client requested. (The client resolved this in a separate DNS transaction.) Similarly, the proxy cannot inspect the content of the request to find the HTTP "Host" header, as the secure socket must be established before the HTTP exchange can commence. Establishing the secure socket requires knowledge of the hostname for certificate validation, hence the paradox. This makes sense, because from the client's network perspective, the proxy server is providing all of the client's content from a daemon on a single IP and port—identical to the name-based virtual hosting problem.

Some advanced proxy configurations can dynamically generate SSL certificates for connections—if, of course, they can determine the hostname the client is requesting before the SSL handshake continues. Version 3.2 of the Squid proxy introduced the Certificate Daemon, which accomplishes this internally. There are external solutions for Squid as well, and commercial proxy solutions such as BlueCoat often provide this as well. There are currently two primary methods of proxying HTTPS traffic:

- Although not functional in a purely transparent model, a client that is given an explicit proxy setting for HTTPS requests uses the "CONNECT" method to request that the proxy retrieve a particular resource via HTTPS. The client then connects to the proxy to request the HTTPS resource. At this point, the proxy initiates the HTTPS connection with the intended destination and brokers the remainder of the connection. In this model, the proxy is aware of the hostname and port of the client's requests, but not the full URL, query string, headers, or content! The proxy acts as a tunneling NAT device in this configuration. However, when operating in this mode, the proxy can optionally use dynamic certificate generation to perform full content inspection.
- Another less-established but promising method is known as the "bump-server-first" method that may work in a pure transparent proxy deployment. The proxy, upon receiving a new connection to a designated SSL-wrapped port like HTTPS, holds the client's request while initiating a new SSL connection to the intended destination server. The proxy inspects the certificate offered during this secondary SSL handshake to determine the certificate's subject name (hostname) and other details. The proxy server can then dynamically generate its own certificate to match that from the intended server, providing it to the original client on its held request. This feature is available in Squid version 3.3 and above, and may be available in other solutions as well.

Does this sound a little bit shady? It should—this is commonly called a "Man In The Middle" or MITM attack. As such, it is important to note that intercepting any SSL traffic may be illegal in your jurisdiction, company, or network environment. Consult with legal counsel before experimenting with a solution like this!

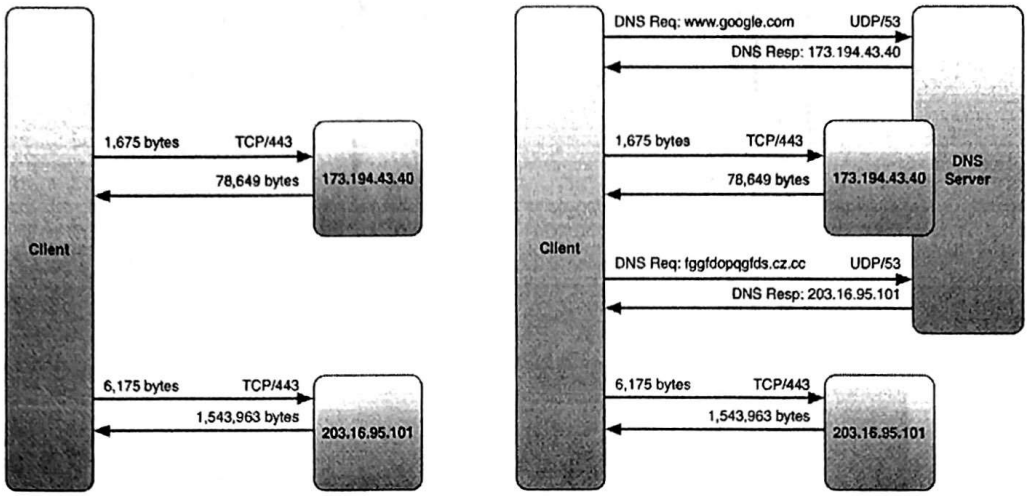
It should also be noted that to do this, the client systems must be configured to trust the certificates that the proxy dynamically generates. We will discuss this more in the Encryption section of the course, but in any environment

where the clients' system configuration can be controlled, it's trivial to accomplish. It is also worth mentioning that at the time this material was written, there was no solution to leverage the Server Name Indication extension for dynamic certificate generation.

References:
`http://for572.com/8e2xh`

Analytic Mitigation: Supplement NetFlow with DNS Correlation

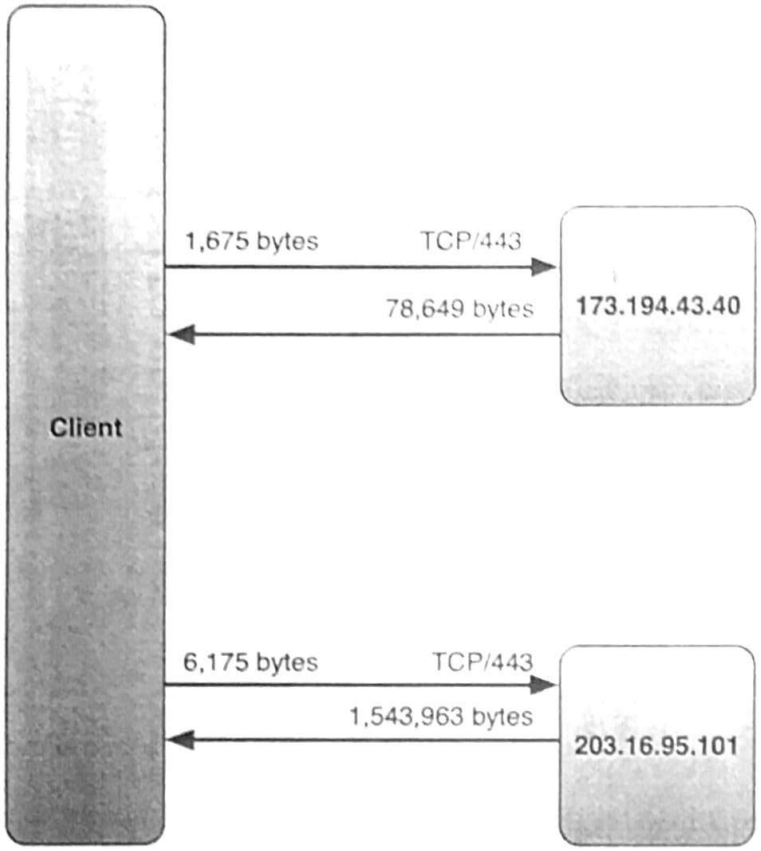
- Hostname needed to validate certificate

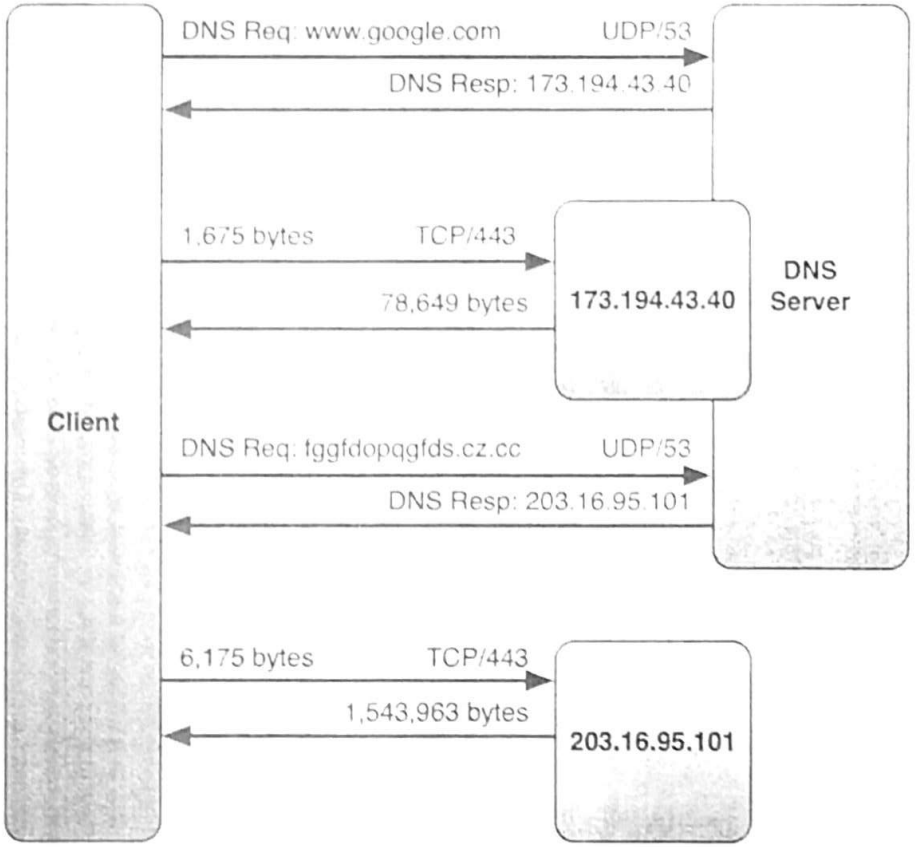


With the exception of the first few packets in the SSL negotiation, the remainder of the SSL traffic will be functionally opaque. For this reason, NetFlow is an exceptionally valuable way to characterize the connection without incurring the storage overhead of a full-packet capture system. However, even without the full visibility to the SSL negotiation process or more complex SSL interception scenarios, there are several ways that a network forensicator can mitigate the impact of SSL on his or her ability to conduct a thorough investigation.

Consider the sequence of events that occurs when a user hits the "Return" key after typing "`https://example.com`" in their browser. The system first performs a DNS lookup to get the proper IP address for the "`example.com`" hostname, then initiates a connection to that IP on port 443 (by default). If we have any insight into the client's DNS activity immediately before it initiates the HTTPS connection, we could learn the host they are attempting to connect with, better characterizing evidence from non-content sources such as NetFlow.

This is no guarantee, of course, because the server could provide a certificate with a subject name that does not match what was expected from the DNS request, but in most cases, this will be a very useful way to combine different sources of evidence. If the user or client software ignores the mismatch and failed certificate validation, correlating the DNS activity to a particular certificate would be much more difficult.





Analytic Mitigation: Certificate Extraction and Field Listing

- Capture certificates, examine to find CNs
  - Can help characterize inaccessible content
  - Still benefits from DNS correlation

```
$ tshark -V -n -r ssl.pcap -Y 'ssl.handshake.certificates'
...
RelativeDistinguishedName item (id-at-commonName=secure.identityvector.com)
...

$ tshark -V -n -r ssl.pcap -Y 'ssl.handshake.certificates' -T fields ⏎
  -e ip.src -e ssl.handshake.certificate |tr ',' '\n'
205.186.148.46    30:82:05:71:30:82:04:59:a0:03:02:01:02:02:11:00:aa:ff:03:cd:d6:c3:49:ba:11:7f:f2...
30:82:04:36:30:82:03:1e:a0:03:02:01:02:02:01:01:30:0d:06:09:2a:86:48:86:f7:0d:01...
30:82:05:74:30:82:04:5c:a0:03:02:01:02:02:10:27:66:ee:56:eb:49:f3:8e:ab:d7:70:a2...
30:82:06:08:30:82:03:f0:a0:03:02:01:02:02:10:2b:2e:6e:ea:d9:75:36:6c:14:8a:6e:db...
```

- NetworkMiner automatically extracts SSL certificates

Another method that may be useful in addressing the hurdle of SSL traffic would be to write out the certificate details to a text file as they cross an observation point on the network. Because the certificate is provided in the clear during the SSL handshake sequence, we can extract the subject names contained in each. This would establish what could be described as a "certificate log", which could be a valuable tool for later correlation.

Of course, because the subject of each certificate is a hostname, DNS query logs would be invaluable to perform this correlation, but even with the timestamp, source/destination IPs/ports, and a few key certificate fields, we could efficiently characterize the nature of otherwise opaque HTTPS traffic.

Because Wireshark parses certificates' ASN.1 encoding to the lowest possible level, the first `tshark` command above can be used to get these parsed and decode field contents, but this will include a great deal of additional field data. The second `tshark` command will write just the hex-encoded bytes for each certificate, one certificate per line. A script could quickly be written to post-process each line into a file for further examination or signature development.

Another interesting approach to this problem, involving a moderate level of awk scripting is available on the serverfault.com thread linked below.[1]

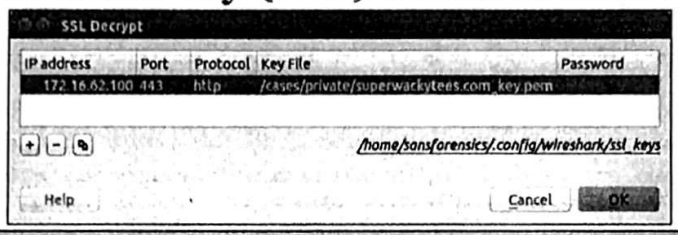
The Windows-based NetworkMiner[2] tool also performs certificate extraction.

References:

[1] `http://for572.com/2tmhj`
[2] `http://for572.com/dlakp`

Analytic Mitigation: Wireshark (1)

- Wireshark can decrypt some captured SSL traffic
  - Requires private key material
  - Only non-Perfect Forward Secrecy (PFS) communications
- `tshark` provides same functionality at command line




```
$ tshark -n -r tcp443.pcap -Y http ⏎
  -o ssl.keys_list:"172.16.43.100","443","http","/cases/private/superwackytees.com_key.pem"
4395 598.304188 172.16.62.42 -> 63.141.200.110 HTTP 270 POST /fcs/ident2 HTTP/1.1
4468 598.328955 63.141.200.110 -> 172.16.62.42 HTTP 218 HTTP/1.1 200 OK  (text/plain)
4506 598.349719 172.16.62.42 -> 63.141.200.110 HTTP 261 POST /open/1 HTTP/1.1
4507 598.372903 63.141.200.110 -> 172.16.62.42 HTTP 263 HTTP/1.1 200 OK
...
```


Fortunately for us, Wireshark also provides inline SSL decryption functionality that can be used when three technical conditions are met:

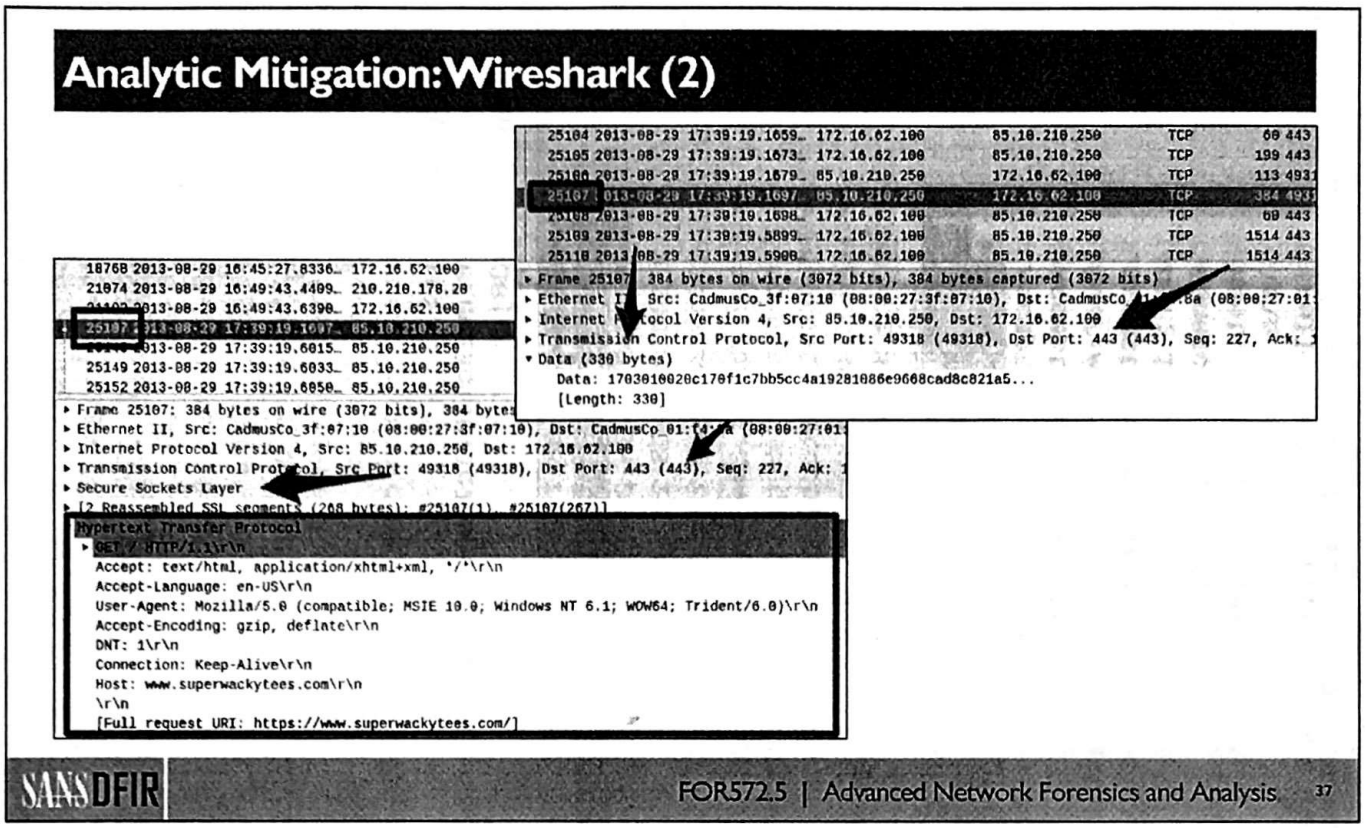
- You have a full packet capture for the entire session.
- You have the private key material for the connection (and the passphrase for it if needed).
- The connection does not use Perfect Forward Secrecy (PFS), which will be discussed in greater detail in a moment.

To enable this feature, use the SSL protocol preferences dialog to specify the IP address, port, underlying application protocol, and full path to the key material for each IP/port pair. After this configuration is applied, Wireshark transparently decrypts and parses the contents of the session and its underlying protocol, allowing the common and convenient display filters, protocol inspection features, etc. These settings also persist across Wireshark sessions, as they are stored in a preference file on disk.

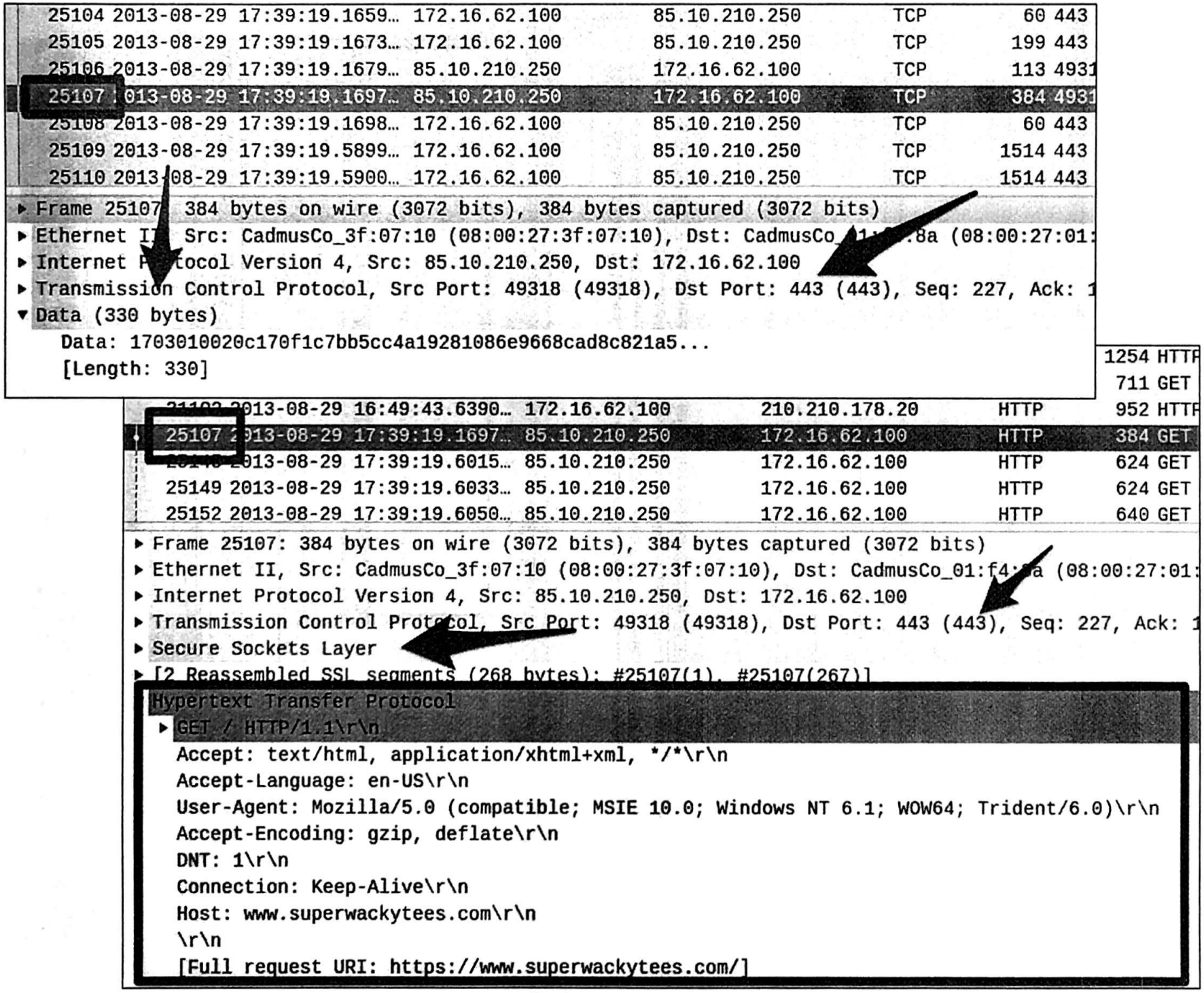
Unsurprisingly, tshark provides the same decryption functionality, but in the shell. There are two ways to enable this feature:

- Directly on the command-line as shown in the slide above, using the "`-o ssl.keys_list`" parameter
- Transparently by using the Wireshark GUI to set the keys as in the screenshot, then running `tshark` without the "`-o ssl.keys_list`" parameter. This option is possible because tshark uses Wireshark's configuration files, including any configured SSL keys.

References:
`http://for572.com/p8jdo`



These screenshots show a "before' and "after" of Wireshark's SSL decryption. In the image on the right, frame 25107 is a part of a TCP/443 connection with effectively opaque data as a part of an SSL stream (previously established). However, the screenshot on the left shows that Wireshark is now able to parse the same frame into its various SSL and HTTP components, despite still being a part of the same TCP/443 connection.



Perfect Forward Secrecy (PFS)

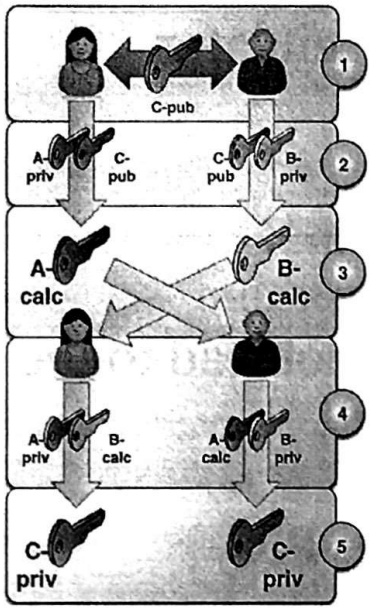
- Any private key compromise may permit later decryption of captured messages
- Public-key system provides PFS by:
  - Generating a random public key per session
  - Keys generated with non-deterministic algorithm
- Compromise of one PFS message cannot lead to the compromise of others
  - Encryption key is truly ephemeral
- Diffie-Hellman key exchange provides PFS

As explained previously, one risk associated with public key cryptography is that any compromise of the private key material will result in potential loss of all messages it previously encrypted. Given the undisputed trend toward use of more resilient encryption on the modern Internet, cryptographers have implemented a mitigation to this weakness called Perfect Forward Secrecy, or PFS. PFS is a means by which even if the key material from both parties were later acquired, their communications would still remain protected and could not be decrypted.

In a PFS connection, each party generates random public keys for the purpose of key agreement—without relying on a deterministic algorithm to do so. This means that the compromise of one message cannot lead to the compromise of others, and also that there is no single secret value (such as the system's private key) that would lead to the compromise of multiple messages. This ultimately means that the session keys the server generates are truly ephemeral.

In other terms, parties to a PFS conversation use algorithms that establish a common private key without that key ever being passed over the wire. Perhaps the most commonly cited implementation of PFS is the Diffie-Hellman key exchange.

Diffie-Hellman Key Exchange



1. Agree on common, public base key
   - (Authenticated with server's pubkey)
2. Apply separate, ephemeral secret keys to the common public key
3. Exchange calculated values
4. Use own ephemeral secret key with other's calculated values
5. Independently derive common, ephemeral secret key

This diagram graphically shows the steps involved in a Diffie-Hellman Key Exchange. As you can see, the "`C-priv`" keys have been mutually calculated, without any private key material being passed across the untrusted public medium.

The steps of this key exchange, which provides perfect forward secrecy for the derived private key, is as detailed below:

1. Alice and Bob agree on a common, public base key.
2. Alice and Bob apply separate, ephemerally generated secret keys to the common public key, which results in two new calculated public values.
3. Alice and Bob publicly exchange the calculated values.
4. Alice and Bob use the same ephemeral secret keys against the exchanged calculated values to derive a common secret value without ever transmitting secret key material over the untrusted medium in either encrypted or unencrypted form.
5. Alice and Bob can use the independently calculated shared secret as needed—for example, as a symmetric encryption key.

Obviously, a PFS-protected communication will be a big hurdle to a network forensicator trying to investigate the connection, because even a later acquisition of the private key will not allow decryption. The only means of debugging these connections involves configuring one endpoint or the other to log the derived private key material to a file for later decryption processing.

For those who prefer a more mathematical approach to this process, the below steps correspond to the diagram with the equations used in each.

1. Agree on known `g`, `p`
2. Pick hidden `a`, `b`
3. $A = g^a \bmod p$
   $B = g^b \bmod p$
4. Exchange `A`, `B` values
5. $C = A^b \bmod p$
   $C = B^a \bmod p$

To give a little more detail on this, the initial "key" that Alice and Bob agree on is actually two numbers. One is a large prime "p" and the other is a value "g", which is a "primitive root modulo p" (also sometimes called a "generator"). The value "g" has the property that for any value V between 0 and p, there is some exponent x such that ($V = g^x \bmod p$). Generally, g and p are well known values that are used over and over again for a given PFS implementation.

Having agreed on g and p, Alice and Bob now each choose their own secret a and b values. Alice calculates a new value ($A = g^a \bmod b$), and Bill calculates ($B = g^b \bmod p$). They then exchange their A and B values with each other publicly.

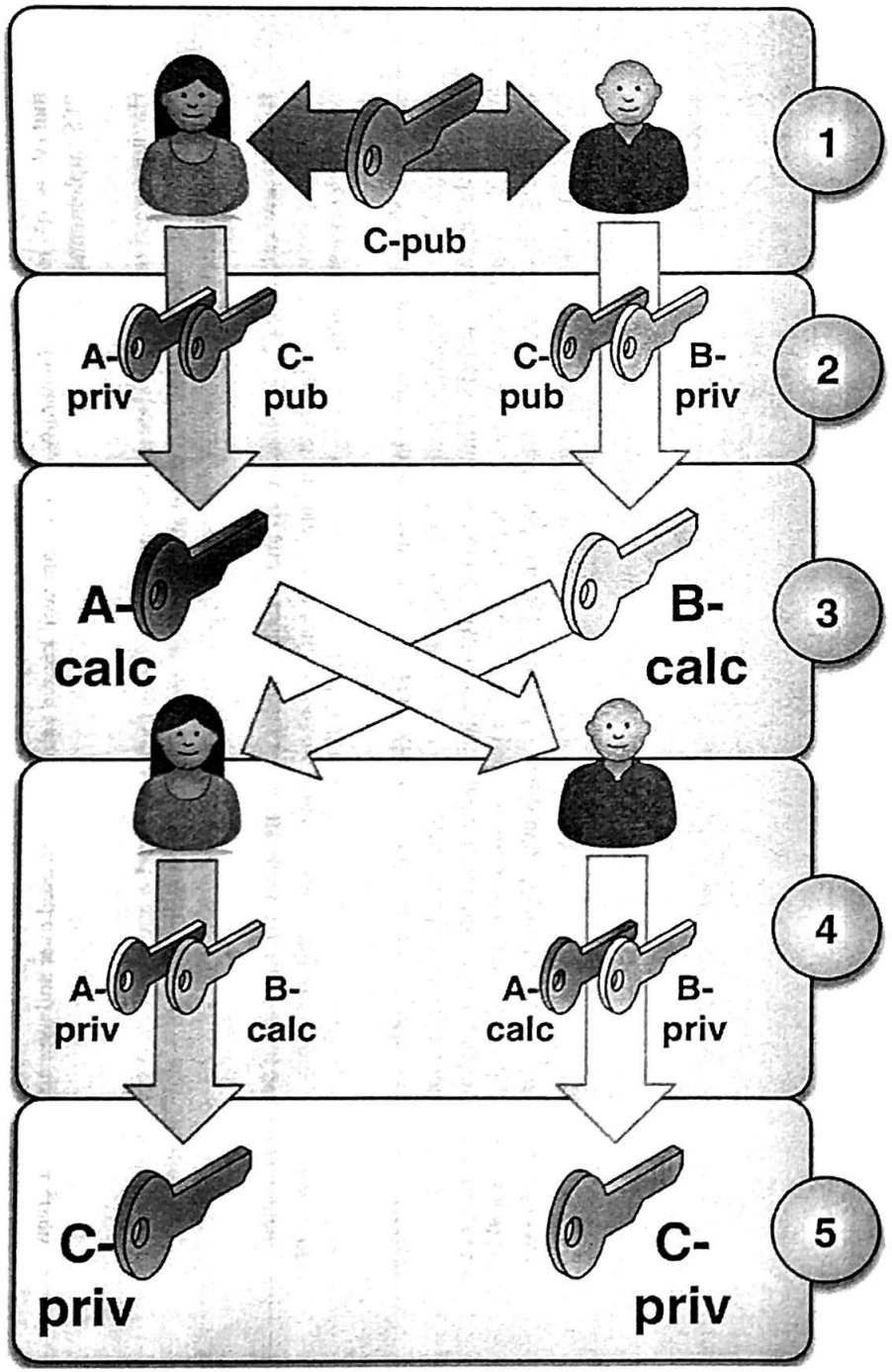
Bill can now calculate ($C = A^b \bmod p$) and Alice calculates ($C = B^a \bmod p$). Because of the mathematical relationship between g and p, both Bill and Alice end up with the same C value. Only somebody who knows a or b could calculate this value.

However, the only thing "secret" in this communication is the chosen a and b values. Figuring out a and b requires solving the "discrete logarithm problem" for which there is no known efficient algorithm ("computationally intractable" as the computer scientists like to say). However, an organization with heavy computing resources could pre-calculate much of the work to solve the discrete logarithm problem for a given pair of g and p. Once A and B are known, this pre-calculation effort would allow solving for a and b in perhaps a minute with modern computing power—fast enough to eavesdrop on the communication in near real-time.

Many thanks to Hal Pomeranz for taking the time to assemble this technical explanation!

References:

`http://for572.com/chapw`
`http://for572.com/jg21r`



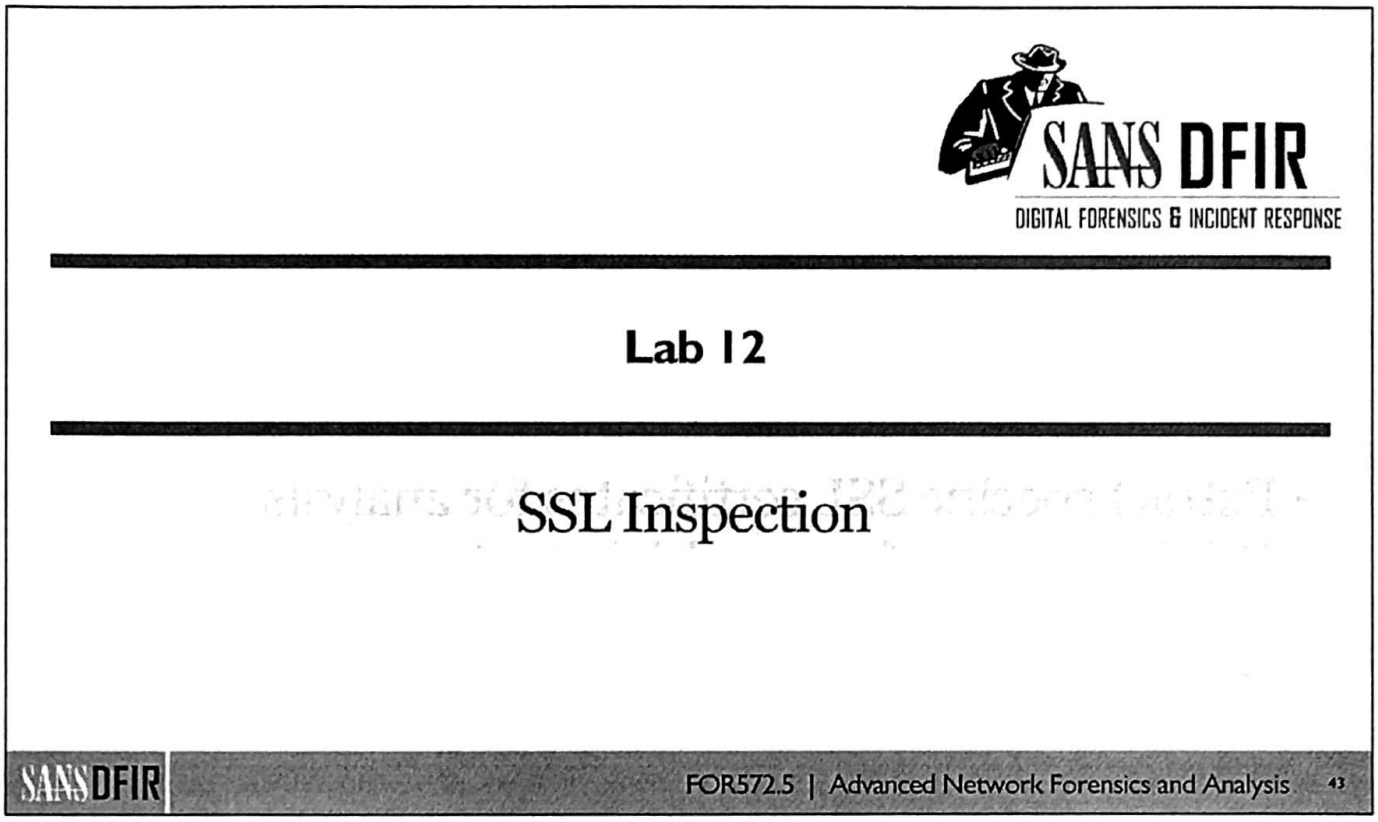
Step 1: Alice and Bob agree on a common, public base key

Step 2: Alice and Bob apply separate, ephemerally-generated secret keys to the common public key, which results in two new calculated public values

Step 3: Alice and Bob publicly exchange the calculated values

Step 4: Alice and Bob use the same ephemeral secret keys against the exchanged calculated values to derive a common secret value without ever transmitting secret key material over the untrusted medium in either encrypted or unencrypted form

Step 5: Alice and Bob can use the independently-calculated shared secret as needed - for example as a symmetric encryption key



This page intentionally left blank.

Lab 12 Objectives: SSL Inspection

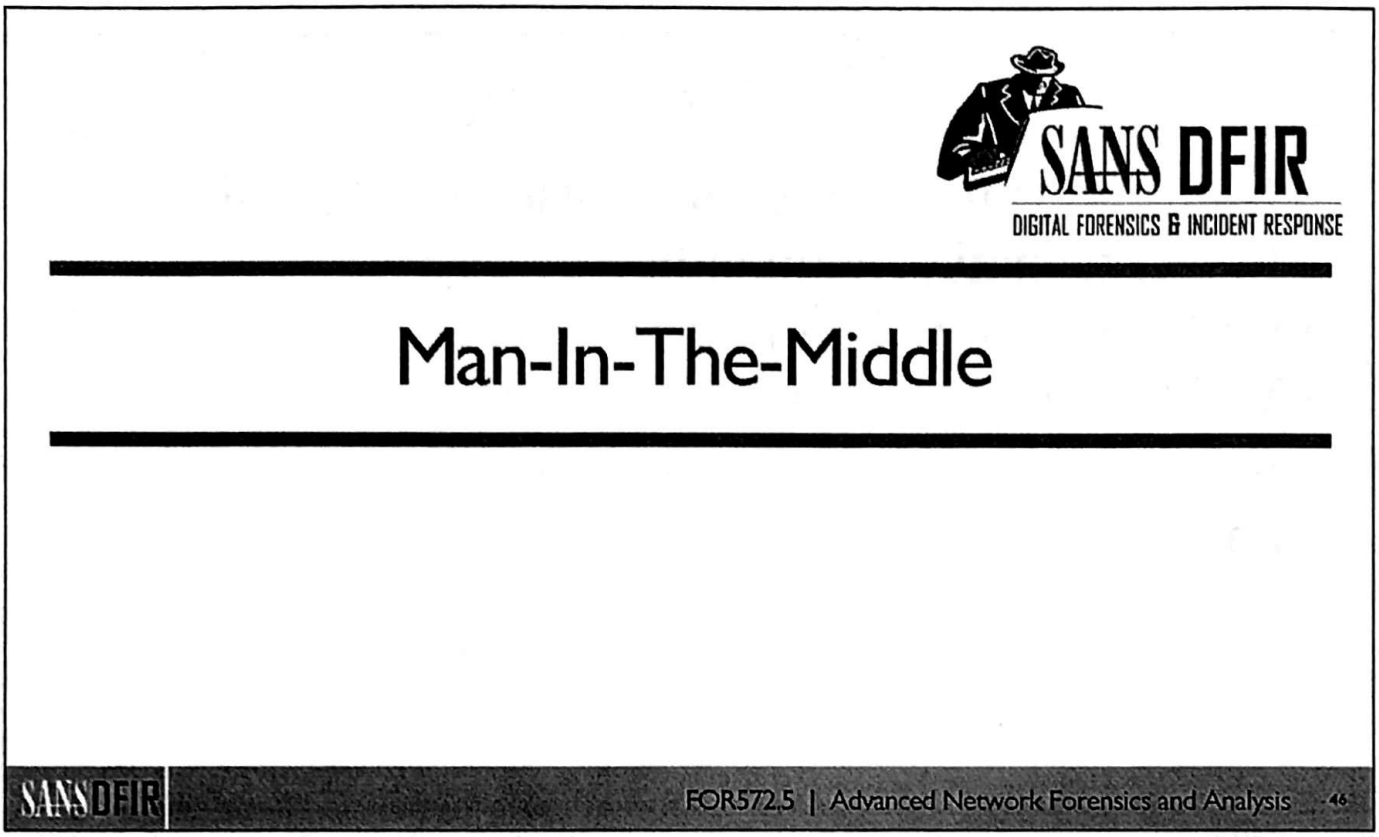
- Wireshark's parsers can reduce a large data set to a more manageable size
- Use pre-encryption phase of SSL traffic to establish profiles of SSL clients
- Extract certificate subject names
- Extract specific SSL certificates for analysis
- Validate output from multiple tools to ensure proper behavior

This page intentionally left blank.

Lab 12 Takeaways: SSL Inspection

- Even encrypted traffic can provide useful insight to the participating endpoints
- Certificates and their fields are useful IOCs
- Validating a new tool's results with established and manual processes is necessary to ensure accuracy and completeness

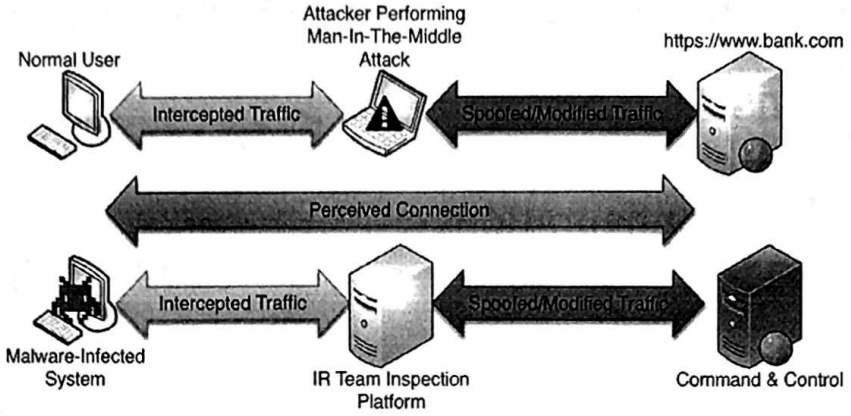
This page intentionally left blank.



This page intentionally left blank.

MITM: Dual Use Technique

- When a system sits between two others
  - Allows for decrypting, filtering, manipulation
  - Used against plaintext and encrypted communications



SSL inspection is a method by which an intermediary device sits between two systems (for example, an attacker and their target system or a user and an external SSL site). The interception/inspection device acts as both a client and server for any traffic that it encounters. This allows the device to decrypt the traffic and perform inspection of the content for malicious activity. To establish the perception of a successful connection, the device re-encrypts the traffic, typically using its own private key, and forwards the traffic to its final destination. This process could be applied to all manners of encrypted protocols, but is most conducive to those using asymmetric or public key encryption.

Of course the main shortfall associated with this architecture is that the attacker in the middle presents a certificate that does not validate in the client's software. While there are enterprise solutions (e.g. "for the good guys") that perform this, they rely on some level of control over the client system's trust database. A far more nefarious possibility would be if the attacker could leverage an existing certificate authority, then issue forged but "legitimate" certificates that didn't suffer validation failure. This is why the compromises of the Comodo and DigiNotar certificate authorities was so alarming for SSL users.[1]

As the Internet becomes more heavily and strongly encrypted, the use of SSL inspection devices will become more prevalent in many corporate environments. However, the use of such devices comes with technical and legal issues. From a technical standpoint, encryption and decryption are computationally expensive operations. Attempting to perform MITM at scale is a significant technical challenge. From a legal standpoint, legitimate websites (online shopping, banks, e-mail, etc.) are encrypted for a reason: they typically contain personal information that a normal user would not want revealed! When performing SSL inspection, an organization will likely have purview to their employees' personal data.

References:

[1] http://for572.com/lo65k

ARP Spoofing

- Floods LAN with spurious ARP replies so attacker's MAC address is associated with target's IP
  - More likely used to spoof the gateway IP address
- Victim(s) traffic framed to attacker's system
- Mitigations
  - Static ARP entries
  - OS-specific configurations

Of course the Address Resolution Protocol (ARP) is used to translate network layer addresses (Layer 3 IP addresses) to link layer addresses (Layer 2 MAC addresses). When an IP datagram is sent from one host to another on a local area network, the sending host must determine the MAC address for the recipient system before the Layer 2 framing header can be applied. When the MAC address for that destination IP address is unknown to the sending system, it must discover that MAC address using ARP. The sending host will broadcast an "ARP request" packet and the destination machine will respond with an "ARP reply" that maps the IP address to the MAC address. The sending system will retain this pairing result in its own local ARP cache for a period of time, so the request/reply processes don't overwhelm the local network segment.

The weakness with ARP is that it is a stateless protocol with no means to validate the ARP replies. For efficiency, network devices cache ARP replies regardless of whether they actually requested them. These weaknesses lead to a MITM attack known as ARP spoofing.

ARP spoofing is the process by which an attacker with access to the Local Area Network segment floods the LAN with forged ARP replies associating a target's IP address with the attacker's MAC address. The systems on the local network segment will see and cache this malicious association for efficiency, then frame any packets destined for that target to the attacker's system instead. Typically, an attacker will use this method against a default gateway, but ARP spoofing can also be done on a per-host basis. When the traffic is redirected to the attacker's host, the attacker has the option of inspecting, manipulating or denying traffic to/from the destination system. If the attacker carries this attack method out properly, the victim(s) will not even know their traffic is being intercepted, as the attacker can simply act as a router and pass the traffic along its intended path—after potentially inspecting, capturing, or manipulating the contents.

A number of defenses are available to mitigate the risk of ARP Spoofing, including implementing static ARP entries and configuration settings for various operating systems and network devices which cache only the replies for which the system sends a request for. While effective, these are often labor-intensive.

Port Stealing

- Attacker manipulates a switch into receiving traffic that is destined to a victim system
  - Attacker sends fake Ethernet frames with the victim's MAC address
  - With enough data, the switch binds the attacker's machine to the victim's MAC address
- Mitigations
  - "Enterprise" grade switch with "port security"

"Port Stealing" is a MITM attack in which the attacker's system is successful in manipulating a network switch into diverting traffic originally destined for a victim's system to their own. This is accomplished through a common vulnerability within Layer 2 Ethernet switches. A layer 2 Ethernet switch can learn what hardware is connected to each of its ports by listening to traffic sent to the switch port. As soon as a system sends any Ethernet frame to the switch, the switch caches this information into its internal state table (called a CAM table). Later, when the switch is making a determination on where to send a newly-arrived packet, it inspects the Ethernet header to identify the destination MAC address. If the packet is destined for a computer that is cached in the CAM table, it is sent only to the port to which the destination system is connected.

However, if an attacker is connected to another port on the switch, they can send fake Ethernet frames with the victim's MAC address forged into the source address field, essentially impersonating the victim machine. As an attacker sends data, the switch becomes confused and repeatedly alters the MAC address binding to either of the two ports by referencing the information within the packets. If the attacker sends data faster, the switch will send the attacker the packets intended to the victim host. When the traffic is redirected to the attacker's host, the attacker has the option of inspecting, manipulating, or denying traffic to/from the destination system.

This method generally incurs a bit of collateral damage, however. It results in the victim seeing intermittent connectivity or being booted off the network entirely.

This attack is most likely to occur on smaller networks as most enterprise grade switches offer "port security", which essentially locks a given MAC address to a single port at a time after it is first observed. Port security can also be enforced on some switch platforms by an administrator manually associating the MAC address and a specific switch port—but again, this is a very labor-intensive option.

UDP First Response Wins (1)

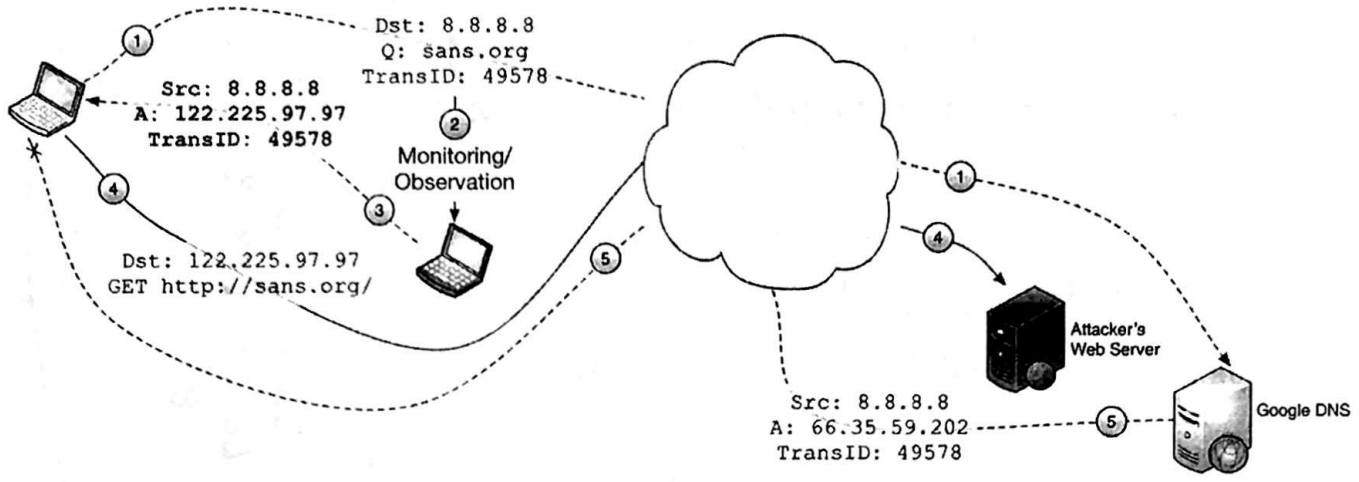
- UDP traffic lacks session tracking information
  - No session setup/teardown
  - No sequencing information
- Several protocols rely on this fact
  - UDP multicast and response
- Mitigations
  - Application layer validation

For all the efficiencies UDP provides by being "unreliable", it also suffers from various security-related issues due to the lack of validation of the protocol itself. Traffic sent via TCP enjoys reliability by establishing an individual session through the three-way handshake and sequencing information. These ensure proper ordering and overall integrity of delivered payloads.

UDP, on the other hand, does not have these checks in place. Therefore, when a UDP-based application listens for a response to an outstanding request, it generally accepts the first response it receives—provided the IP and port information match what's expected. A spoofed packet crafted to meet the expected parameters will be happily passed to the calling application. It is up to the application to validate the response and determine the truthfulness of the source of the information.

Several UDP-based protocols benefit from this paradigm, including DNS, DHCP, and more.

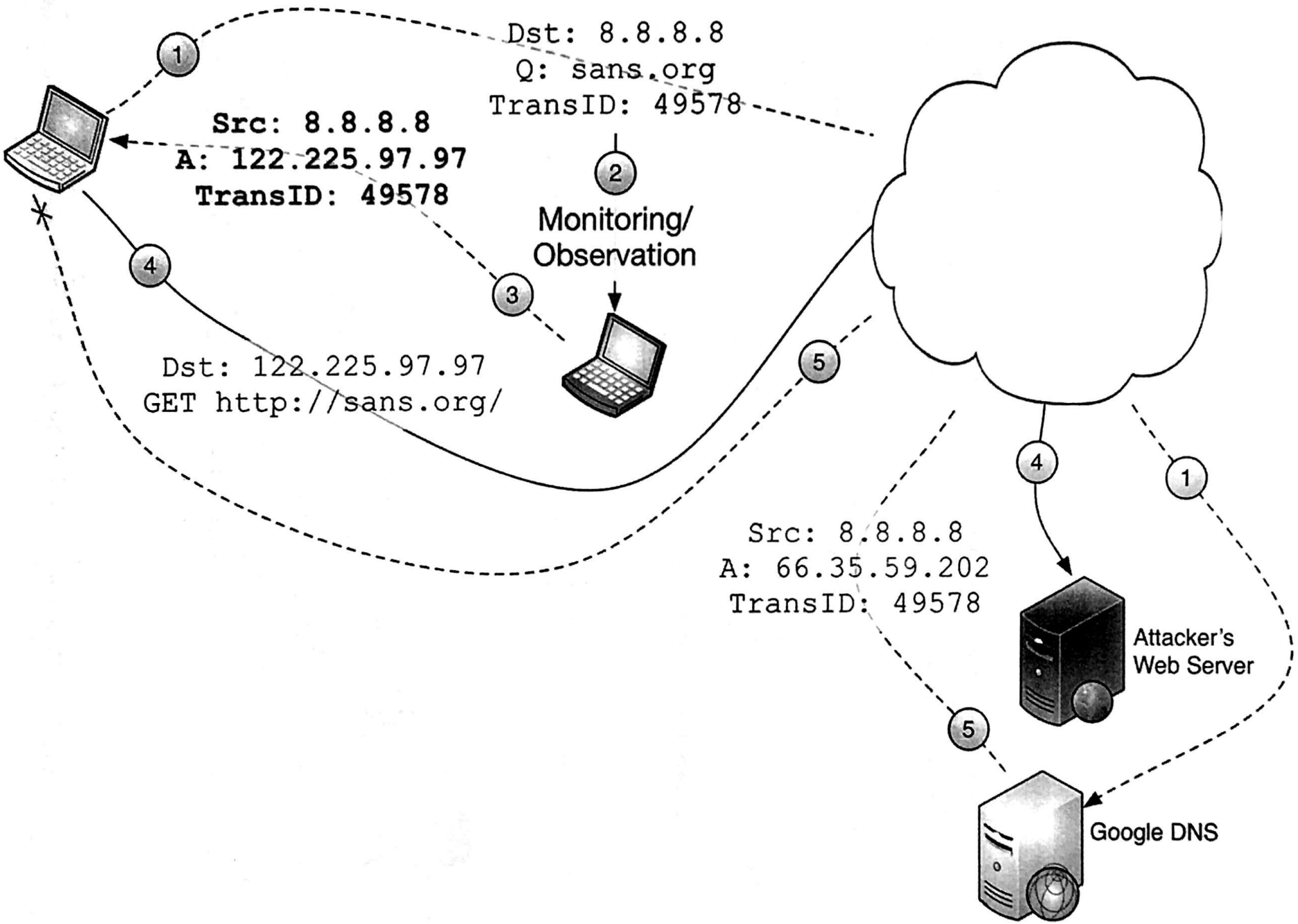
UDP First Response Wins (2)

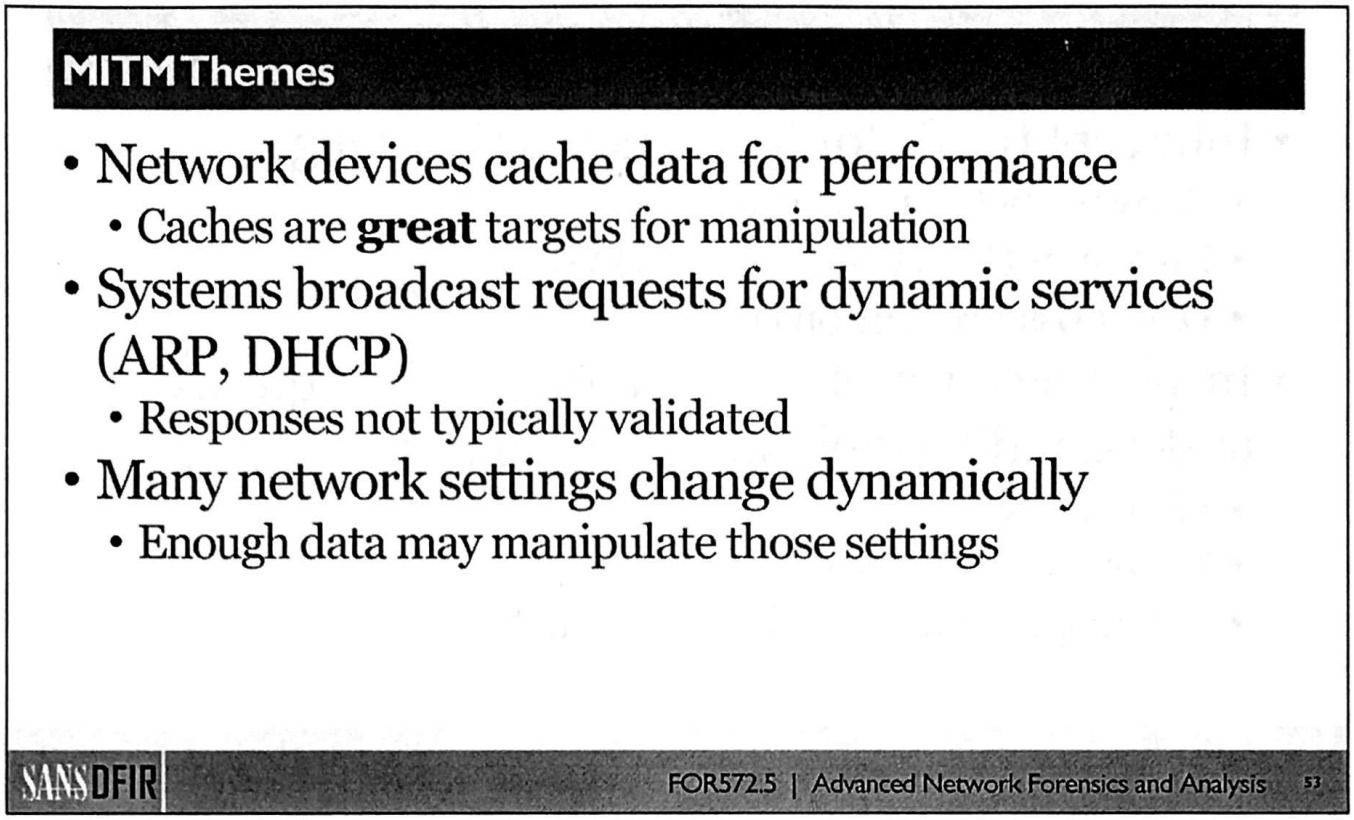


An example of the "UDP first response wins" problem:

1. A client application sends a DNS request to a server using a third-party DNS provider outside the client's network.
2. An attacker, monitoring the client's local network, examines the DNS request.
3. The attacker crafts a response that corresponds to the request and crafts a spoofed reply that matches. The client receives the DNS response and interprets the results as if it had come from a DNS server.
4. The client establishes communication based on the contents of the spoofed DNS reply.
5. At some point, the client receives the response from the true DNS server it queried. However, because the outstanding DNS request has already been satisfied with the spoofed response, the client ignores the content.

As the attacker is monitoring the user's network locally and is closer to the client application, the specially crafted DNS response would be received by the client before the DNS server providing responses outside the clients network, thus, the client application would receive whatever response that arrives first (in this case, most likely the attacker's).





Through the previous three examples, there are three common themes:

- Flooding a network device with information will ultimately modify the device's cache with information provided by the attacker, thus allowing the attacker to redirect traffic to their system. Caches are used by network devices to improve overall performance, making them an easy target for attack.
- Responding to broadcast requests faster than the legitimate respondent device allows the attacker to provide the first response and ultimately manipulate the target machine. The majority of broadcast requests are used for initial setup and dynamic configuration of hosts on a network. Because most responses to broadcast requests (ARP, DHCP, etc.) are not validated, the victim machine trusts the first reply.
- Sending enough data to a network device may manipulate dynamic settings associated with the device. Many network devices have several dynamic settings, as the Internet as a whole is a fairly dynamic and fluid place. Instead of statically implementing every network device setting, many organizations will allow devices to inherit settings based on the data that is seen across the network. Although this can reduce the overall likelihood of a static setting conflicting with a change in an upstream device, this also allows attackers to manipulate those settings depending on the amount of data sent by the attacker.

There are several additional MITM attacks not covered here but that also fall into the same three themes:

- STP Mangling: Allows an attacker to manipulate the spanning tree protocol configuration of a network device on a switched network to obtain data.
- DHCP Spoofing: Because DHCP requests are made as broadcasts, if an attacker responds faster than the real DHCP server, the attacker can manipulate the IP address, Gateway, and DNS assigned to the victim.
- ICMP Redirect: An attacker forges an ICMP redirect packet to redirect traffic to the attacker's machine.
- IRDP Spoofing: An attacker can forge an advertisement packet pretending to be a router on the LAN. The attacker can send a preference level and lifetime to ensure the attacker's system will be the preferred router.

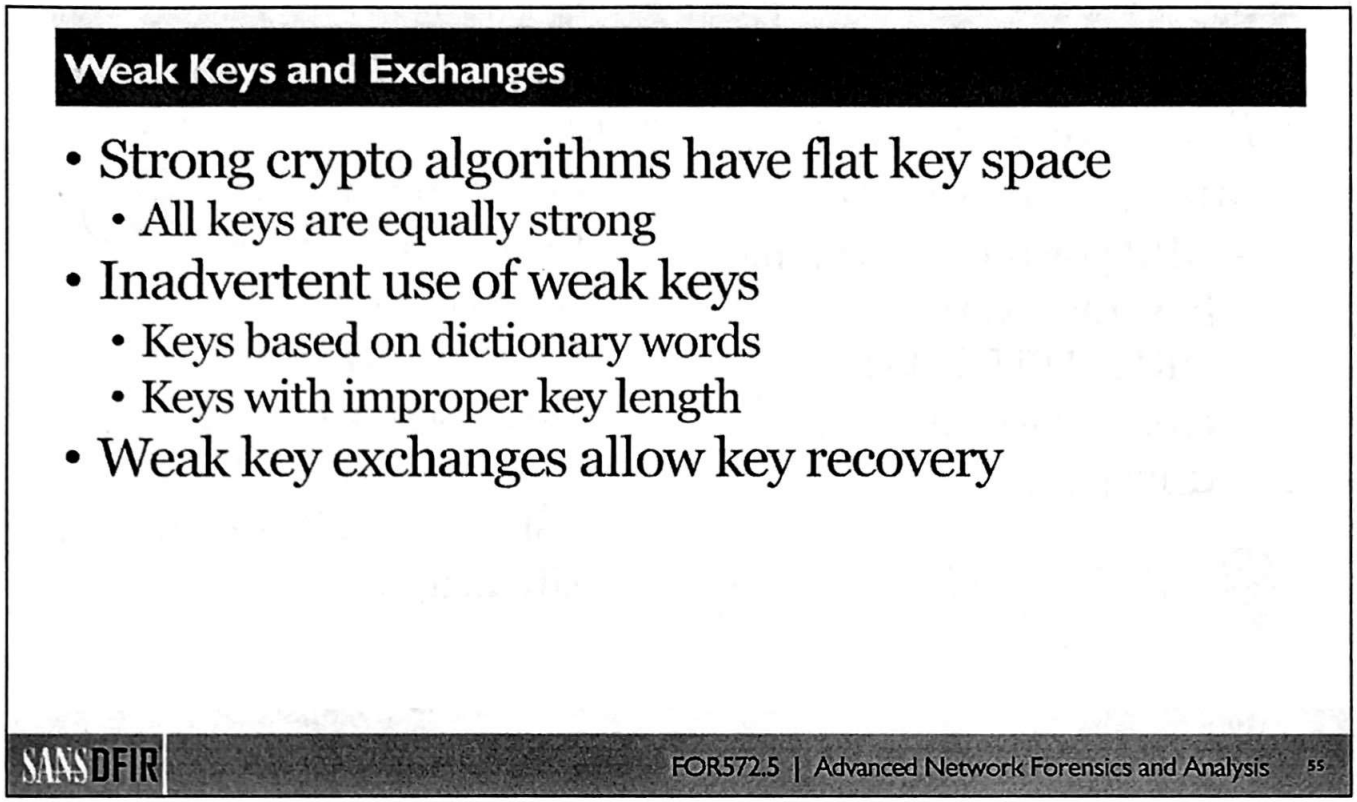
MITM for Network Defense

- Intercept traffic for inspection or blocking
  - Forward network proxies
  - Intrusion Detection System (IDS)
  - Data Loss Prevention (DLP)
- Inspecting encrypted communications requires understanding their specific weaknesses
  - Weak keys
  - Weak key exchanges
  - SSL inspection and subverting trust

Man-in-the-Middle "attacks" are not just used by attackers to inject, manipulate, or filter network traffic. In fact, several legitimate technologies perform MITM functionalities to aid in network defense.

- Forward(ing) proxy: a network proxy that forwards requests from a client to the target server. Typically forward proxies are capable of inspecting outbound requests and their subsequent responses to determine whether requests or responses should be blocked or filtered. Some common methods used for content filtering include URL or DNS blacklisting, IP blacklist, URL regex filtering, MIME filtering, or content keyword filtering. More sophisticated proxies may implement content analysis techniques to look for patterns of activity such as beaconing.
- Intrusion Detection System (IDS): Broad content inspection attempting to identify patterns of activity that match known signatures of malicious behavior. Uses the same basic technology and methodology as DLP systems, but in a less focused manner.
- Data Loss Prevention (DLP) Systems: Systems that perform focused deep content inspection, specifically seeking the outbound transfer of sensitive material such as intellectual property, personally identifying information, credit card data, etc. This type of platform also serves an important role in a live IR context, in that the IR team can observe an attacker's traffic as it leaves the environment to gain intelligence on their intent and capabilities. This is essentially a "reverse IDS" that is often deployed with connection-killing features to mitigate the risk of loss when an attacker starts to exfiltrate data.

When specifically dealing with encrypted traffic, most traditional devices that perform network traffic inspection are foiled, because encryption is inherently intended to prevent inspection! However, depending on the environmental architecture and nature of the encrypted traffic, there are a number of options available that may allow for a defender to become aware of what the attacker is doing. These include weak keys, weak key exchanges, or attacking at the termination point of encrypted traffic and then re-encrypting it to the target system.



When a weak encryption key is used, it makes a cipher more vulnerable to an overall attack. As mentioned during the prior module, the overall strength of a cipher is dependent upon multiple factors, including the key that is used to generate a more secure ciphertext. One design goal of encryption algorithm inventors is to have a "flat" key space where all keys are equally strong. However, for some algorithms (such as DES) a small number of weak keys is acceptable, provided they are all identified or identifiable. In the case of DES, there are exactly four keys where the key for encryption and decryption is the exact same. A large number of weak keys is a serious flaw in any cipher design because there will be a large chance that a randomly generated key will be a weak one, thus compromising the overall security of the algorithm.

Attackers implementing encryption functions can inadvertently use weak keys as part of their algorithm through the following methods:

- Using dictionary-based words as keys. This subsequently makes the entire ciphertext susceptible to dictionary-based attacks.
- Using keys that fill a portion of the key buffer (e.g., using a 20-bit key when the algorithm uses a 32-bit buffer). Typically, encryption algorithms will pad the rest of the buffer with "`0x00`" bytes, making the key easier to reveal the underlying ciphertext.

In certain circumstances, regardless of how strong a key is, if the key exchange between two cryptographic systems is weak, then the key is susceptible to being recovered and used for decryption purposes. It is not uncommon for attackers to use embedded hardcoded passwords within malware or transmit an encryption key through an alternate means.

Open Source MITM Software: Bettercap and dsniff

- Bettercap (replaces outdated Ettercap)
  - ARP poisoning, dynamic host discovery
  - URL, HTTP POST, credential dumping, transparent proxy

- dsniff
  - Collection of tools focused on various protocols
  - Can use ARP, MAC, or DNS spoofing for MITM
  - Includes tools to MITM SSH, SSL, pull credentials, URLs, and more

Bettercap[1] is a new incarnation of the venerable (but abandoned) Ettercap MITM tool. It primarily uses ARP poisoning to facilitate dynamic host discovery and traffic interception on a per-host or per-network basis.

After subverting ARP to intercept the traffic, Bettercap can employ any of a handful of modules to acquire information from the user's target(s). These allow the attacker to intercept, manipulate, or block the victims' traffic. These plugins include the following features out of the box, and the tool can be extended easily with Ruby code to accommodate new protocols and payloads.

- Credential collection (IMAP, POP, SMTP, SMB v1/v2, IRC, HTTP Basic/Digest)
- URL acquisition
- HTTPS hostname
- Transparent proxy

`dsniff`[2] is an open-source collection of tools originally designed for penetration testing and auditing. It includes a number of applications designed to monitor various protocols for interesting data such as e-mails, passwords, and files. It also includes several applications (`arpspoof`, `dnsspoof`, `maccof`) to facilitate the interception of network traffic normally unavailable to an attacker due to Layer 2 switching, allowing for the user to perform MITM attacks against targeted hosts. In addition, `dsniff` includes two additional applications that allow for exploiting weak key bindings associated with some SSL and SSH connections.

References:

[1] `http://for572.com/njzrs`
[2] `http://for572.com/c5mfd`

Open Source MITM Software: Yersinia

- Appropriately named after "Yersinia pestis" bacteria
- Linux/Unix Only
- Focused on a large number of layer 2 protocols
- Demonstrates even older tools can be ruthlessly effective against legacy protocols

Yersinia is another open-source tool that implements various MITM attacks against Layer 2 protocols that result in allowing the targeted system's traffic to be redirected through the attacker's.

Yersinia is capable of attacks against protocols that are fundamental to most networks—including many that are seldom monitored for misuse:

- Spanning Tree Protocol
- Cisco Discovery Protocol
- DHCP
- Hot Standby Router Protocol (HSRP)
- Dynamic Trunking Protocol
- 802.1Q
- 802.2X
- VLAN Trunking Protocol

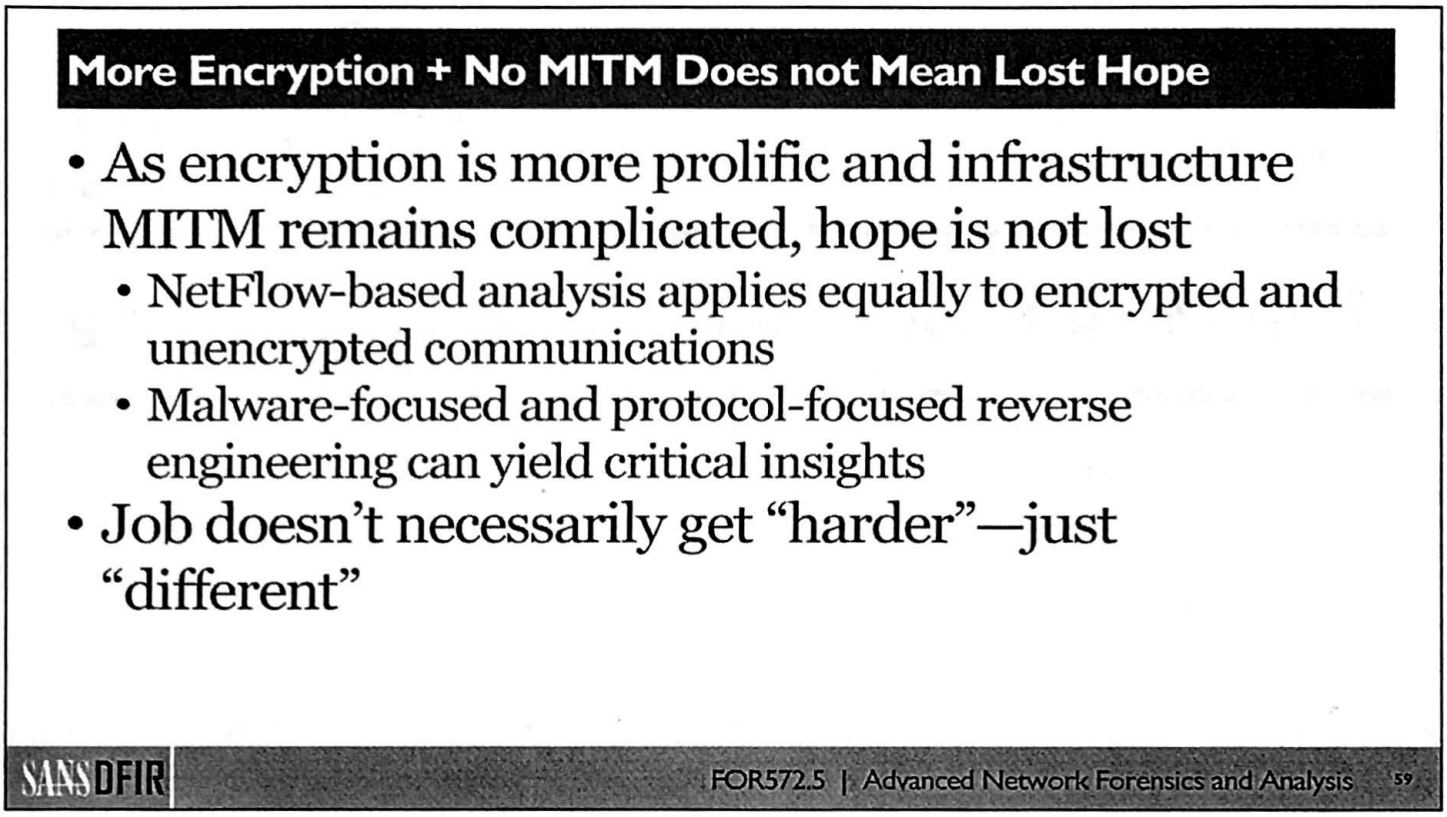
References:
`http://for572.com/94siw`

Several commercial solutions also offer varying ranges of MITM capabilities. Although most commercial solutions typically provide the ability to perform SSL inspection, a number of appliances also allows for inspection of other secure protocols such as SSH/SCP and SFTP. Commercial solutions also provide additional benefits over open-source solutions including focus on performance and speed, in addition to offering a number of options combined into a single product. Commercial solutions offering SSL inspection capabilities include:

- Symantec/Bluecoat Web Proxy
- Multiple Cisco Appliances such as Cisco Ironport
- PaloAlto Networks Firewall
- Cisco/SourceFire SSL Inspection Appliance
- WebSense Content Gateway Web Proxy

Because these solutions represent legitimate uses of the MITM model, they also assume a level of trust between the client systems and the SSL Inspection platform. One way many of these systems provide a transparent user experience is through the use of dynamic certificate generation. The MITM platform identifies the hostname the client is requesting, then uses its own CA (which is already trusted by client systems) to generate the necessary certificate that allows the SSL negotiation to complete.

Additionally, the platform convergence currently seen in the security market means many of these devices and services are starting to address a wider variety of typical requirements. Several of the platforms listed (for example, Symantec/Blue Coat and PaloAlto Networks for starters) provide transparent SSL decryption but also create a pcap collection of the decrypted traffic for real-time inspection with legacy or incompatible platforms. This affords a great deal of flexibility to the IR team, who may be able to perform common functions from an existing collection point.

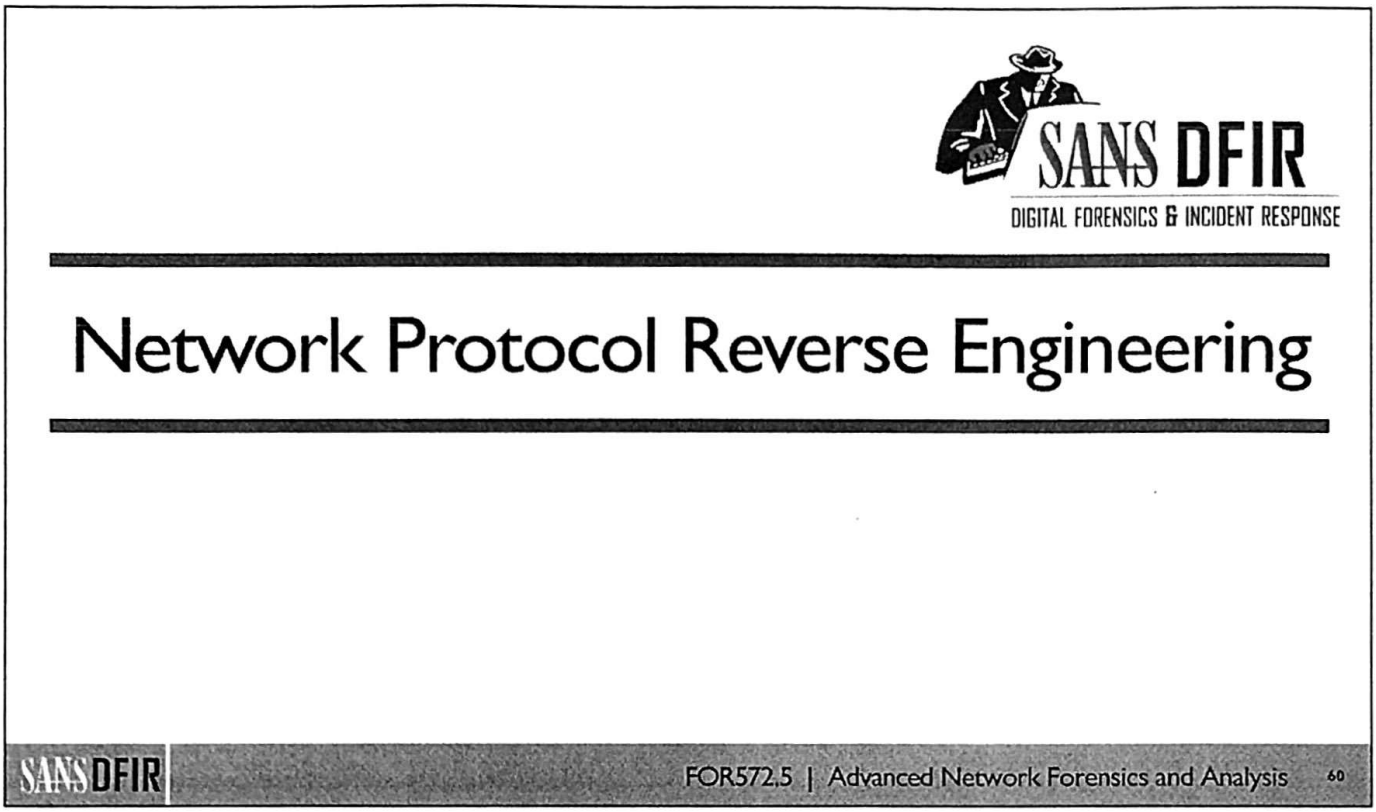


Although the trend toward more extensive use of encryption is obviously underway and arguably irreversible, the use of Man-in-the-Middle technologies remains legally and technically complicated. However, although these issues will take time and investment to fix, there are still significant benefits DFIR professionals can provide even when faced with functionally opaque communications.

NetFlow analysis is a content-agnostic investigative medium that can provide equally valuable insights into any communications, regardless of whether they are encrypted or not. Additionally, advances in the state of reverse engineering and protocol analysis often yield deep insight into the nature of encrypted communications that can often give a forensicator the critical foothold needed to pick apart an encryption implementation based on its uncovered weaknesses.

Although these both represent a change in our collective investigative approach, they do not necessarily make our work "more difficult". To the contrary, early investment in developing these skills will pay tremendous dividends in the future.

References:
`http://for572.com/dre39`
`http://for572.com/odbqz`

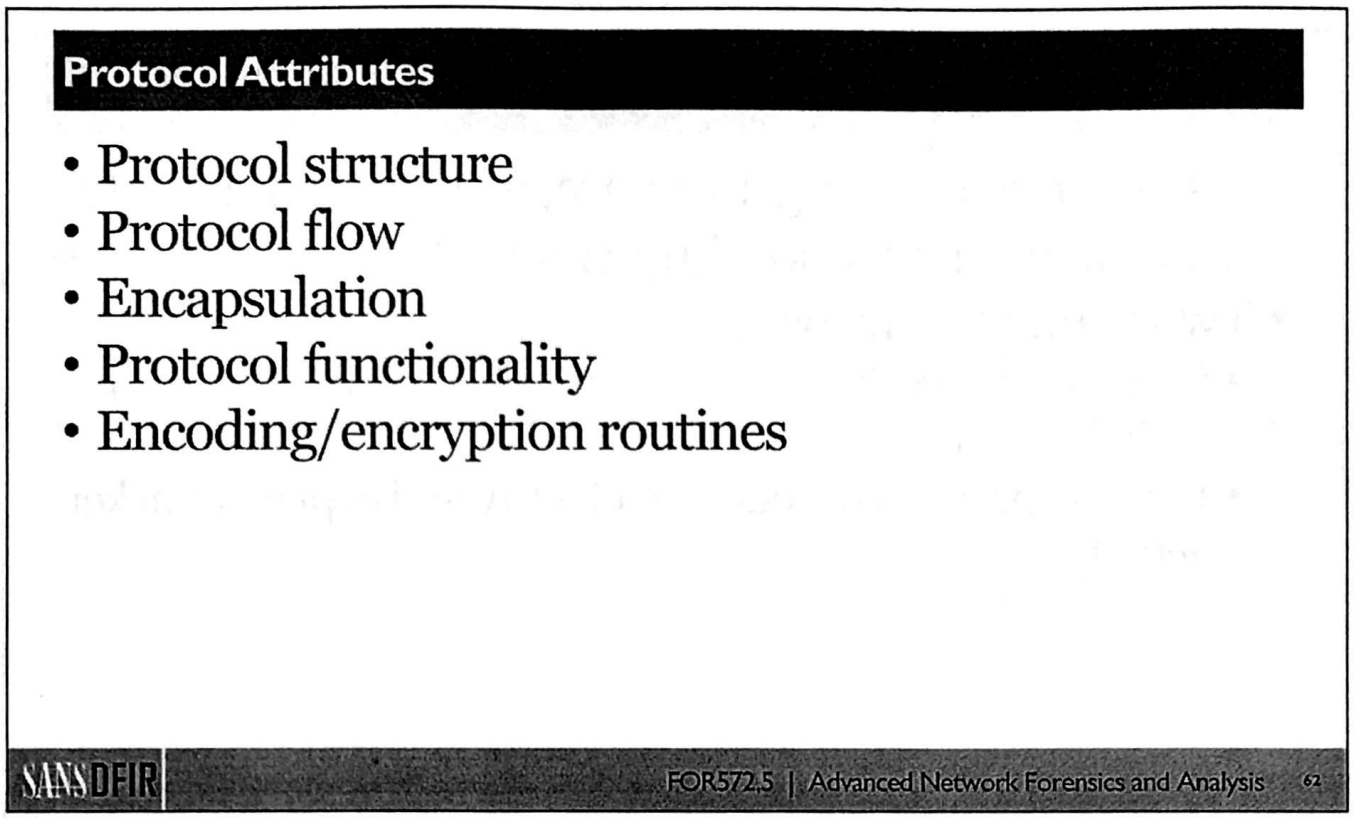


This page intentionally left blank.

Network Protocol Reverse Engineering

- Process of extracting structure, attributes, and data from network protocol implementations
- Two primary purposes
  - Generate network-based indicators to identify additional malicious activity
  - Develop protocol decoders to identify and explain attacker activity

Network protocol reverse engineering can best be defined as the process of extracting structure, attributes, and data from a network protocol implementation without access to its specifications. Typically, the content and purpose of a protocol is typically determined through "conventional" reverse engineering of the malicious binary using static or dynamic analysis. This process can be time-consuming and for some protocols may be unnecessary. In certain circumstances, the goal is to obtain as much information as possible as quickly as possible to support one of two goals: the generation of additional network-based indicators to identify additional malicious activity and the development of protocol decoders to identify what actions the attacker carried out.



When performing protocol reverse engineering, analysts typically focus on attributes that can be used to generate signatures or uniquely identify the protocol in question. These attributes commonly include the following:

- **Protocol structure:** The layout of control signaling, metadata, and payload data for each command issued as part of the protocol
- **Protocol flow:** The timing, order, size, and directionality of each complete command and corresponding response
- **Encapsulation**: The protocol encapsulating the user's data (i.e., if the carrier protocol is ICMP, UDP, TCP, SSL, HTTP, etc.)
- **Protocol functionality:** The set of functions and commands that may be issued to a client by the attacker
- **Encoding/encryption**: The means by which each protocol datagram is transformed prior to encapsulation, commonly used to evade generic IDS signatures by attackers

Approaching Protocol Reverse Engineering

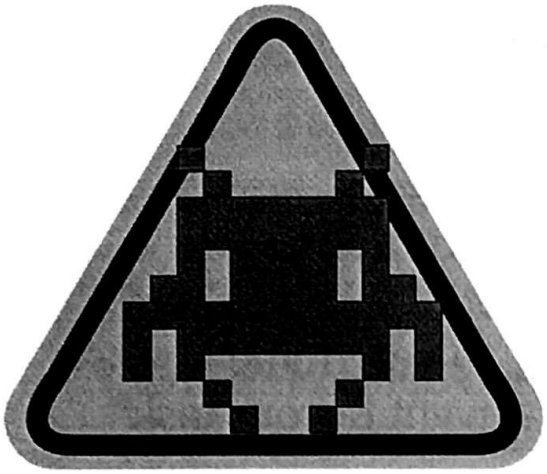
- Requires one of three data sources
  - Client binary or source code
  - Server binary or source code
  - Captured network traffic
- Identify what you are attempting to achieve
  - Balance goals against time and effort to achieve them
  - You don't need to know everything to determine how malware works or how it's most relevant to your task

Performing protocol reverse engineering typically involves using at least one of three sources of data: the client binary or source installed on the compromised machine, the server binary or source code used by the attacker, and the captured network traffic. Ideally, all three items are available but in reality, typically only the client side binary and/or network traffic are available.

Outside of requiring data sources to perform the protocol reverse engineering, an analyst must also determine what their ultimate goal is. Many protocols today have levels of complexity that might make this goal impossible without a significant amount of time and effort. However, realistic goals in understanding how a protocol works can help to develop additional network-based indicators to identify future malicious traffic on a network. What this also means is that an analyst may need to become comfortable with several principles that typically go against digital forensics.

Through protocol reverse engineering, you may not need to know everything about a particular protocol to identify both how to detect the protocol for additional malicious activity or identify the capabilities of the malware itself (such as file uploads and downloads, etc.). If as an analyst you identify a unique string within the protocol itself that always exists among a large enough population of traffic, that information is good enough to write an indicator that can identify subsequent malicious activity.

Disclaimer



All protocols discussed in this section are real life samples used by Advanced Persistent Threat Actors

All the protocols contained within this module are real life samples, used by Advanced Persistent Threat Actors and documented by various security companies. Although the protocols are the same, the names and faces have been modified to ensure no organizational or actual APT indicator information is released. You can read additional information about each sample based on the report linked within the notes section.

Similarly, we will follow a path of a traditional APT-based compromise, something that has been repeated thousands of times at organizations all across the world. Assume an organization has been targeted by a spear-phishing e-mail. One of the recipients assumed the e-mail was legitimate and opened the attached PDF, infecting their system with a first stage downloader communicating to an external website.

Compromised Web Pages/Sites

- Commonly used by attackers to
  - Host malicious executables
  - Embed commands read by malicious software
  - Participate in Fast Flux DNS architectures
- Hides in plain site on legitimate sites
  - Complicates use of endpoint threat intelligence
- Analysis requires:
  - Knowledge of the HTTP Protocol
  - Knowledge of HTML standards
  - JavaScript/CSS coding skills

Attackers commonly use legitimate websites and domains to host malicious information and executables. This allows the attackers to make use of resources like processing power, bandwidth, and the hosting availability of compromised web servers in addition to being able to mask the true source of their location. However, because in many cases, the websites are legitimate websites, an analyst may need to troll through a significant amount of information to identify what is actually happening. For example, does the site contain malicious JavaScript that is redirecting to another site to download a file, does the web page contain embedded information that is read by a malicious backdoor, etc.? Having a thorough understanding of both the HTTP protocol and HTML assists in the identification of what is actually happening.

Example 1: Request

```
GET /index.htm?1308772046 HTTP/1.1
User-Agent:
  IPHONE8.5(host:malwarehunter,ip:192.168.1.135)
Accept: */*
Host: bad.domain.com
Connection: Keep-Alive
```


This first example covers a request that is issued by a compromised system after the user opens a malicious e-mail. You spoke with the victim and identified the time the user opened the e-mail and started your search for additional activity. Immediately after opening the e-mail, you identify some suspicious HTTP-based traffic. Although basic, you start to identify additional oddities. For example, in this case the GET request states the User-Agent is IPHONE8.5; however, the user claims to be using their laptop, not an iPhone. Similarly, embedded within the request is the name of the compromised host and the IP address. Additional research indicates that there are no known User-Agents that start with IPHONE8.5.

This request has at least one attribute (the unique User-Agent) that would be able to be turned into a signature to identify additional malicious activity on the network. Similarly, additional information is embedded within the User-Agent that can assist the analyst in identifying additional compromised systems. It is not uncommon for attackers to embed identifying information into common request headers. This allows attackers to look through log files and differentiate between which systems may be connecting back. Although sometimes the information is not encoded, many times it is.

Example 1: Response

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
X-Powered-By: ASP.NET
Connection: close
Content-Type: text/html
Accept-Ranges: bytes
Content-Length: 2550

<yahoo sb="Sv|@yC2wyMxwy1#XC|wm(26322)"></yahoo>

<html>...
```


When reviewing traffic, you examine the response associated with the issued request. When looking at the response, you easily identify the response header indicating the server responding is a Microsoft-IIS/5.0 server and the application that is providing the response is Microsoft's Active Server Page programming language, which indicates the page may be capable of generating dynamic content. Lastly, the total content length for the returned information is 2550 bytes.

However, when reviewing the actual data returned, you identify an odd tag `<yahoo sb=></yahoo>` that looks like normal HTML tags but are before the `<html>` tag associated with the returned document. With an understanding of HTML structure, you find this odd because typically tags are contained between `<html>` and `</html>` for the browser to render the information effectively. Similarly, the usage of the word "yahoo" raises suspicions, as in most cases that is not a typical name for an HTML tag, nor built within the HTML standard. It's further suspicious through the random characters that appear between the tags.

Once again, without knowing exactly what the malicious software is doing, an analyst can easily pick out additional unique information that may be searchable—in this case, specifically the `<yahoo sb=>` and `</yahoo>` tags. This information can lead to further signatures or allow an analyst to look for additional unique information contained within the captured traffic.

Example 1: Expanded Scope

- Additional samples:

```
<yahoo sb="Sv|@yC2wyMxwy1#XC|wm(26322)"></yahoo>
<yahoo sb="HCwSk6Y#kj3#kUg4fw#G(4378)"></yahoo>
<yahoo sb="qNH#Z|YM6GiKhe|Y3USeWDOln(12336)"></yahoo>
...
```


Through your analysis, you decide to check for additional "yahoo sb" tags within the captured network traffic. From the results returned, you find at least two other instances, also calling out to the same malicious IP address. As with most network analysis (and analysis in general), the observations you make help identify patterns. Although you may still not be sure what the actual malware does, you have gotten to the point where you believe the malware dropped by the spear-phishing e-mail appears to interact somehow with information contained within the `<yahoo>` tags. Simultaneously, you can start to build some assumptions about the encoding of the data contained between the yahoo tags. Specifically, all three variants have an integer between parentheses. Two of the three variants have two pipe characters (|) between the tags; the third does not. Based on those assumptions, it appears that although it is not exact, the malware probably parses something along the lines of the following statement:

```
<yahoo sb="<encrypted data>(<integer>)"></yahoo>
```

Because the server response contained the phrase ASP.NET, which is a programming language used for generating dynamic web pages, you may also assume that the encoded statement is probably dynamically generated using ASP.NET. Because the data appears to be dynamically generated and is encoded, there is a likelihood that a key is needed to both encode and decode the data. The key may be contained within the malware itself or because there appears to be an integer field that changes dynamically but is contained between parentheses, the key itself could be a key that is used for encoding.

Example 1: Meta-temporal Analysis

- "Meta" activity
  - After receiving the "<yahoo>..."-tagged page, each system made an additional download from the same IP address
  - Contents of the second download appear to be a Windows executable based on file magic ("MZ" bytes)

Finally, through your analysis, you decide to look at traffic coming and going from the malicious IP address. After each request, you see another request from the compromised system to download a file from the malicious IP address. This downloaded file appears to contain a Microsoft Windows portable executable header.

Example 1: Summary

- By analyzing just the protocol and not knowing anything about the malicious software, we can determine that the malicious software reads encoded commands from a legitimate web page between <yahoo></yahoo> tags and has the capability to download and execute a malicious file on a compromised system

Based on the analysis conducted so far, we've arrived at the following observations:

- The get request contains information about the compromised host including its host name and IP address within the User-Agent string.
- The User-Agent string is a non-standard value which makes for a good indicator when looking for additional traffic.
- The response contains <yahoo> tags, which are non-standard HTML tags.
- The information contained between the <yahoo> tags appears encoded and may be encoded using some algorithm with a dynamically generated key, potentially the integer value identified between the parentheses.
- The command, when decoded, most likely downloads a file from a location on the compromised server.

By analyzing just the protocol and without analyzing the malicious software, we can theorize that the malicious software reads encoded commands from a legitimate web page between <yahoo> tags and has the capability to download and execute a malicious file on a compromised system.

From the Mandiant APT 1 Report:

> WEBC2-YAHOO enters a loop where every ten minutes it attempts to download a web page that may contain an encoded URL of a file to download and execute. The encoded URL will be found in the pages returned inside an attribute named "sb" or "ex" within a tag named "yahoo". This tag may be placed anywhere in the page and be usable by the malware but in practice it is normally placed at the very beginning of the page by the attacker.
>
> The malware will send beacon requests every 2 or 3 minutes. When the malware receives a response from the server, it then searches the returned content for "<yahoo sb="<%text%>(<%int%>)"> </yahoo>". The "<%int%>" variable is an integer used to index into a crypto array when decrypting the "<%text%>" portion.

References:
http://for572.com/5aqly

Backdoors Hidden in Common Protocols

- Malware may use common protocols for command and control purposes
  - Often uses protocols with arbitrary and optional fields
  - Know normal to spot outliers for further inspection
- Allows hiding in plain sight
- Analysis requires
  - In-depth knowledge of the protocol
  - Knowledge of usage of the protocol
  - Pattern recognition

Although malicious software can contain custom protocols, often times malicious software uses existing libraries and common protocols for command and control purposes. As seen within the prior example where the attacker added host information to the User-Agent string, in many cases such information may be encoded within the GET or POST statements, cookies or other fields of the HTTP backdoor. This allows the attacker to embed identifying information within a common protocol to "hide in plain sight" and potentially make it past a company's web-based proxy, something that may be harder for a typical binary protocol to achieve. Similarly, an attacker may wrap a custom protocol within HTTP headers to ensure their communications can flow over common protocols and avoid detection.

Example 2: Request

```
GET /1.html HTTP/1.1
Cookie:
  Y29tbWFuZD1HZXRDb21tYW5kO2NsaWVudGtleT0zOTU0O2hvc
  3RuYW1lPW1hbHdhcmVodW50ZXI7
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0;
  Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR
  3.0.04506.648; .NET CLR 3.5.21022; .NET CLR
  3.0.4506.2152; .NET CLR 3.5.30729)
Host: 123.123.123.123:8080
Connection: Keep-Alive
```


As part of your analysis, you look for additional traffic that originated from the compromised system—specifically traffic that occurred immediately after the malicious executable was downloaded (and likely executed). You filter out all the known malicious traffic and come across the following GET request which is the first GET request after the executable. When looking at the GET request you initially go through all the parts of the request itself. The URI information and format appear to be correct, the User-Agent string appears to be valid, and the host information appears to contain an IP address and port. However, one oddity that does just not look right within the header is the Cookie field.

Although the HTTP protocol does not define the layout of fields contained within HTTP headers, armed with basic knowledge of how browsers work in general, as an analyst you identify the Cookie field is typically contained at the bottom of a request if issued by a web browser. For example:

Safari:

```
GET /favicon.ico HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_4)
AppleWebKit/536.30.1 (KHTML, like Gecko) Version/6.0.5 Safari/536.30.1
Accept: */*
Referer: http://www.google.com/
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Cookie: SID=DQAAAO8AAACI3BDW39lzpe8Zra-J7nDJg3tO<snip>
```

Chrome:

```
Host: www.google.com
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_4) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/28.0.1500.95 Safari/537.36
X-Chrome-Variations: CNOlyQEIl7bJAQiptskBCMS2yQEIloTKAQj7hMoBCLiFygE=
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Cookie: PREF=ID=5138dfe5
```

Firefox:

```
Host: www.google.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:22.0)
Gecko/20100101 Firefox/22.0
Accept: image/png,image/*;q=0.8,*/*;q=0.5
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Referer: http://www.google.com/
Cookie: NID=67=sQ8UktBtJk4QGQgS3HJro
```

Example 2:Request (Decoded)

```
$ echo
  "Y29tbWFuZD1HZXRDb21tYW5kO2NsaWVudGtleT0zOTU0O2hvc3RuYW
  1lPW1hbHdhcmVodW50ZXI7" | base64 -d
command=GetCommand;clientkey=3954;hostname=malwarehunter;
```


Armed with this knowledge, you identify that this information is unlikely to be requested by a web browser and potentially by some other application, maybe the newly downloaded backdoor. When looking at the Cookie information, you remember that HTTP protocols commonly encode information in an encoding scheme that is guaranteed to return ASCII text such as base64, UTF-7. Trying some of the more popular encoding routines, you stumble across the following when decoding via base64.

```
command=GetCommand;clientkey=3954;hostname=malwarehunter;
```

With absolutely no knowledge of the malware but experience based on knowledge of protocols and understanding of HTTP, you are successfully able to identify and start to recover commands issued by the backdoor.

References:
`http://for572.com/jler7`

Mixed Protocol Analysis

- Mixes common protocols with binary protocols
- Example may include binary data wrapped within HTTP headers
- Analysis requires:
  - In-depth knowledge of the protocol
  - Knowledge of usage of the protocol
  - Knowledge of binary data structures
  - Pattern recognition skills

Analyzing mixed protocols (such as binary-based information submitted over HTTP) provides attackers the ability to transmit custom-encoded data within headers that appear to be legitimate and may bypass traditional detection mechanisms such as web proxies. Analyzing mixed protocols requires knowledge of both the common protocol (in our case, this will be an HTTP header) and knowledge of identifying patterns within binary output.

Example 3: HTTP POST Request Headers

```
POST /search25548?h1=FIFEFDAHAPGDENCMFNFFFNAGAH
  HTTP/1.1
User-Agent: Mozilla/5.0 (compatible;Windows NT 5.1)
Host: 123.123.123.123
Content-Length: 144
Connection: Keep-Alive
Cache-Control: no-cache
```


When looking at additional traffic, you identify yet another potential odd request header. When looking at the header itself, you see another User-Agent string that doesn't appear to be a normal User-Agent. When looking at your favorite resources for identifying common User-Agent strings, you notice that most User-Agent strings have a space after a semicolon for information contained between parentheses. While not necessarily indicative of malicious activity, you decide to extract the POST data to a file so you can look more closely at its contents.

References:
http://for572.com/gqu12

Example 3: HTTP POST Request Body

```
$ cat http_post_payload.bin | hexdump -C
000000CD  f3 d7 dd cc d1 cd d1 d8  ca 9e e9 d7 d0 da d1 c9 |........ ........|
000000DD  cd 9e e6 ee 9e e5 e8 db  cc cd d7 d1 d0 9e 8b 90 |........ ........|
000000ED  8f 90 8c 88 8e 8e e3 b3  b4 96 fd 97 9e fd d1 ce |........ ........|
000000FD  c7 cc d7 d9 d6 ca 9e 8f  87 86 8b 93 8c 8e 8e 8f |........ ........|
0000010D  9e f3 d7 dd cc d1 cd d1  d8 ca 9e fd d1 cc ce 90 |........ ........|
0000011D  b3 b4 b3 b4 fd 84 e2 fa  d1 dd cb d3 db d0 ca cd |........ ........|
0000012D  9e df d0 da 9e ed db ca  ca d7 d0 d9 cd e2 cb cd |........ ........|
0000013D  db cc d0 df d3 db e2 fa  db cd d5 ca d1 ce 80 be |........ ........|
0000014D  be be be be be be be be  be be be be be be be be |........ ........|
```


Because the header information is suspicious, there's very little to go on and indicate the activity is malicious. However, the POST request itself appears to be binary information. One unusual observation is that not even a single byte of this content is printable ASCII.

When analyzing binary protocols, it is common to look for repeated patterns. This may suggest a key used for encryption or decryption, because it will appear whenever the key is XORed with the null byte (`0x00`), which is often used for padding a message to a block boundary (usually 64 or 128 bits). Although in some cases a key may be easily recovered, testing against basic encoding or encryption techniques using permutations of the associated pattern can often times lead to breakthroughs.

In this example, the entire last row contains the hex value "`0xbe`" repeated over and over. As the value appears to be used repeatedly at the end of the message, and the message itself aligns with a 128-bit boundary, the "`0xbe`" may indicate a one-byte key used for encoding the associated data.

Example 3: HTTP POST Information Decoded

```
$ xor-dec $( echo -n -e "\xbe" ) ⏎
  http_post_payload.bin http_post_decrypted

$ file http_post_decrypted
http_post_decrypted: ASCII text, with CRLF line
  terminators

$ cat http_post_decrypted
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\username\Desktop>
```


After XOR-decrypting the file, its contents reflect plaintext ASCII. Examining the decrypted file contents reflects what appears to be a Microsoft Windows shell prompt in what was the POST data.

This is an example of a basic (yet absurdly effective) web shell, in which the HTTP request contains the prompt. The server, which is under the attacker's control, responds with the command to be run.

Binary Protocol Analysis

- Intended purely for machine reading
- Typically contains message headers or metadata encoded within the protocol
- Analysis requires:
  - Basic binary data structure knowledge
  - Pattern recognition skills

A binary protocol is a protocol that is intended or expected to be read by a machine rather than a human being, as opposed to a plaintext protocol such as IRC, SMTP, or HTTP. Binary protocols have the advantage of terseness, which translates into speed of transmission and interpretation. Detection of a binary protocol is dependent upon the security devices in place. For example, configuring a web proxy to block all non-HTTP-based traffic over port 443 will most likely result in blocking a binary protocol from communicating with its command and control server.

When analyzing binary protocols, typically additional metadata about the payload must be passed between the client and server. This information commonly includes the size of the following message, such that the server can translate the appropriate amount of data.

Example 4: GH0ST RAT (1)

- GHoST RAT is a remote administration tool
  - Used in compromises of embassies, political opponents
  - Source code released by unknown parties
- Implements binary command and control protocol

```
struct gh0st_header {
        char    gh0st_word[5]
        DWORD   compressedSize;
        DWORD   uncompressedSize;
        char    compressed_payload[];
};
```


The GH0ST Remote Administration Tool (RAT)[1] is a backdoor whose source code has been widely distributed online. It was used by a nation-state-like actor known as "Gh0stNet" to compromise numerous embassies and various offices of the Dalai Lama.

The compiled source code provides attackers with many ways to control a victim system, including the ability to create/manipulate/delete/launch/transfer files, perform screen capture, perform audio capture, enable a webcam, list/kill processes, open a command shell, and wipe event logs. The GH0ST RAT source code uses a custom binary protocol that is compressed using the ZLIB compression library and prepended with the `gh0st_header` structure shown here and includes the following three fields:

- A static string appears in the first five bytes
- The size of the compressed payload byte array
- The size of the decompressed payload byte array

References:

[1] `http://for572.com/lgm61`

Example 4: GH0ST RAT (2)

- Encoded Message

0x8c000000 = 140 0xe0000000 = 224

```
$ cat ghostrat.tcpflow.bin | hexdump -C
00000000  41 64 6f 62 65 8c 00 00  00 e0 00 00 00 78 9c 4b  |Adobe........x.K|
00000010  63 66 60 98 c3 c0 c0 c0  0a c4 8c 40 ac c1 c5 c0  |cf`........@....|
00000020  c0 04 a4 83 53 8b ca 32  93 53 15 02 12 93 b3 15  |....S..2.S......|
00000030  8c 19 18 1c 58 5a 19 78  19 20 40 86 c1 02 ac 66  |....XZ.x. @....f|
00000040  01 50 40 01 88 05 1f 33  c0 01 23 d8 14 06 86 48  |.P@....3..#....H|
00000050  20 7e 00 55 c3 c3 07 64  28 30 30 5c 00 9a c1 c8  | ~.U...d(00\....|
00000060  80 07 30 32 82 1d 01 52  c3 0c e5 ef e5 64 60 a8  |..02...R.....d`.|
00000070  07 b2 7c 1d 7d c2 1d 83  5c 3d 42 fd 42 5c 83 f0  |..|.}...\=B.B\..|
00000080  99 81 00 85 fc 40 02 00  66 ac 12 27              |.....@..f..'    |
```

140 bytes

The initial beacon packet contains basic system information about the compromised host. This includes hostname, system stats (OS version and CPU speed), and whether any capture devices (i.e. webcams) are available. Because GH0ST RAT is an open-source tool, it is not difficult to modify the source code to implement additional features. In this case, the attacker modified the source code to remove the "Gh0st" string that is typically used for default GH0ST RAT communications at the beginning of the packet header and replaced it with the string "Adobe". Following the string Adobe is the compressed size of the message (0x8C000000 in the second box) and the uncompressed size (0xE0000000 in the third box). Through analyzing the first few bytes of the message, you could determine that 0x8C000000 translates to the digit 140. You also identify the payload size here is also 140 bytes. Translating 0xE0000000 from hex to a digit indicates the uncompressed size should be 224 bytes.

```
$ ghostrat.tcpflow.bin | hexdump -C
00000000 41 64 6f 62 65 8c 00 00  00 e0 00 00 00 78 9c 4b   |Adobe........x.K|
00000010 63 66 60 98 c3 c0 c0 c0  0a c4 8c 40 ac c1 c5 c0   |cf`........@....|
00000020 c0 04 a4 83 53 8b ca 32  93 53 15 02 12 93 b3 15   |....S..2.S......|
00000030 8c 19 18 1c 58 5a 19 78  19 20 40 86 c1 02 ac 66   |....XZ.x. @....f|
00000040 01 50 40 01 88 05 1f 33  c0 01 23 d8 14 06 86 48   |.P@....3..#....H|
00000050 20 7e 00 55 c3 c3 07 64  28 30 30 5c 00 9a c1 c8   | ~.U...d(00\....|
00000060 80 07 30 32 82 1d 01 52  c3 0c e5 ef e5 64 60 a8   |..02...R.....d`.|
00000070 07 b2 7c 1d 7d c2 1d 83  5c 3d 42 fd 42 5c 83 f0   |..|.}...\=B.B\..|
00000080 99 81 00 85 fc 40 02 00  66 ac 12 27               |.....@..f..'    |
```

Example 4: GH0ST RAT (3)

- Post zlib decompression

224 bytes

```
$ cat ghostrat_decompress.bin | hexdump -C
00000000  66 03 00 00 9c 00 00 00  05 00 00 00 01 00 00 00  |f...............|
00000010  28 0d 0a 00 00 02 00 00  00 53 65 72 76 69 63 65  |(........Service|
00000020  20 50 61 63 6b 20 33 00  00 40 04 85 00 0d 00 00  | Pack 3..@......|
00000030  00 00 00 00 00 1c 00 38  00 02 00 00 00 a0 0d 00  |.......8........|
00000040  00 20 0d 00 00 11 e3 00  00 00 00 00 00 00 00 00  |. ..............|
00000050  00 01 00 01 00 00 00 00  00 59 00 00 00 e0 00 00  |.........Y......|
00000060  00 a0 0d 00 00 0c 0e 00  00 00 20 00 00 d0 04 85  |.......... .....|
00000070  00 01 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 01 01  |................|
00000090  00 00 01 00 00 01 00 00  00 03 00 00 00 00 01 01  |................|
000000a0  00 bd 09 00 00 7f 00 00  01 4d 41 4c 57 41 52 45  |.........MALWARE|
000000b0  48 55 4e 54 45 52 00 00  00 00 00 00 00 00 00 00  |HUNTER..........|
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000d0  00 00 00 00 00 00 00 00  00 00 00 00 71 0f 00 00  |............q...|
```


Decompressing this payload proved to be somewhat difficult. Although the malware uses standard `zlib` compression, it does not use the same headers and magic bites found in the most widely used `zlib` implementation: `gzip`. Therefore, standard shell utilities such as `gunzip` and `zcat` will not successfully decompress the payload. Instead, a custom python script was created to force decompression of the payload without relying on header values from a typical `gzip`-compressed file. This script is available on your SIFT workstation as "`raw_zlip_uncompress.py`" not because the script itself is particularly useful beyond this one scenario, but because writing such scripts is an inherent necessity in the line of DFIR work.

After performing decompression, you identify the decompressed message is 224 bytes and can pick out several additional strings of interest within the decompressed payload that look familiar such as "Service Pack 3" and "MALWAREHUNTER".

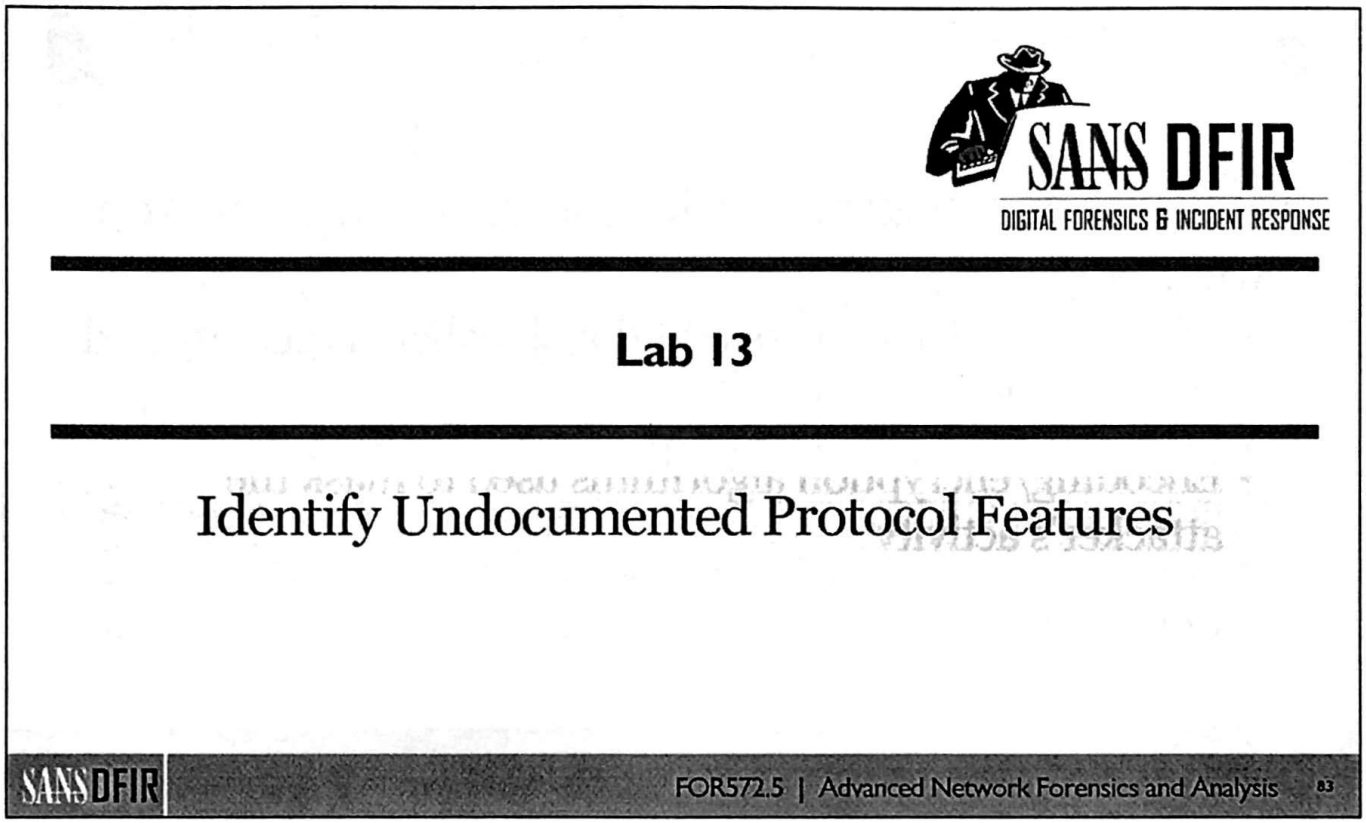
```
$ cat gh0st_rat_payload.bin | hexdump -C
00000000  78 9c 4b 63 66 60 98 c3  c0 c0 c0 0a c4 8c 40 ac  |x.Kcf`........@.|
00000010  c1 c5 c0 c0 04 a4 83 53  8b ca 32 93 53 15 02 12  |.......S..2.S...|
00000020  93 b3 15 8c 19 18 1c 58  5a 19 78 19 20 40 86 c1  |.......XZ.x. @..|
00000030  02 ac 66 01 50 40 01 88  05 1f 33 c0 01 23 d8 14  |..f.P@....3..#..|
00000040  06 86 48 20 7e 00 55 c3  c3 07 64 28 30 30 5c 00  |..H ~.U...d(00\.|
00000050  9a c1 c8 80 07 30 32 82  1d 01 52 c3 0c e5 ef e5  |.....02...R.....|
00000060  64 60 a8 07 b2 7c 1d 7d  c2 1d 83 5c 3d 42 fd 42  |d`...|.}...\=B.B|
00000070  5c 83 f0 99 81 00 85 fc  40 02 00 66 ac 12 27     |\.......@..f..' |

$ raw_zlib_uncompress.py -i gh0st_rat_payload.bin -o gh0st_rat_decompress.bin
```

For a comprehensive look at GH0ST RAT and its command structure, see the MITRE chopshop project's[1] decoding script for the protocol[2].

References:

[1] `http://for572.com/uhrxo`

[2] `http://for572.com/3q1rn`



This page intentionally left blank.

Lab 13 Objectives: Identify Undocumented Protocol Features

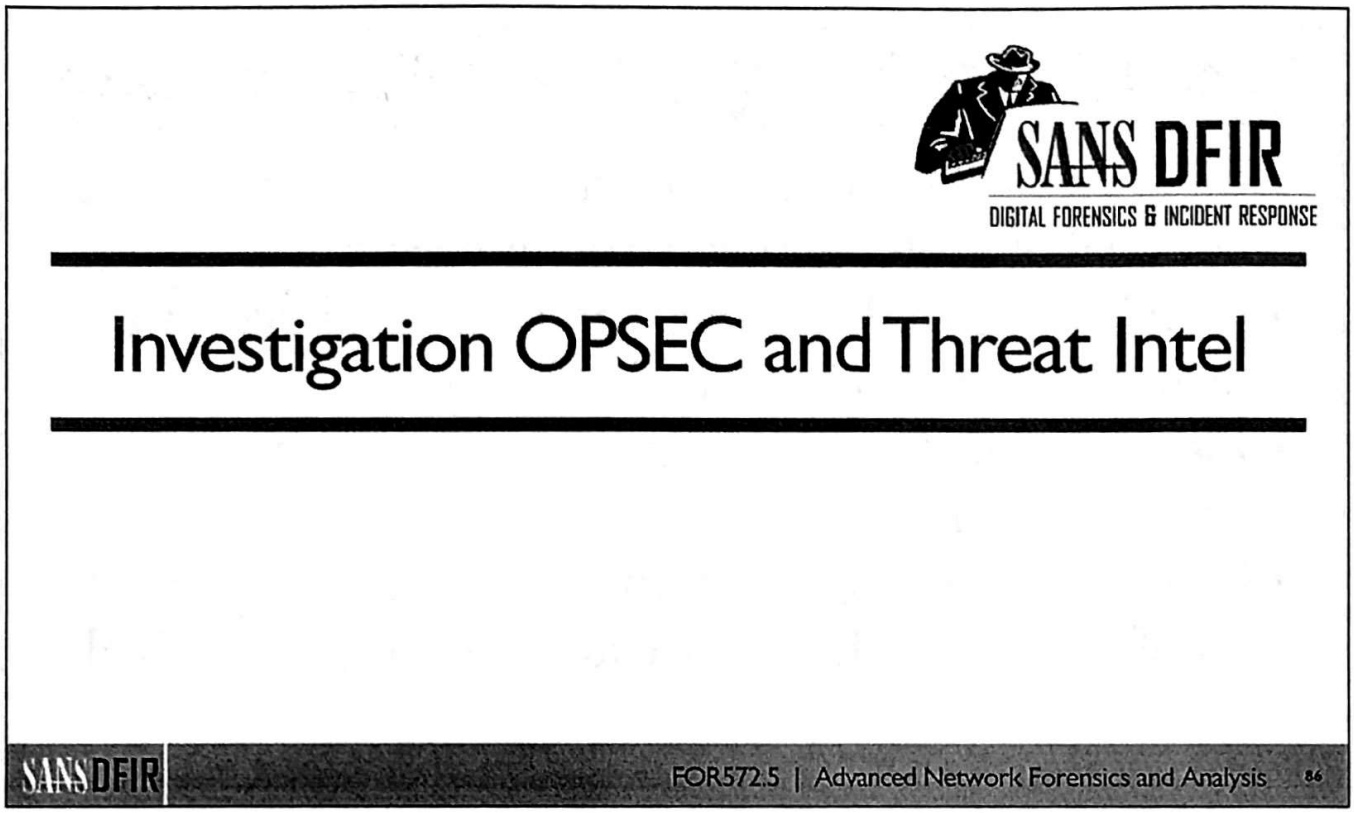
- Review real-world APT malware traffic
- Reverse engineer network protocols to identify the following:
  - Artifacts for detection of additional malicious activity and signature generation
  - Encoding/encryption algorithms used to mask the attacker's activity
  - Host information in undocumented protocol fields
  - Commands the attacker executed

This page intentionally left blank.

Lab 13 Takeaways: Identify Undocumented Protocol Features

- Identifying IOCs from undocumented protocols is a key goal of protocol reverse engineering
- Find known fields to start, then use ancillary data from DNS, etc. to fill knowledge gaps
- Identify unique and consistent byte streams to generate useful IOCs
- Fully documenting an unknown protocol can take a lot of time and effort—pace and scoping are critical

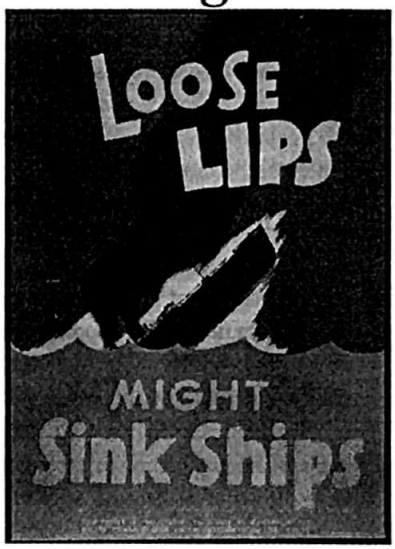
This page intentionally left blank.



This page intentionally left blank.

The Network is Alive and OPSEC is Critical

- Network environment brings unique challenges
- Research can tip off attackers
  - DNS lookups, blocking C2 traffic, pwned email/proxy/etc. servers
  - Attackers track the hunters (AKA you)
- Live collection and analysis needed
  - Offline analysis not always feasible
  - Understand implications of actions
  - Build "live" incident response plans



The concepts behind disk and even memory forensics are relatively sound, and have been tried and tested extensively. We generally acquire a "snapshot" of data at a point in time, then perform repeated analysis in an offline fashion against that data. However, performing forensic analysis in the network realm brings with it a number of concerns that must be considered before engaging in the analytic process.

For one, our research itself—or even the collection process—can cause events to occur that can let an attacker know that we're looking for them. These could include something as seemingly benign as a DNS lookup, or implementing a perimeter firewall block on a host known to be involved with malware command and control channels. If an attacker has successfully breached a proxy server, they could easily observe all web research being conducted on their malware, identities, or tactics/techniques/procedures. Of course, loading URLs associated with a malware infestation could quite likely be hazardous, bringing downloads of yet more unknown binary evil into the environment. For these reasons, we must be highly attuned to the operational security used by anyone involved with an investigation.

Another somewhat different dynamic is that given the sources of evidence in the network forensic world, much of our acquisition is done in a "live" fashion. This requires human interaction with an operational device or system that often cannot be removed from service for very long—if at all. While forensically sound procedures for disk-based acquisition of high-availability systems have been used for some time, there is no easy analog for switches, routers, firewalls, or other devices the network investigator might be interested in. Snapshots and restore points are great for this on disk-based systems, but there is not likely to be such a feature built into switch firmware, for example.

For these reasons, we must consider how conducting an investigation in such a live environment can be accomplished without compromising the integrity of its results.

First, let's talk about operational security, or OPSEC. Just as militaries and governments have a vested interest in protecting their information from the eyes and ears of adversaries, so must the forensic investigator. While OPSEC is a key factor for disk-based forensic investigations as well, it is fundamentally built into network-based

investigations from the point of collection onward. Poor or nonexistent OPSEC safeguards could effectively stop an investigation dead in its tracks.

Ideally, network capture devices and sensors would access the network only through a one-way, read-only network tap, and subsequent analysis would be performed offline, in a segregated enclave, where no manner of network traffic could be released to the Internet at large. This would almost completely mitigate any risk of an adversary knowing what the investigation team was up to. However, reality must intercede. This is not always possible, nor is it always practical. Investigators can gain great benefits from live or real-time access to Internet resources—in some cases, such access is paramount to success.

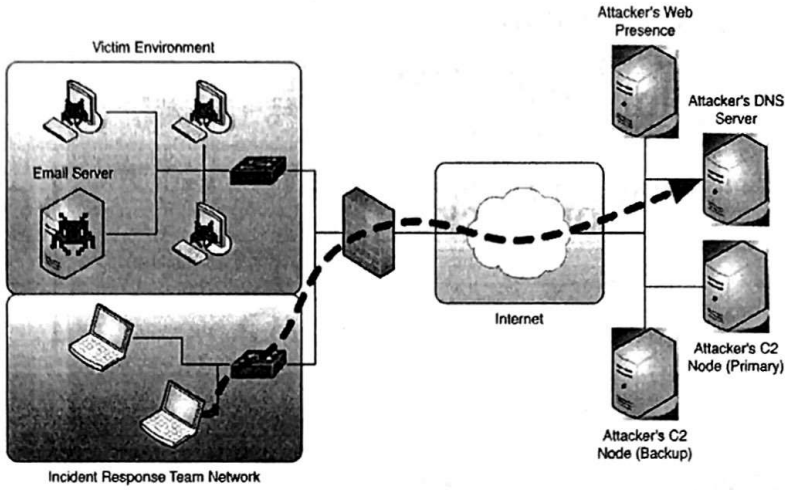
For example, maintaining duplicate databases and other records in an offline enclave sounds straightforward, but simply cannot achieve Internet scale with any degree of currency. It will never be realistic to have a near-live, offline duplicate of the entire DNS or WHOIS hierarchy, for example.

Another aspect of how important online access can be is seen in scoping an intrusion. By definition, the iterative, dynamic nature of the scoping process—determining which assets are compromised and the details surrounding an adversary's compromise and continued access of the environment—require continual access to that environment as well as the Internet. How else would it be possible to determine whether a particular type of HTTP traffic was related to an advertising company's cookie tracking or an attacker's spearphishing campaign?

We will examine some of the more common concerns with regard to investigation OPSEC in turn.

Live Research Complications: DNS Activity

- May reach attacker's own DNS server
- Uniqueness of DNS query and source network may be clear giveaway
  - Great use case for self-collected Passive DNS



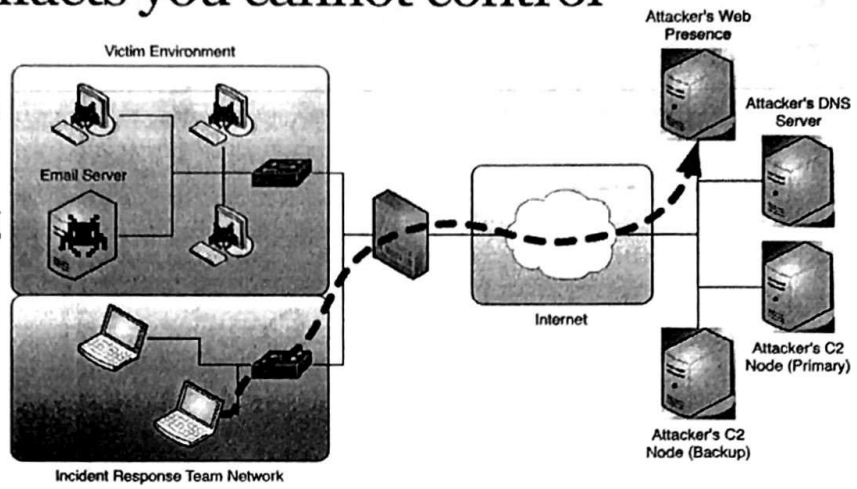
Let's start with DNS lookups. As we discussed previously, DNS traffic is some of the most common, fundamental network traffic on our networks today. However, attackers often control their own DNS servers, giving them full and complete visibility to who is querying their domains, when, from where, and how often. Simply making a query to determine the current IP address for the "c2.badguy.evil" hostname could immediately let an attacker know that an in-house or contracted incident response team is on their trail.

Because malware often uses time-based hostnames through the use of deterministic name generation algorithms, an investigator performing open-source research on a domain name that was only valid last Tuesday could let the adversary know which generation of malware has been put under the microscope. The attacker could then adapt to this new landscape, or at least know how far they are ahead of the investigation.

This is also a risk to consider when building and configuring capture and monitoring devices. A system that performs real-time DNS lookups on all observed IP addresses will not only cause a great deal of network traffic, it would also give the adversary a good understanding of the sophistication of the victim's defensive posture.

Live Research Complications: OSINT Research

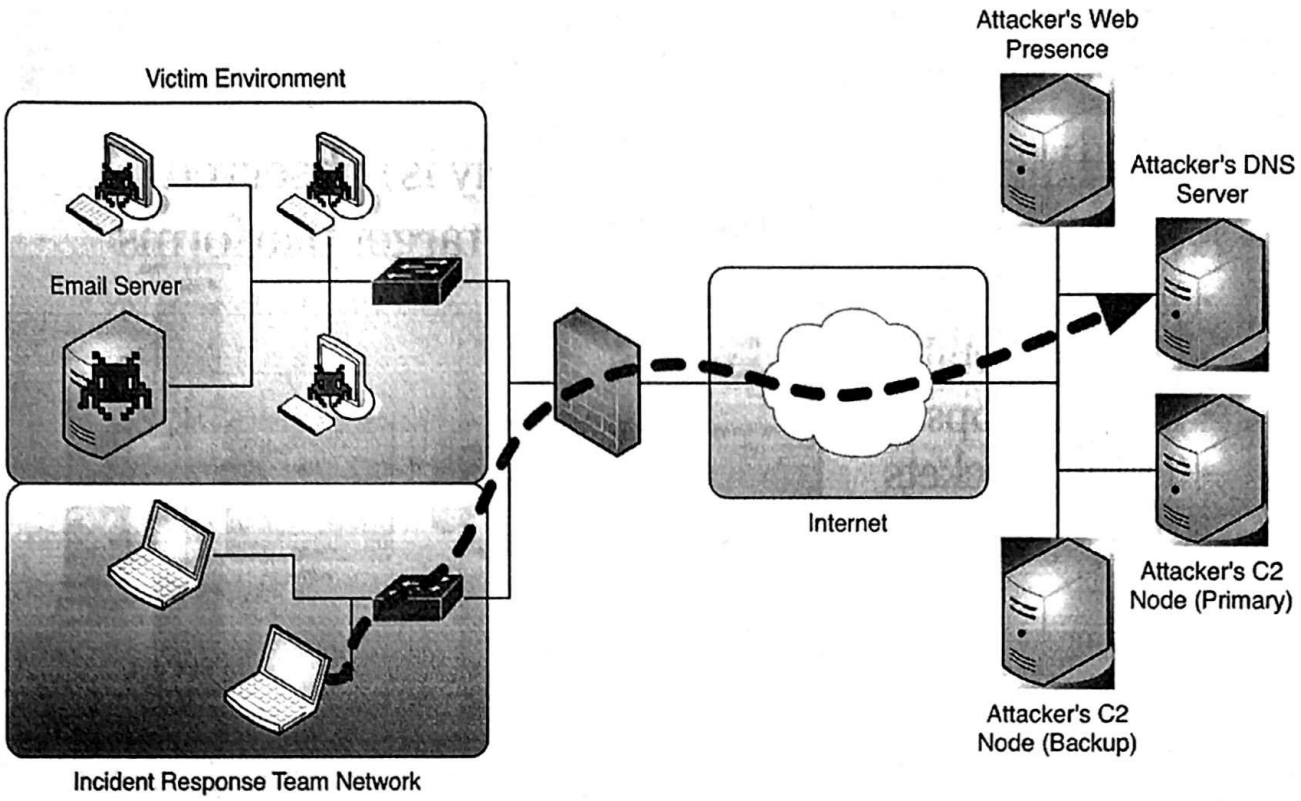
- Load web content from attacker's infrastructure
- Create public artifacts you cannot control
  - AV/malware scanning services
  - Social graphs
  - Location tracking



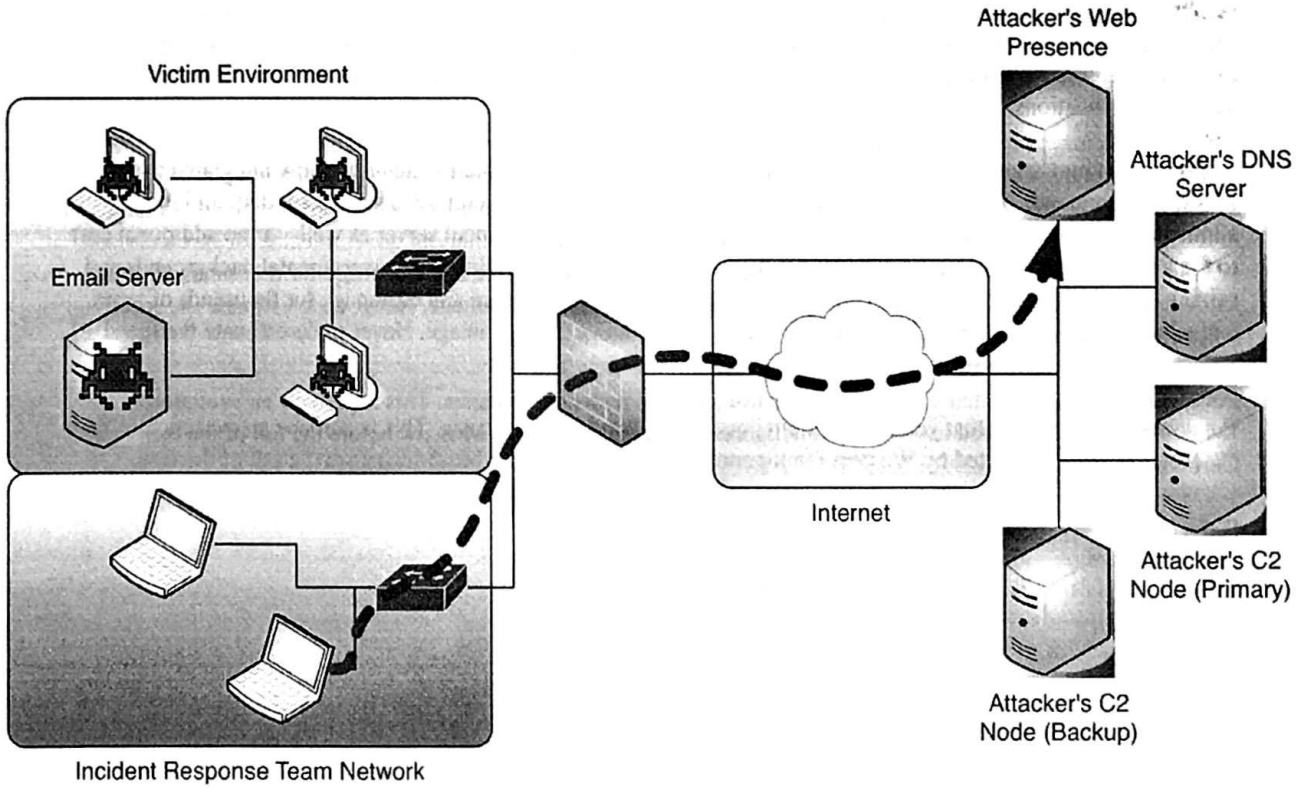
Similarly, basic Open Source Intelligence (OSINT, not to be confused with Open Source software) can provide the adversary with an early warning system of sorts. Especially if they control any sources of information (e.g., web servers) where you may find their name, investigators could quickly cause a similar situation to that of DNS lookups on the previous slide. An attacker that runs their own server can mine access logs, search terms, time frames, and other seemingly benign data sources to establish a pretty accurate picture of your research. Although many attackers won't have such a high level of counter-investigative capabilities, we must plan for well-funded and well-placed groups to have at least the same level of professional expertise as we do. Every technique we use to track an adversary can be used against us if we access infrastructure they can also access.

Even in research using hosted platforms and social media sites, the collections and tracking procedures used by the site can compromise an investigation. A startling example of this is the LinkedIn.com "People who looked at this profile also looked at…" sidebar section. If an investigation team uses a single LinkedIn account—whether real or fake—profiles can quickly become associated by the LinkedIn.com software. It's quite hair-raising to see all of your supposedly disconnected subjects/suspects show up in that sidebar at the same time. Location-based services are adding a layer of concern to this, as our mobile devices can give up a significant amount of sensitive information without our knowledge. Although more of an issue for physical investigations, the trend toward "unwitting transparency" is a real risk to our OPSEC.

Another way your own habits and behavior could compromise an investigation's OPSEC is through "leaked" tracking cookies. Many advertisers and social media sites (who themselves are arguably just advertisers) use cookies to track your activity across many different sites. If an investigator accidentally logs onto their own social media profile, then conducts research related to a case, some level of their identity could be passed along to a third-party site—possibly even to the attackers themselves. The pervasive use of the Facebook "Like" widget, Google+ "+1" button, embedded Twitter posts, and other ubiquitous "Share this page" functions all involve countless cookies and other tracking features that could contribute to a loss of OPSEC.



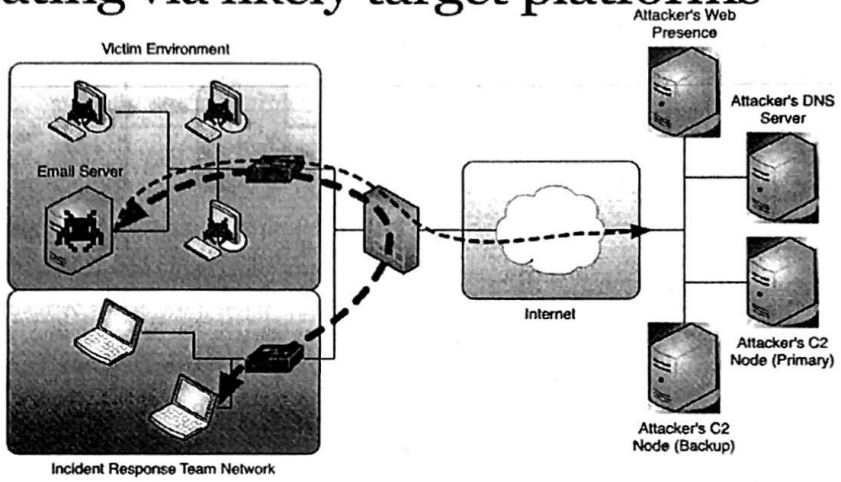
Hostname Lookups



OSINT Research

Live Research Complications: The Attacker is Watching

- Everything you do can (and likely is) observed
- Avoid communicating via likely target platforms
  - E-mail servers
  - Collab portals
  - Business ops
  - Trouble tickets



Another major risk to consider, especially before an incident has been fully scoped, is that an attacker could easily be observing all of your actions within the compromised environment. While spreading within an environment, savvy attackers seize opportunities to monitor email, IM, and other communications. This enables them to identify the state of any incident response or investigations surrounding their activity. We must therefore assume that any such communications sent within a compromised environment could be fully read by the attacker.

Although this may sound like an exotic capability, remember that most email systems are now integrated to the domain environment (Active Directory, LDAP, or something similar). Therefore, a successful domain-level administrative compromise gives immediate and complete access to the email server as well—at no additional cost to the attacker. Many nation-state or organized crime adversaries have military or governmental backgrounds and seizing a victim's lines of communication has been a basic tenet of warfare and espionage for thousands of years. Little has changed today, aside from the methodology to achieve this advantage. Never underestimate the speed with which your communications will be targeted and/or compromised.

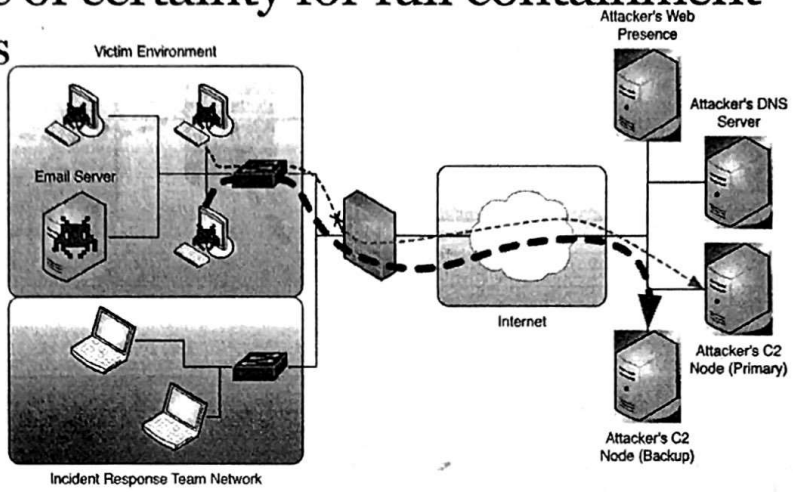
Put another way, expect that you and your environment are under surveillance. This is perhaps an evolution of the "Moscow Rules"[1] adopted by Western intelligence services operating in Moscow before the fall of the Iron Curtain.

References:

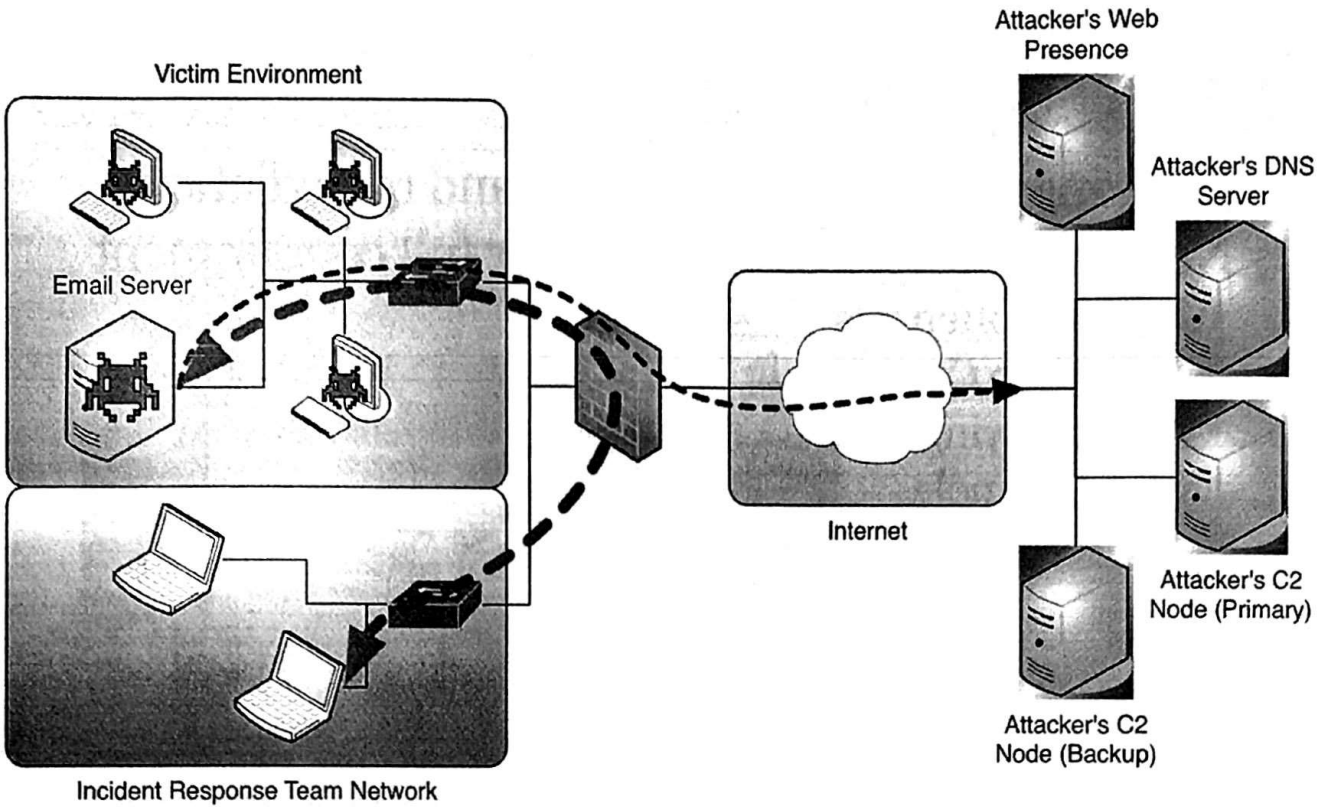
[1] `http://for572.com/wexz5`

Live Research Complications: Premature Traffic Block

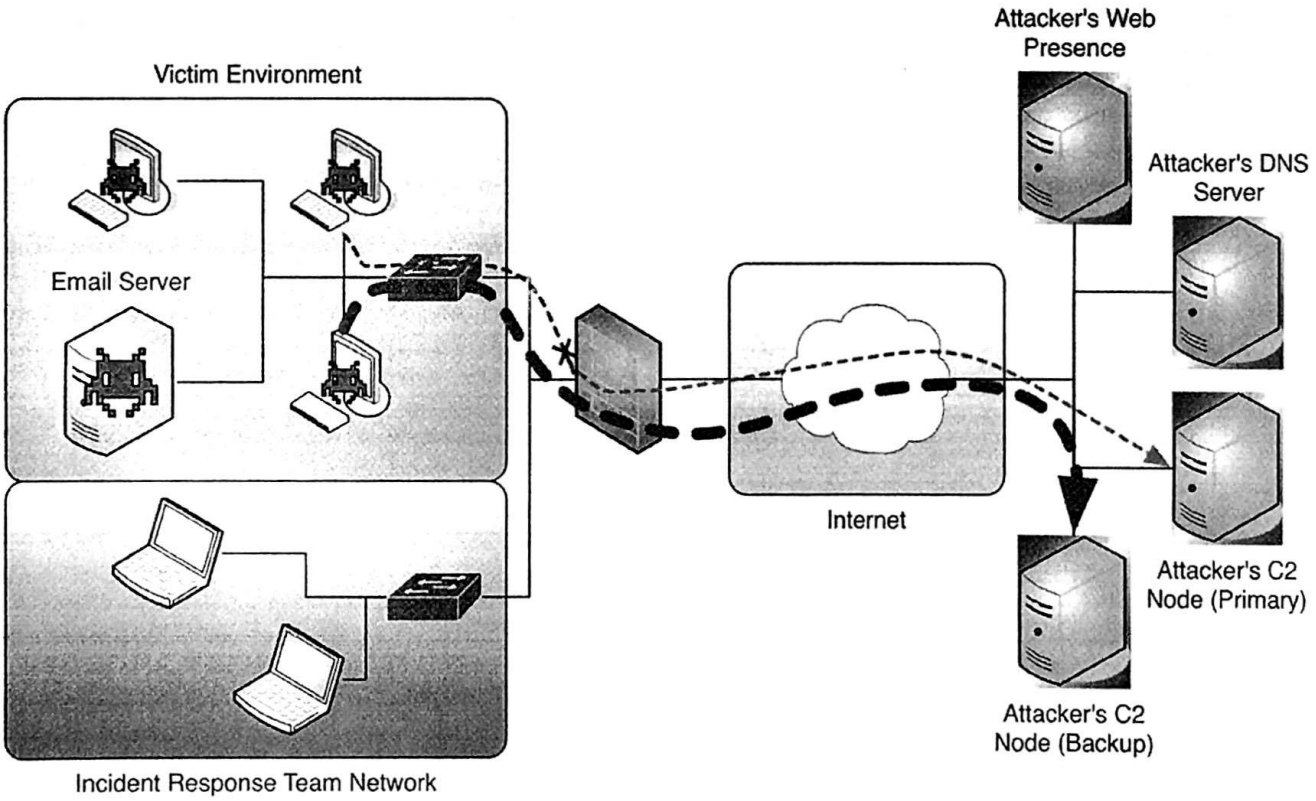
- Blocking C2 before full scoping and remediation
- Need high degree of certainty for full containment
  - Attacker often has 1+ backup C2
  - Avoid playing whack-a-mole



In what is possibly the most difficult "sell" to management, aggressive containment can be a significant OPSEC risk. Often acting too soon can prevent any hope of an effective remediation. For example, the legacy approach was to immediately block a newly-identified command and control server, or to load a new malware traffic signature to the enterprise fleet of IPSes to prevent any further infestation. However, attackers were severely trounced when this was done—but only for a short while. The attackers adapted and therefore so must the investigators.



The Attacker is Watching



Premature Traffic Block

Local Network Impact: DHCP/DNS Traffic

- DHCP/DNS are typically non-interactive
  - As soon as system links to switch, packets flow
- MDNS/UPnP broadcast services/software
  - Sends details via plaintext multicast
  - Often includes serial numbers, GUIDs, and other uniquely identifying data points

Although login sessions are typically a deliberate action taken by an investigator, there is also a great deal of automatic activity that occurs simply by connecting a system to the network. We discussed how DHCP is often the first evidence of a piece of hardware being introduced to the environment—this applies to the investigators as well! With most modern configurations, as soon as a device is connected to a switch port and a link is established, the device sends either a DHCPDISCOVER or DHCPREQUEST packet, seeking a lease from a DHCP server so it can proceed with network communications. Similarly, DNS traffic often follows, because most applications use hostnames rather than IP addresses while initiating communications.

Both of these protocols can provide critical evidence when examining suspect activities, so being able to rule out any investigator-initiated traffic will help to improve the overall signal-to-noise ratio. For example, seeing a known C2 hostname or IP address in a DNS query log could indicate further infection within the environment, or an examiner performing OSINT on the IP. (Of course that would mean the examiner hasn't taken this class to learn about managing OPSEC posture yet!)

An interesting new development in DNS traffic has been the growing use of multicast DNS (also known as zero-configuration networking, Universal Plug and Play, or Bonjour). This protocol uses DNS traffic, though usually over UDP port 5353, and usually limited to local network segments. MDNS traffic advertises the hardware and software capabilities for a given device, such as whether it is a web server, file server, printer, etc. MDNS-aware clients use this traffic to find resources with which they want to interact. This protocol can be very noisy, and provides a wealth of information about the devices using it. The simple act of an Apple system joining a network, for example, can create dozens or even hundreds of Bonjour packets to enter the network environment.

The next page contains several MDNS/bonjour messages, with interesting data points in bold text. Of course these have been redacted to prevent possible misuse.

iTunes v12.4.3.1: (Note various "*ID" values, music library name, system name, ports)

```
14:24:18.308731 IP 172.16.164.1.5353 > 224.0.0.251.5353: 0*- [0q] 10/0/4
(Cache flush) TXT "libid=A775F796FD968FE9", PTR _atc._tcp.local., PTR Phil's
MacBook Pro._atc._tcp.local., TXT "model=MacBookPro11,5" "osxvers=15", (Cache
flush) TXT "txtvers=1" "Version=196621" "MID=0xdeadbeefdeadbeef" "Database
ID=badcoffee4dad00" "Machine ID=dabbad004me2" "dmv=131085" "OSsi=0x1F6"
"Media Kinds Shared=2163759" "iTSh Version=196621" "Password=0" "Machine
Name=Philip HagenM-bM-^@M-^Ys Library", PTR _daap._tcp.local., PTR Philip
HagenM-bM-^@M-^Ys Library._daap._tcp.local., PTR _home-sharing._tcp.local.,
(Cache flush) SRV Phils-MacBook-Pro.local.:61720 0 0, (Cache flush) SRV
Phils-MacBook-Pro.local.:3689 0 0 (658)
```

1Password Password Database application

```
14:23:52.261150 IP 192.168.75.7.5353 > 224.0.0.251.5353: 0 [2q] [1au] PTR
(QM)? _hap._tcp.local. PTR (QM)? _1password4._tcp.local. (80)
```

Apple iPhone Device Name

```
14:24:18.476019 IP 192.168.75.7.5353 > 224.0.0.251.5353: 0*- [0q] 1/0/5
(Cache flush) SRV Phil-Hagens-iPhone.local.:32498 0 0 (235)
```

Google Chrome browser automatically seeking a Google Chromecast device (Note IPV6 address):

```
14:26:50.942621 IP 172.16.164.1.5353 > 224.0.0.251.5353: 0 PTR (QM)?
_googlecast._tcp.local. (40)
```

CentOS 7 system broadcasting system-level hostname (ive-netflow.identityvector.com) and HTTP service name used for Ubiquiti UniFi network management (unifi.identityvector.net):

```
14:27:06.515911 IP 192.168.75.32.5353 > 224.0.0.251.5353: 0*- [0q] 3/0/0 PTR
UniFi Controller (unifi_identityvector_net)._http._tcp.local., (Cache flush)
SRV ivs-netflow-identityvector-com.local.:8080 0 0, (Cache flush) TXT
"path=/" "name=unifi.identityvector.net" (186)
```

QNAP TS-419P II Turbo NAS (Note device name, model number, firmware version information and serial number):

```
10:48:01.396033 IP 192.168.75.250.5353 > 224.0.0.251.5353: 0*- [0q] 4/0/0
(Cache flush) SRV HagenNAS1.local.:8080 0 0, (Cache flush) A 192.168.75.250,
PTR HagenNAS1._qdiscover._tcp.local., (Cache flush) TXT
"accessType=https,accessPort=443,model=TS-419P,displayModel=TS-419P
II,fwVer=4.0.5,fwBuildNum=20140117,serialNum=NAS_serial_number,webAdmPort=0,w
ebAdmSslPort=0,webPort=0,webSslPort=0" (290)
```

Local Network Impact: Other Network Traffic

- "Automatically check for updates?"
  "Send anonymous usage statistics?"
  Licensing and anti-piracy verification
  - Most often use HTTP/HTTPS
- Merely creating traffic not necessarily a problem
  - Must be identified to minimize impact on investigation
- Host-based activity notification may help
  - Zone Alarm
  - Little Snitch

Somewhat recent developments in software design have also created a situation in which network traffic is generated behind the scenes, without the user requesting or being aware of that activity. There are a number of situations that meet this description, but here are a few:

- Software often "calls out" to its developer to determine the latest version available, and notifies the user if they need to update. With marketplaces such as Apple's App Store and Google Play, these transactions can be accomplished with a single protocol for many different software titles. However, there are countless developers who have deployed their own version-check systems that you may see flying across the wire.
- Developers have also incorporated "usage statistic" collection in their works. These features, often briefly explained and enabled at installation, queue and send a variety of data points to the developer for logging and analysis. These are sent automatically, after the user agrees to their collection.
- Some software uses the network to validate licensing status and/or as a means of addressing piracy.

In all of these cases, the developer is free to use whatever protocol they want, but in practice this most often occurs over HTTP or HTTPS because in most environments, this traffic will almost certainly get through to its intended destination.

In all of these scenarios, it's important to recognize and accept that introducing network traffic to the environment—potentially changing the environment being observed—is a reality that we need to handle. It can never be fully avoided, so our practices must adapt. It's far more important to approach this problem by minimizing the impact on the environment and then fully understanding and explaining the residual activity that may be observed within the scope of the investigation. This also includes understanding how the environment may change this traffic, through the normal use of proxies, firewalls, routers, etc.

Some software solutions such as "Little Snitch" on macOS,[1] Zone Alarm[2] on Windows, or any of their numerous competitors may be helpful in preventing traffic "leakage". However, as with any such tool, it is imperative that the forensically minded user knows the tool inside and out before using it operationally to avoid any accidental disclosure during an investigation.

References:

[1] `http://for572.com/ko9wt`
[2] `http://for572.com/sv4e3`

Research Risk Mitigations (1)

- Third-party research minimizes association
  - domaintools.com
  - archive.org, Google web cache (Watch out for images and other side-loaded resources!)
  - Proxy and VPN architectures
- Use air-gapped network when possible
  - No connectivity means no leak risk

Now that the "doom and gloom" is out the way, we'll take a look at some basic ways to mitigate the risks of compromising the OPSEC of an investigation. Of course there are no perfect solutions in this business, but knowing how to make risks less severe is a key factor in our success.

An easy method is to use external, third-party sites to conduct network-centric research. By pushing your research off the victim's network, the research more easily melts into background noise from the adversary's point of view. Many of these services, such as domaintools.com, also add extra value to their offerings, such as cross-referencing across a larger data set, historical correlation, and more.

For some OSINT research requirements, sites like the Internet Archive Wayback Machine at archive.org, or even Google's web cache can put reasonable distance between you and the source material. However, it's important to fully vet a new service like this to understand its strengths and weaknesses. For example, the Google web cache does not cache or otherwise block image loads from the original source web server by default. This leads to a situation where the source server not only knows who is loading their content, but also that they're using the Google web cache to do so. Depending on the adversary's savvy, this could be a significant finding, resulting in a significant blow to your OPSEC posturing.

Another possible solution is to use any of the various privacy-focused proxy and VPN routing services or technologies. However, some of these services and providers have profiles of their own. For example, identifying an IP address as a TOR exit node is trivial on either a one-off or an automated basis.[1]

However, it's important to consider your adversaries' skill level when determining which research methods are acceptable. Some attackers employ single-operation domains, so any research on that domain at all will uniquely identify the target that is conducting research.

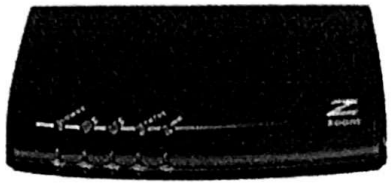
Perhaps the most straightforward solution would be to prevent any network research whatsoever! It's said that an unplugged computer encased in concrete and sunk to the bottom of the ocean is provably secure, although not very useful. In the same vein, though, a network-based investigation without network access will usually not be too effective. So rather than a wholesale air-gap solution, it makes sense to consider what really **needs** online access, which actions are just more convenient to conduct with network access, and which actions are equally effective on the quiet side of an air gap as they would be otherwise. Again, the key is to consider and mitigate.

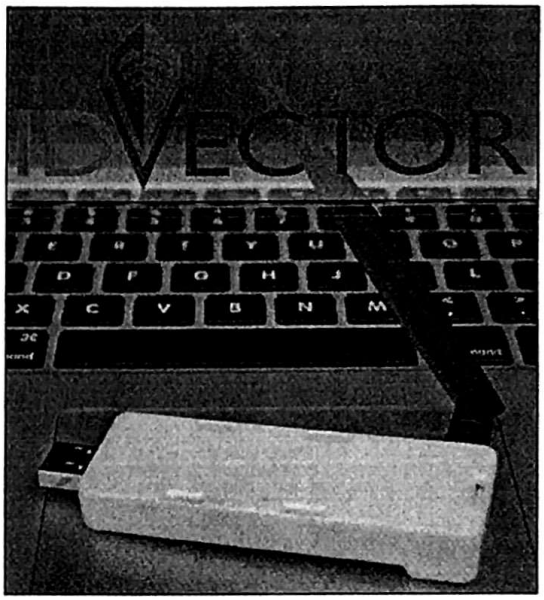
References:

[1] `http://for572.com/htp8z`

Research Risk Mitigations (2)

- Use separate network access
  - LTE hotspot in IR kit
  - Alternate access path
  - Throwaway VMs
- Never use forensic platforms!





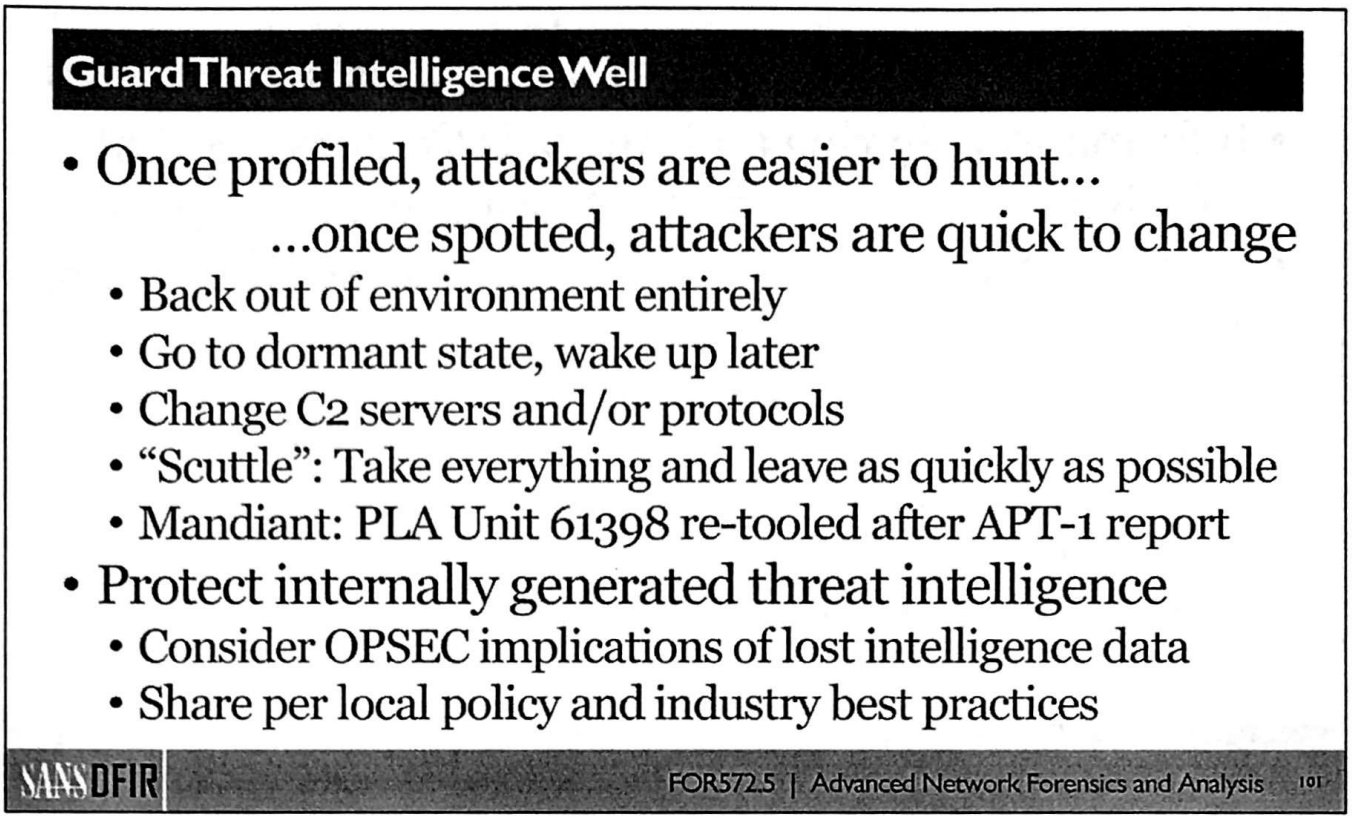
When conducting network-based research is absolutely required, a strategically appropriate solution must be used. As stated previously, operating within a victim environment can be very risky—especially before fully scoping the incident. For this reason, it is a good idea to use a fully separate connection, with no direct association to the victim (or the incident responders, if they are from an external team).

The most basic method is to keep an LTE hotspot in the "go bag", because it provides a fully separate line of communication. Another possible solution would be to install a separate land-based Internet connection for the IR team. This separate cable/DSL/fiber connection would be used solely for investigative purposes. New VPN-based technologies such as the IDVector[1] provide a secure and anonymous means of communicating from a "hostile" environment such as public wireless. This USB dongle sets up a dedicated VPN path via any wireless environment, ensuring no network traffic leaks from the system to which it is connected. The IDVector suite includes an iOS client and will expand platform support in the future.

And, although this should go without saying, it is important enough to re-iterate here:
NEVER use your forensic workstation or platform to perform Internet-based research! The risk of compromise of your entire analysis is simply too great to justify saving the time it would take to use a more appropriate research solution. Don't become the "lesson learned" at the end of the investigation!

References:

[1] `http://for572.com/idvector`



Engagement managers must walk a fine line between sharing necessary intelligence with system administrators, but doing so in a manner does not give the attackers insight to the IR team's intel.

Today, any sufficiently advanced adversary considers the risk associated with being identified by the victim, and includes several levels of contingencies to address that risk. So a C2 block that is implemented before the full scope of the infestation is known, or the full capabilities of the malware have been determined can be disastrous. Modern adversaries will quickly identify that the primary C2 was lost, then shift to a secondary method. Many will implement a "back off" period, because they know the investigative team is closing in. They may defer or delay data theft transfer activity, pause C2 check-ins for days or weeks, or implement any number of other backup plans to maintain a foothold in the victim's environment.

Sometimes, attackers may even move to scuttle an operation, stealing anything that's already been staged for exfiltration, in expectation of soon being driven out of the environment.

If an attacker ever does gain access to the team's intelligence on them, they will certainly change their tactics to evade further detection and mitigation. This was documented by Mandiant, when just three months after releasing their APT1 report[1] the attack group they identified as PLA Unit 61398 was already retooling their operations and moving to new infrastructure.[2]

As covered in SANS FOR578, Cyber Threat Intelligence, generating your own threat intelligence is an important component of incident response and threat hunting. However, this intelligence would be incredible useful to the attacker, allowing them to remain concealed and free to operate. Therefore, keeping that intelligence as close-hold as possible is paramount to the operational security and eventual effectiveness of the incident responders' actions.

References:

[1] `http://for572.com/c39t8`
[2] `http://for572.com/j16kh`

Plan to Share Before the Incident: ISACs

- Information Sharing and Analysis Centers (ISACs)
  - Evolved from critical infrastructure to wider infosec base
- Provide means to share intel within industries
  - Defense Industrial Base (DIB)
  - State/local governments
  - Healthcare
  - Financial Services
  - Aviation
  - Public Transportation

NationalCouncilofISACs

In 1998, US Presidential Decision Directive 33[1] laid the groundwork to create sector-specific groups that would share information about threats and vulnerabilities faced by their members. Soon thereafter, the notion of an Information Sharing and Analysis Center, or ISAC, was born. Today, nearly two dozen ISACs cover a wide variety of industry sectors. Some groups provide 24/7 warning functions, disseminate critical information about vulnerabilities and evolving threat actors, and more. ISACs are often more agile then similar (yet inherently bureaucratic) government organizations. This is a critical factor when dealing with aggressive, fast-paced threat groups.

These organizations are aligned by industry, which provides a narrower focus than the likes of the US-CERT or even The MITRE Corporation's CVE® program—which are both still very important, of course. Perhaps one of the most useful features the ISACs provide is a relationship structure in which member organizations can "get to know" their counterparts from other organizations within the same industry.

The National Council of ISACs[2] provides an umbrella over all of the member ISACs and is a good starting point for anyone wishing to learn more or become involved.

References:

[1] `http://for572.com/u25c6`
[2] `http://for572.com/87xdu`

Protect Threat Intelligence but Share Responsibly

- Community-defined "Traffic Light Protocol" (TLP)
  - Defines widely-recognized threat intel protection levels

Color	When should it be used?	Quick Reference
RED	Sources may use **TLP:RED** when information cannot be effectively acted upon by additional parties, and could lead to impacts on a party's privacy, reputation, or operations if misused.	Named recipients only
AMBER	Sources may use **TLP:AMBER** when information requires support to be effectively acted upon, but carries risks to privacy, reputation, or operations if shared outside of the organizations involved.	Organizational distribution
GREEN	Sources may use **TLP:GREEN** when information is useful for the awareness of all participating organizations as well as with peers within the broader community or sector.	Community-wide distribution
WHITE	Sources may use **TLP:WHITE** when information carries minimal or no foreseeable risk of misuse, in accordance with applicable rules and procedures for public release.	Unlimited

Regardless of whether you share information internally, within an ISAC, or through other partnerships, knowing how to share—or not to share—data is important. To address this requirement, the security community developed the "Traffic Light Protocol", or TLP,[1] as a means of designating the sensitivity of intelligence that needs to be shared with partners, but kept from adversaries. The TLP structure is not limited strictly to information security use, nor only to incident responders.

It is used in many nations and industry segments, so the definition for each level of protection is defined generically enough to apply widely. For example, someone who receives threat intelligence that is coated "TLP:RED" must not share that with any other party unless explicitly authorized by the originator. A "TLP:AMBER" communication can be shared within each recipient's organization. "TLP:GREEN" designates information that can be freely shared within the overall community it is intended for, and "TLP:WHITE" indicates the information can be disseminated freely and publicly (within copyright constraints).

Those coming from a government or military background will be familiar with this style of dissemination restriction, and it is important for those in strictly commercial/civilian situations to understand so they can interact with the broader security community.

References:

[1] http://for572.com/i68gk

FOR572 Alumni Google Group, SANS DFIR Mailing List

- http://for572.com/alumni-group
  - Stay in the loop on developments in the Network Forensic community
  - Collaborate with other practitioners in a variety of industries and locations
  - Members must be approved by their instructor
- http://for572.com/sans-dfir-list
  - DFIR-wide community discussions
  - Job opportunities

The course authors and instructors have created a Google Group so former FOR572 students can discuss what they're seeing in their daily tasks. This community-type focus has been a big benefit for several other SANS courses in the Forensic curriculum, and we hope to foster the same camaraderie with Network Forensicators as well.

Membership in the group requires your instructor to validate that you took the class, so please be sure to provide the instructor's name and the course location/dates when you request access.

Lab 14

Mini-Comprehensive Investigation

This page intentionally left blank.

Lab 14 Objectives: Mini-Comprehensive Investigation

- Review evidence from various sources and correlate the findings between them
- Understand the relative strengths and weaknesses of different sources of evidence and when each can be most useful during an investigation
- Use Wireshark's SSL decryption features to provide insight to otherwise impervious communications

This page intentionally left blank.

Lab 14 Takeaways: Mini-Comprehensive Investigation

- NetFlow provides broad overview, path to add'l data
- HTTP server logs lack detail for POST actions
- pcap requires efficient data reduction
  - Encryption is a major hurdle; can't always mitigate
- Correlating evidence may require revisiting findings
  - Events rarely discovered in order they occurred
  - Re-order as needed to maintain chronological awareness
- Many activities leave artifacts of communication
  - Port scanning, brute force attacks, data theft, etc.

This page intentionally left blank.

FOR572.5 | Advanced Network Forensics and Analysis

Encryption, Protocol Reversing, OPSEC, and Intel

Authors:
Phil Hagen, Lewes Technology Consulting, LLC
phil@lewestech.com | @philhagen
Mat Oldham
mat.oldham@gmail.com | @roujisecurity

Capstone Challenge Preparation

This page intentionally left blank.

This page intentionally left blank.

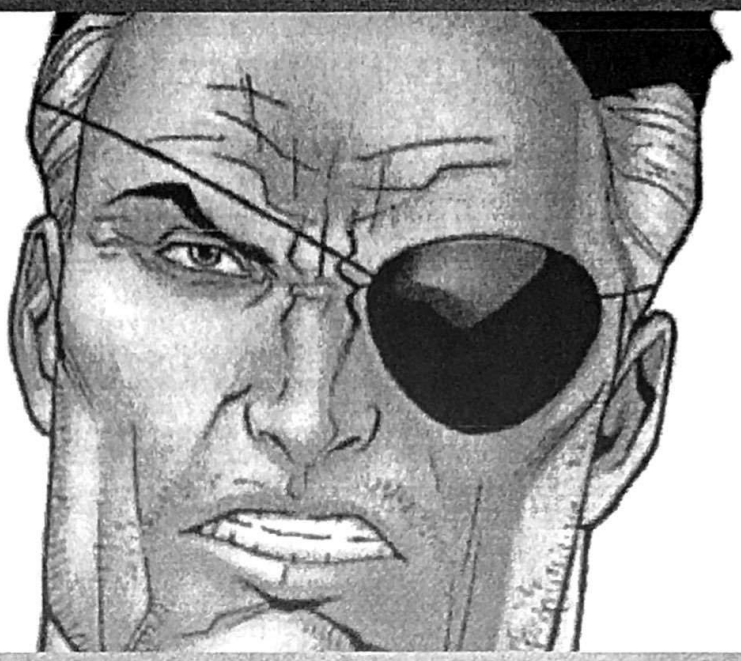
Welcome to Stark Research Labs

- Government-sponsored laboratory
- Specializes in metal alloys and bio engineering
- Recently found the secret to the formula to make VIBRANIUM alloy
- Also a front company for SHIELD
  - Clandestine pseudo-governmental branch with covered agents and programs deployed globally

This page intentionally left blank.

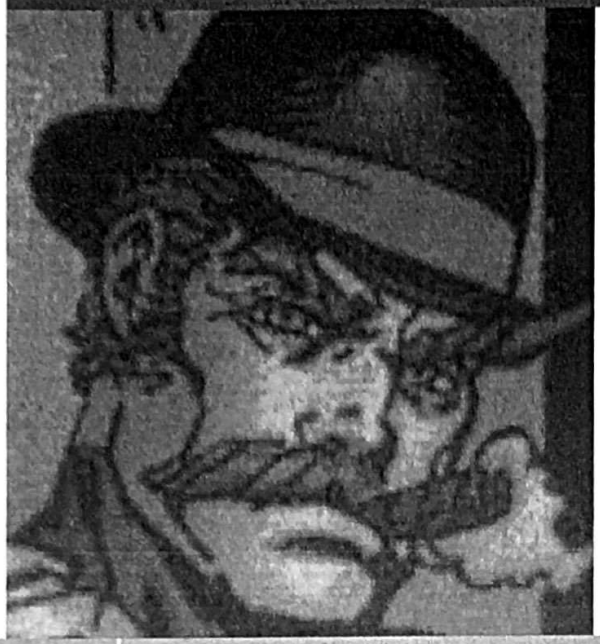
Nick Fury

- Stark Research Labs Director
- Working on STAR FURY
- Twitter: `@NickFury6`
- IP Address: `10.3.58.6`



This page intentionally left blank.

Timothy Dungan

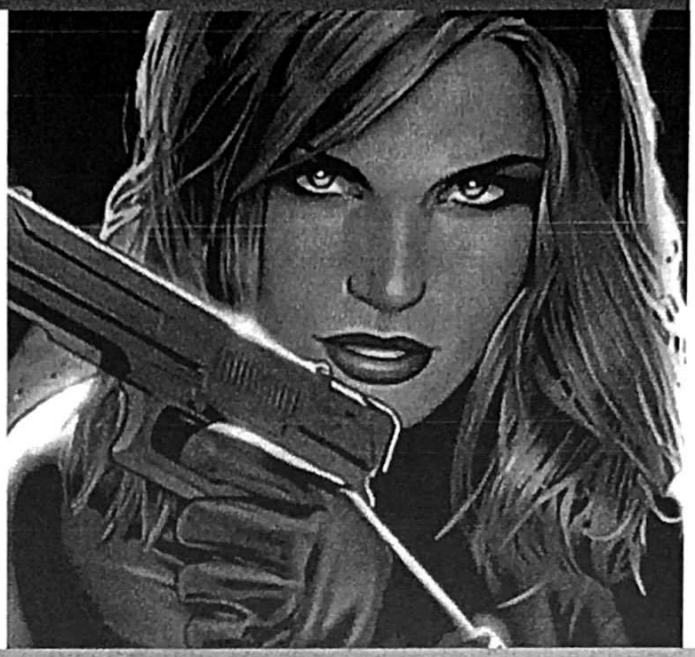


- Stark Research Labs Lead Researcher
- Working on VIBRANIUM Alloy project
- Possible OPSEC slip
- Twitter: `@TimothyDungan`
- IP address: `10.3.58.7`

This page intentionally left blank.

Natasha Romanoff

- Stark Research Labs Personnel Director
- Manages SHIELD's undercover agents
- Twitter: `@RomanoffNatasha`
- IP: `10.3.58.5`

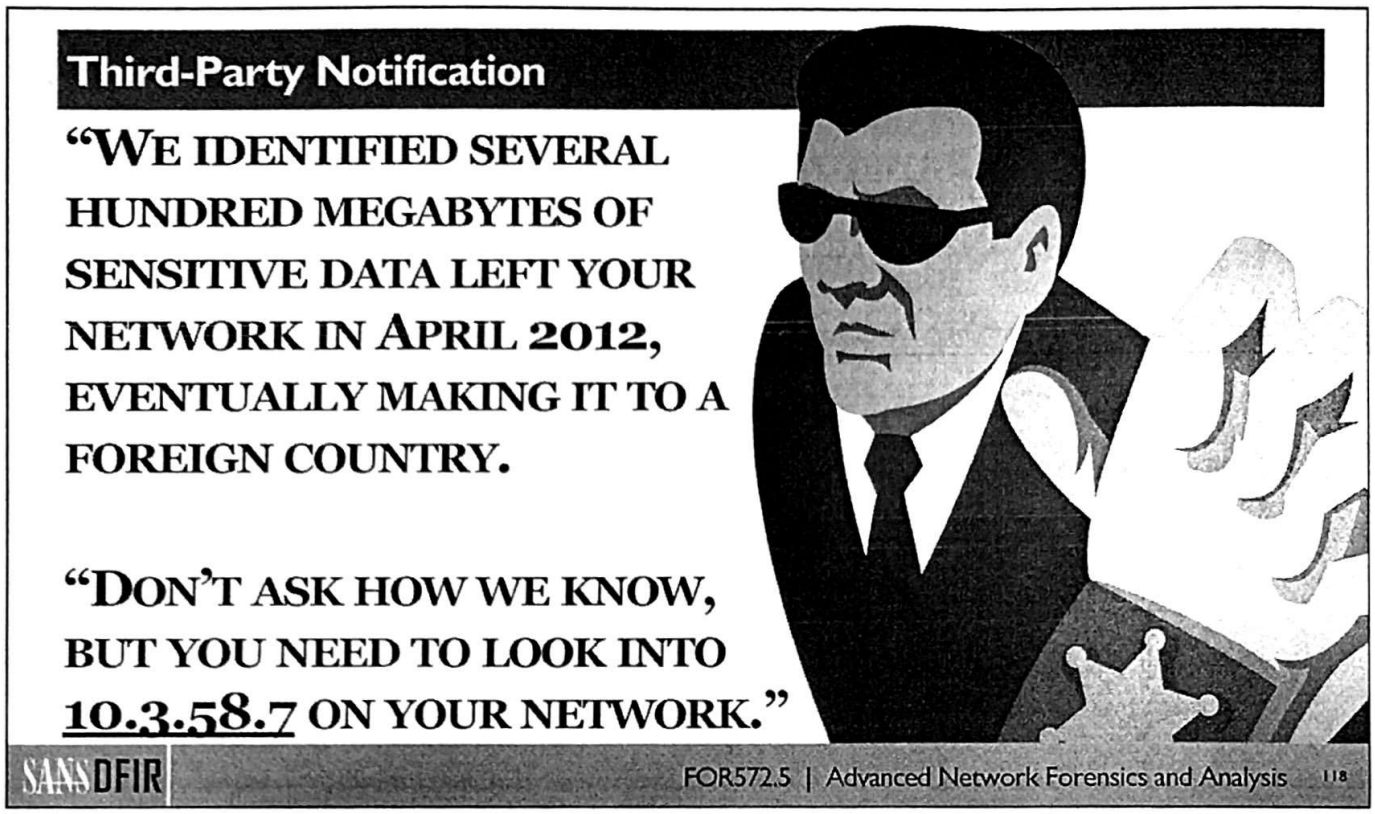


This page intentionally left blank.

Maria Hill

- Stark Research Labs Administrator
- Appointed by the President
- IP: ???????

This page intentionally left blank.



This page intentionally left blank.

Available Evidence

- Full-packet DMZ pcap no longer available
  - Incident occurred before 14-day rolling buffer
- Log data retention policy: Expunge after six months
  - Few/no log-based Artifacts of Communication
- NetFlow from period of attack has been provided
- Security team member remembered she captured traffic during application test at time of incident
  - Isolated ~6GB of internal pcap data from port mirror on "key" network subnet (10.3.58.0/24)

This page intentionally left blank.

Your Mission, Should You Choose to Accept It...

- Did a breach occur?
- How did they breach our network?
- What systems were compromised?
- What was taken?
- Where did they go?
- Who compromised us?
- What malware was used?
- What should we do?

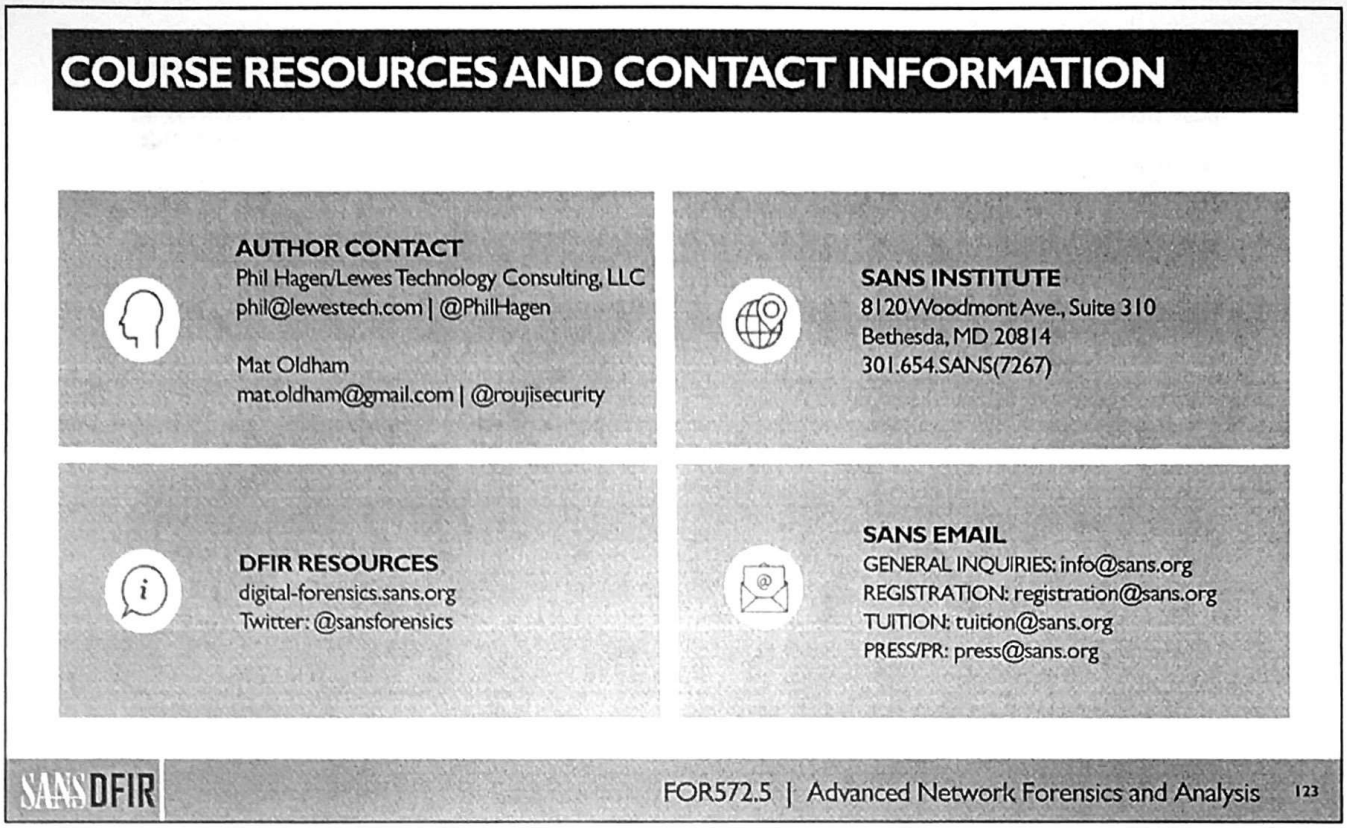
This page intentionally left blank.

The Rules...

- Teams of 3-4 only
  - One team of 5 if absolutely required
- A "choose your own adventure" challenge
  - There is no singular correct solution!
- Use any tools you have—online or locally
- Teams will present findings to the client
  - Maintain timeline and indicators of compromise
  - Identify knowledge gaps and how to address
  - Don't wait until T-0:15:00 to start presentation!

This page intentionally left blank.

This page intentionally left blank.



This page intentionally left blank.

"As usual, SANS courses pay for themselves by Day 2. By Day 3, you are itching to get back to the office to use what you've learned."
Ken Evans, Hewlett Packard Enterprise - Digital Investigation Services

SANS Programs
sans.org/programs

GIAC Certifications
Graduate Degree Programs
NetWars & CyberCity Ranges
Cyber Guardian
Security Awareness Training
CyberTalent Management
Lease Purchase Arrangements
DoDD 8140
Security of Interest for NetSec
Cybersecurity Innovation Awards

Search SANSInstitute

SANS Free Resources
sans.org/security-resources

- E-Newsletters
  NewsBites: Bi-weekly digest of top news
  OUCH!: Monthly security awareness newsletter
  @RISK: Weekly summary of threats & mitigations
- Internet Storm Center
- CIS Critical Security Controls
- Blogs
- Security Posters
- Webcasts
- InfoSec Reading Room
- Top 25 Software Errors
- Security Policies
- Intrusion Detection FAQ
- Tip of the Day
- 20 Coolest Careers
- Security Glossary

SANS Institute
8120 Woodmont Avenue | Suite 310
Bethesda, MD 20814
301.654.SANS(7267)
info@sans.org