Cobalt Strike Analysis and Tutorial: CS Metadata Encryption and Decryption

By Chris Navarrete, Durgesh Sangvikar, Yu Fu, Yanhui Jia, Siddhart Shibiraj

Published: 2022-07-13 · Archived: 2026-04-05 17:47:41 UTC

Executive Summary

Cobalt Strike is commercial threat emulation software that mimics a quiet, long-term embedded actor in a network. This actor, known as Beacon, communicates with an external team server to emulate command-and-control (C2) traffic. Due to its versatility, Cobalt Strike is commonly used as a legitimate tool by red teams – but is also widely used by threat actors for real-world attacks. Different elements of Cobalt Strike contribute to its versatility, including the processes that encrypt and decrypt metadata sent to the C2 server.

In a previous blog, "Cobalt Strike Analysis and Tutorial: CS Metadata Encoding and Decoding," we learned that the encrypted metadata is encoded for an HTTP transaction.

When Cobalt Strike's Beacon "phones home," it encrypts metadata – information about the compromised system – with the RSA algorithm public key and sends it to the Cobalt Strike TeamServer. The TeamServer will use the private key to recover the Beacon plaintext metadata to differentiate the Beacon clients. Also, the AES symmetric key can be extracted from decrypted metadata. The client and server can use the AES key to encrypt and decrypt the further request and response data to finish the C2 traffic communication.

In this blog post, we will detail and demonstrate the data encryption and decryption algorithm, key generation and extraction, metadata encryption and decryption, and metadata schema definitions. One of the interesting components is how the encryption and decryption algorithm works during C2 traffic communication – and why this versatility makes Cobalt Strike an effective emulator that is difficult to defend against.

Data Encryption/Decryption Algorithm

The Cobalt Strike Beacon communicates with the TeamServer using a combination of symmetric (AES) and asymmetric (RSA) encryption key algorithms. The TeamServer will then create a new public/private key combination and store the key pair in a .cobaltstrike.beacon_keys file. The file is stored in the same directory where the Cobalt Strike setup is extracted. If the file already exists, it uses the same key pair.

The asymmetric key algorithm uses RSA/ECB/PKCS1Padding, while the symmetric key algorithm uses the AES/CBC/NoPadding format to encrypt/decrypt the data. The AES algorithm is initialized with a hard-coded initialization vector (IV). The static IV is abcdefghijklmnop.

Figure 1 highlights the C2 traffic between the Cobalt Strike Beacon and the TeamServer.

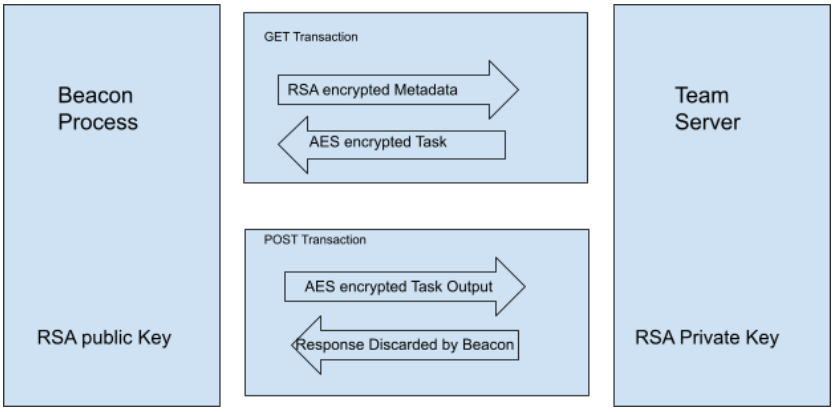

Figure 1. C2 Communication between the Beacon process and the TeamServer.

Metadata Schema Definition

As the name suggests, metadata contains information about the target. The metadata follows a structured format with the 4-byte magic number (0xBEEF) at the start. Figure 2 shows the metadata structure.

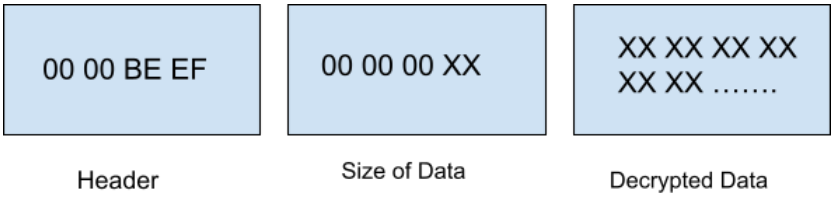

Figure 2. Beacon metadata structure.

The decrypted data is a blob of different information. The structure of the decrypted blob was updated in Cobalt Strike version 4.0, and the Beacon has added more information in the metadata. The size of the data field is 4 bytes long, and this suggests that the author may update the metadata structure in the future. Figure 3 shows the various types of information packed in the metadata. This structure is in accordance with the current implementation.

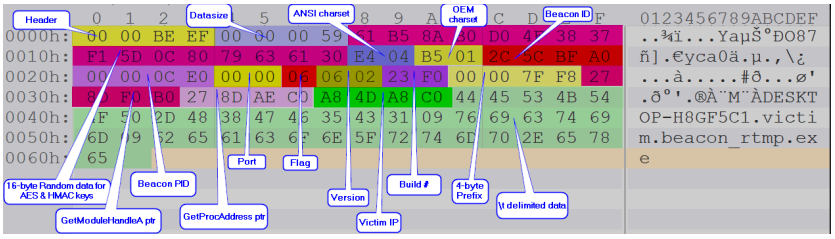

Figure 3. Metadata schema.

Below is the breakdown of the various data fields in the decrypted data in order.

- The first 16 bytes are the random bytes generated by the Beacon and are unique to each process run. The Beacon and TeamServer use these bytes to create the AES key and HMAC key. The process calculates the SHA256 hash of the bytes. The first 16 bytes of the SHA256 are assigned as the AES key for symmetric encryption, and the remaining 16 bytes are the HMAC keys for the message authentication code.
- The next two little endian bytes are decoded as ANSI Charset. For the complete list of charsets, refer to documentation on Code Page Identifiers.
- Two little endian bytes are assigned to OEM Charsets.
- Four big-endian bytes are the Beacon ID, with each Beacon getting a unique ID.
- Four big-endian bytes are the Process ID of the Beacon on the victim's machine.
- Two bytes are decoded as the port.
- One byte decodes as the flag. In the current implementation of Cobalt Strike, the flag value is used to set the architecture (32/64 bit) of the Beacon.
- Two bytes are the Beacon version number. These bytes are converted into a string with a "." inserted between them. Ex: A.B
- Two bytes are decoded as the build version of the Beacon.
- The next 4-byte big endian value is used to prefix the pointers to functions if the architecture of the beacon is 64 bit. These 4 bytes are discarded if the architecture is 32 bit.
- The following two 4-byte values are the pointers to GetModuleHandleA and GetProcAddress. The value of these will help the shellcode to resolve further functions without being imported explicitly. If the Beacon is 64 bit, the earlier values are prefixed and the entire value is stored in a variable.
- Next four bytes are the IP address of the target. The bytes are then converted to an IPv4-readable address.
- Last set of bytes are UTF-8, and the values are delimited by \t. As of the current version, the data is structured as ComputerName\tUserName\tBeaconProcessName

Public/Private Key Generation and Extraction

When the Beacon checks in, it will send the metadata blob encrypted by the RSA public key to the TeamServer. The TeamServer uses the private key to decrypt and recover the plain text metadata and extract the AES key along with other metadata used for further communication. This can prevent a meddler-in-the-middle (MitM) attack and evade detection since the AES key is encrypted by asymmetric key, which is extremely difficult to decipher. Additionally, the C2 communication is encrypted by the symmetric key, making it difficult to find a fingerprint to mark it.

When the TeamServer starts with the profile loaded, it generates a public/private key pair and stores them in the .cobaltstrike.beacon_keys file in the TeamServer root directory if the file doesn't exist.

We can use the key dump Java program shared in GitHub to extract the public/private key. See below for details on how this is done.

1. Public/private key pair stored in .cobaltstrike.beacon_keys as java.security.KeyPair object as Figure 4 shows.

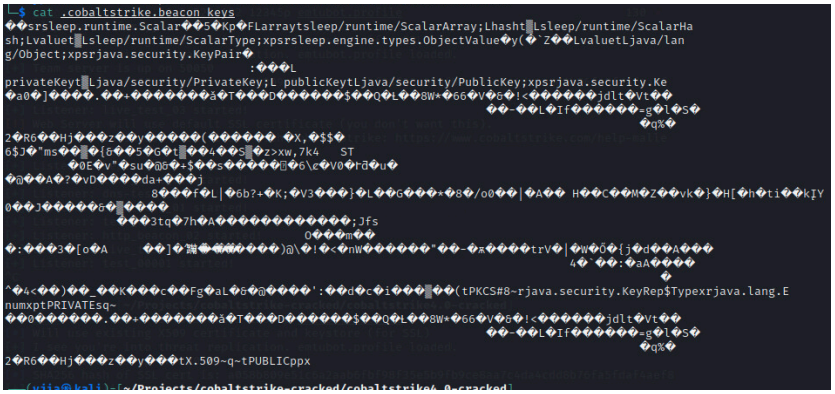

Figure 4. Public/private key pair Java object.

2. Execute the command in Figure 5 to compile and run the Java program in the TeamServer root directory, and then the public/private key pair will be base64 encoded.

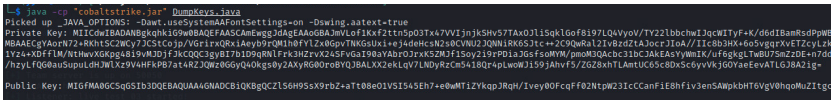

Figure 5. Public/private key.

An Example of C2 Metadata Encryption/Decryption with RSA and AES

In the following example, we will discuss how both encryption and decryption work in the context of Cobalt Strike Beacon's metadata and C2 HTTP traffic communication.

The analysis is of a sample taken from the wild, downloadable directly from VirusTotal (SHA256: 50ea11254f184450a7351d407fbb53c54686ce1e62e99c0a41ee7ee3e505d60c).

The C2 traffic analysis for this example is separated into three sections:

1. Cobalt Strike Beacon Download
2. C2 Beacon Heartbeat
3. C2 Tasks Request and Response (Callback)

These sections contain encryption/decryption analysis that will describe the following process:

1. Decrypt metadata using RSA public/private leaked keys to get the AES and HMAC keys.
2. Use an AES key to decrypt the task data encrypted in the HTTP GET response.
3. Use an AES key to decrypt the task execution result in the HTTP POST request body.

Cobalt Strike Beacon Download

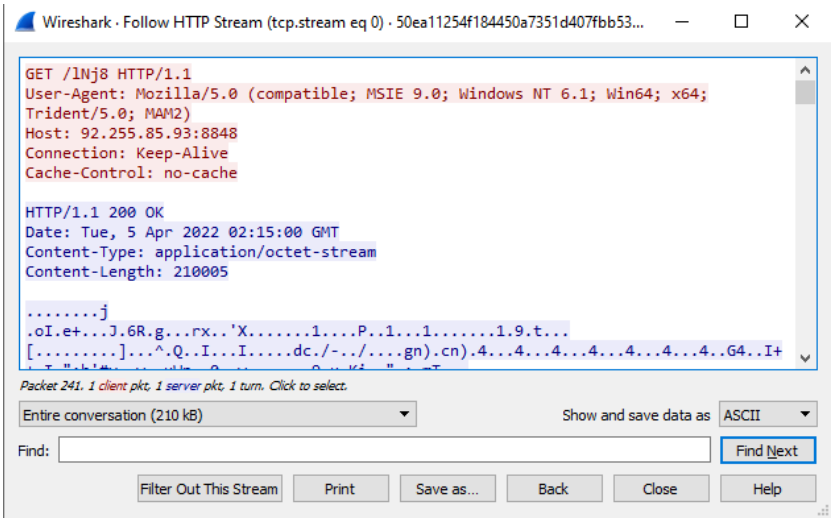

Figure 6. Beacon download

By parsing the configuration using the 1768.py script, it can be shown that the Beacon was generated by a version of Cobalt Strike's software that has leaked RSA private keys.

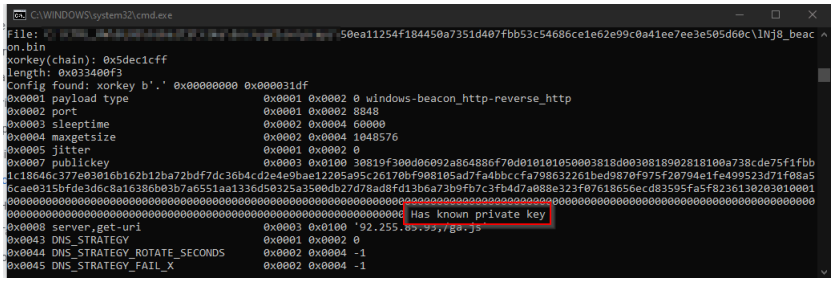

Figure 7. Beacon parsing/private key search.

C2 Beacon Heartbeat

Once the Beacon was downloaded and executed, the next step was to perform a C2 checkin on the TeamServer and to exfiltrate encrypted metadata information about the compromised machine inside of the cookie of the HTTP request.

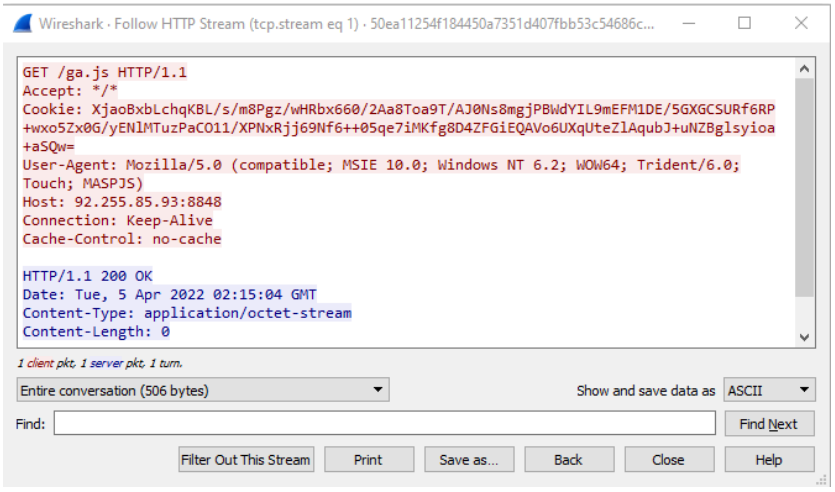

Figure 8. C2 Heartbeat.

In order to decrypt the metadata information, an execution of the cs-crypto-parser.py script (a slightly modified version of the cs-mitm.py script) should be performed and pass the value from the cookie – XjaoBxbLchqKBL/s/m8Pgz/wHRbx660/2Aa8Toa9T/AJ0Ns8mgjPBWdYIL9mEFM1DE/5GXGCSURf6RP+wxo5Zx0G/yENlMTuzPaCO11/XPNxRjj69 – as a first argument.

RSA Decryption

The first task this script performs is a call to the RSADecrypt() function that receives two parameters: 1) The private key, and 2) The encrypted cookie value. Once this task is completed, it decodes and imports the RSA private key and instantiates a new PKCS1 object. Finally, the script calls the decrypt function and passes the encrypted data variable as argument to perform the actual decryption of the ciphertext.

```
def RSADecrypt(key, data):
    oPrivateKey = Crypto.PublicKey.RSA.importKey(binascii.a2b_hex(key))
    oRSAPrivateKey = Crypto.Cipher.PKCS1_v1_5.new(oPrivateKey)
    ciphertext = data
    try:
        cleartext = oRSAPrivateKey.decrypt(ciphertext, None)
```

Figure 9. RSADecrypt function.

The output will show a detailed breakdown of the metadata along with its hex values.

Figure 10. C2 decrypted metadata.

Readers can use the Metadata Schema Definition section of this blog for reference.

C2 Task Request and Response (Callback)

AES and HMAC Keys

The payload is encrypted using AES256 encryption in CBC mode with an HMAC SHA256-keyed hashing algorithm. Since there is access to the RSA private key and the decrypted data, it includes the raw key. This key is 16 bytes long and is located at the eighth byte of the decrypted payload. In the case of this malware sample, the key (hex) is 1a 13 7e 76 f9 15 6a 67 f9 99 af d6 57 64 75 bd. In order to generate the AES and HMAC keys, the SHA256 hash is computed out of the raw key where the first half (16 bytes) is the actual AES key and the second half (16 bytes) is the HMAC key. Figure 11 below depicts such computation by the execution of the crypto-parser script.

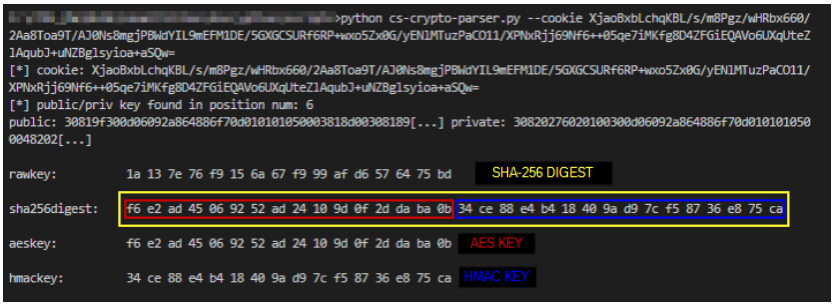

Figure 11. Encryption/decryption hash and keys.

C2 Task Request

Once a C2 channel is established and the checkin action is in place, the Beacon performs a check or any new tasks. In Figure 12, you will see that the task request received a response payload from the TeamServer.

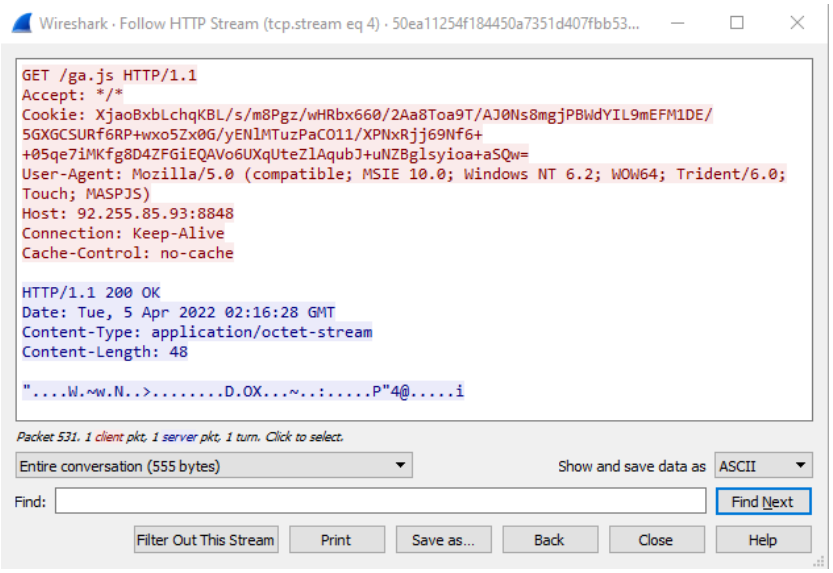

Figure 12. C2 task request (retrieval).

The payload data of 48 bytes is now passed to the script to get decrypted. The last 16 bytes of the encrypted blob is the HMAC Signature that acts as an integrity measure for the request. FIgure 13 below shows the data parsing and decryption of the task payload.

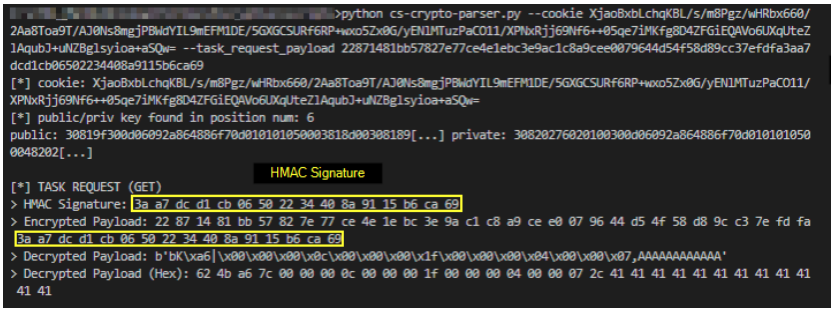

Figure 13. Cobalt Strike task request payload decryption.

The decryption process is handled by the Decrypt() function, which performs the following actions:

1. Receive the encrypted data as parameter.
2. Extract the HMAC signature out of the encrypted payload.
3. Calculate and validate the HMAC signature by using the HMAC key on the encrypted payload.
4. Load the AES key and set the mode (CBC), and its initialization vector (IV).
5. Decrypt the encrypted payload.

```
def Decrypt(self, data):
    if self.aeskey == None:
        return data
    encryptedData = data[:-16]
    hmacSignatureMessage = data[-16:]
    hmacsSgnatureCalculated = hmac.new(self.hmackey, encryptedData, hashlib.sha256).digest()[:16]
    if hmacSignatureMessage != hmacsSgnatureCalculated:
        raise Exception('HMAC signature invalid')
    cypher = Crypto.Cipher.AES.new(self.aeskey, Crypto.Cipher.AES.MODE_CBC, CS_FIXED_IV)
    decryptedData = cypher.decrypt(encryptedData)
    return decryptedData
```

Figure 14. AES / HMAC decryption function.

C2 Task Response

When a Beacon receives and executes a task provided by the C2 server, the results are collected and returned to the TeamServer.

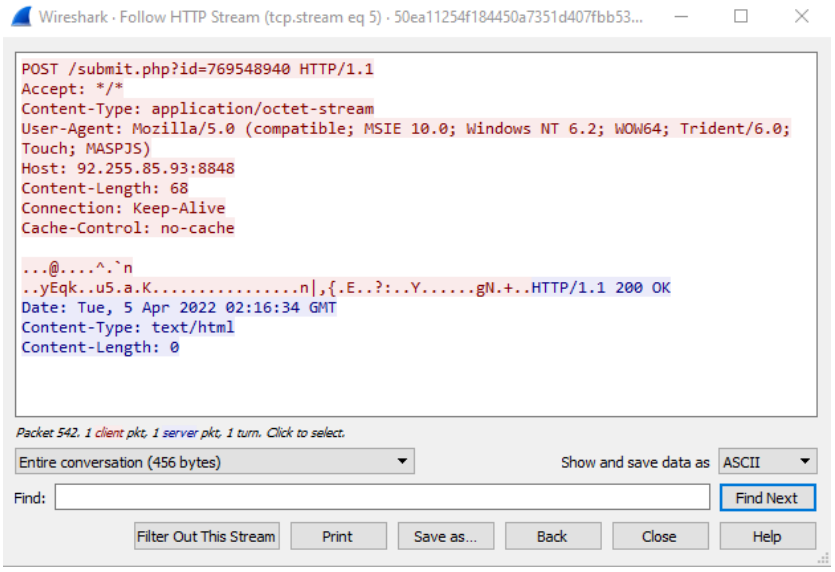

Figure 15. Cobalt Strike task response (data exfiltration).

By using the same Decrypt() function mentioned above, the encrypted payload is also provided. However, for a Cobalt Strike task response, the first four bytes are not considered for decryption, resulting in those bytes being excluded in the data passed to the function.

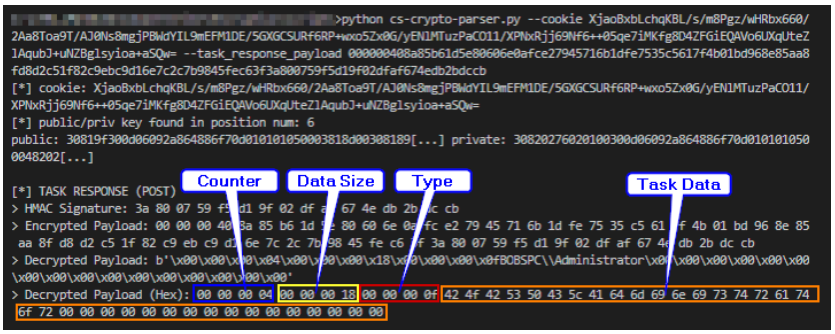

Figure 16. Cobalt Strike task response decryption.

The task data response contained in this request corresponds to the ASCII string BOBSPC\\Administrator, which is the machine and user of the compromised computer.

Conclusion

Cobalt Strike is a potent post-exploitation adversary emulator. The metadata encryption/decryption detailed above are elaborate and are designed to evade security detections. A single security appliance is not well-suited to prevent a Cobalt Strike attack. Only a combination of security solutions – firewalls, sandboxes, endpoints and the appropriate software to integrate all these components – can help prevent an attack of this nature.

Palo Alto Networks customers receive protections from attacks similar to those by Cobalt Strike with the help of:

1. Next-Generation Firewalls (NGFW) with Threat Prevention signatures 86445 and 86446 identify HTTP C2 requests with the base64 metadata encoding in default profiles. Advanced Threat Prevention has new capabilities to detect these attacks without the need for signatures.
2. WildFire, an NGFW security subscription, and Cortex XDR identify and block Cobalt Strike Beacon.
3. AutoFocus users can track this activity using the CobaltStrike tag

Indicators of Compromise

CS Samples

- 50ea11254f184450a7351d407fbb53c54686ce1e62e99c0a41ee7ee3e505d60c

CS Beacon Samples

- /lNj8
  - SHA256 Hash:

- e712d670382ad6f837feeb5a66adb2d0f133481b5db854de0dd4636d7e906a8e

CS TeamServer IP addresses

- 92.255.85[.]93

Additional Resources

- Cobalt Strike Training
- Cobalt Strike Malleable C2 Profile
- Cobalt Strike Decryption with Known Private Key
- Cobalt Strike Analysis and Tutorial: How Malleable C2 Profiles Make Cobalt Strike Difficult to Detect
- Cobalt Strike Analysis and Tutorial: CS Metadata Encoding and Decoding
- Cobalt Strike Attack Detection & Defense Technology Overview

---