



CS-501

Introduction to Malware, Threat Hunting &
Offensive Capabilities Development

Lecture 9: C2 Dev, Encoding, Stealers

Implant Functionality

Registration

Sleep

Tasking

Implant Functionality

Registration: Authenticate ourselves to the server. We might want to prevent replay attacks

Sleep: The implant goes dormant for some fixed or random amount of time while receiving commands.

Tasking: The implant checks in with the server and pulls down tasks

Execute: The agent executes all tasks, and sends the response to the server

Example from Last time

RPC is RESTFUL (JSON based) where functionality is divided by HTTP Endpoint

C2 Advice:

When developing covert C2 Channels, you generally want to accomplish the following:

- 1) C2 traffic should blend in with host traffic. Your comms should not look anomalous compared to legitimate traffic
- 2) Be kept to only what is necessary. The more noise you make, the more likely you are to get caught. Every connection carries risk, and you should do your best to mitigate that.

Other examples

The implant signals the action they wish to perform based on HTTP params, cookies, Headers, POST payload data, HTTP Method...etc

Anywhere you can encode data, you can encode implant functionality

You could even use Time to signal agent commands

JSON

- Kind of annoying to parse.
- RESTFUL traffic is common to see egressing from networks, especially for web traffic
- Plenty of libraries, some heavier than others

Dealing with Strings

OK so I lied. Strings in windows are a bit more complicated than

8 bit \leftrightarrow char \leftrightarrow UTF-8

16 bit \leftrightarrow wchar_t \leftrightarrow UTF-16

A full treatment of encodings is beyond the scope of this class.

While you should still prefer the UTF-16 version of most Windows API Functions, there are several situations where you should use 8-bit characters, std::string and the ANSI version of functions.

Dealing with HTTP traffic

HTTP traffic is by default UTF-8. You can send UTF-16 encoded traffic so long as you configure your listener to use Unicode as well

This unfortunately, requires double the space, and makes your HTTP requests larger.

One thing you can do is encode UTF-16 values (computer names, file lists...etc) and include it in a UTF-8 JSON payload. Then simply indicate to the server that the value needs to be decoded as utf-16 and you are good to go

The Absolute Nightmare, that is Code Pages

OK so I lied. Strings in windows are a bit more complicated than

8 bit \leftrightarrow char \leftrightarrow UTF-8

16 bit \leftrightarrow wchar_t \leftrightarrow UTF-16

While you should still use

While the kernel deals exclusively with UTF-16 strings, many legacy applications

Legacy applications, rely on CodePages set based on the Locale.

This scheme can be non standard, and is a serious pain in the ass to deal with

Dealing with binary Data

Most use cases for implants require sending and receiving binary data to servers (think uploading/downloading arbitrary files)

We need a way to send this data to the C2

When dealing with raw bytes, we need a suitable container to interact with them.

- C: unsigned char *, unsigned char[]
- C++: std::vector<>
 - BYTE, unsigned char, uint8_t...etc
 - Std::vector is guaranteed to allocate data in a contiguous block in memory. This makes it suitable for dealing with raw bytes
 - It also comes with many handy built in functions to make manipulation easy. Vector's also make memory management easy.

Useful Concepts: Compression

Lossless Compression: Take byte sequence of length N and reversibly map it to a byte sequence of length $M < N$

If we are going to send and receive data from the C2, we should always compress data.

Interview Question

When sending data to the C2, should we Compress then encrypt, encrypt then compress, or does it even matter?

Always compress, then encrypt

Compression algorithms fail to compress data when it has high entropy (i.e., random looking data)

Informally, encrypting data tries to make it look random

Encrypt→ compress leads to data that is basically the same size (if not larger!) than its pre-compressed counterpart

Can also be useful for obfuscating data, even if the the compression function doesn't compress our data. I.e., good luck writing YARA rules for data that is randomly composes encryption, compression, encoding...etc

Useful Concepts: Binary Data encoding

How do we send binary data to the server?

We can just send it in the HTTP body, but this is

1) annoying to parse, 2) suspicious

By encoding data, we can continue using an RPC that doesn't support raw Binary (JSON) by converting it into an encoded format

Hex (base 16) Encoding

- Hex encoded inputs:
- Python: `bytes.hex()`, `bytes.fromhex(hex_data)`
- Encodes raw bytes as a string
- Interview question: How many bytes do you need to encode a fixed length byte sequence?
- Encodes bytes as pairs of hex (base 16) integers

Base64 Encoding

- Encodes data using base 64 integers, possibly with padding
- Python Example

Base64: Python

Exercise: Base64 Encoding/Decoding

Writing an HTTP Client in python

I highly recommend that any functionality you implement in C++, you at least outline with python. This can help you easily determine if the C2 is behaving incorrectly or if it is the implant

Python requests library has all the functionality you could possibly need along with the json library

Writing an HTTP Client with Windows

PAIN. Even Windows hates it.

WinInet is not supported in a service or an IIS application

02/27/2020 • 2 minutes to read •



① Important

The Internet Explorer 11 desktop application will be retired and go out of support on June 15, 2022 (for a list of what's in scope, see the [FAQ](#)). The same IE11 apps and sites you use today can open in Microsoft Edge with Internet Explorer mode. Learn more [here](#).

The Microsoft Win32 Internet Functions (exported from WinInet.dll) are not supported when run from a service or an Internet Information Server (IIS) application (also a service). This article discusses using the WinInet.dll in a service or in IIS applications.

Original product version: Internet Explorer

Original KB number: 238425

Your options for HTTPs: Native C++

URLMON: Allows for smaller code. Can be made proxy aware, and is fairly easy to use. Not as customizable as other options though.

WinSock + Custom HTTP client: Good luck handling proxies. You can, but it is pain++.

WinHttp: Supports reading default proxy. Requires a lot of setup code to use. Is agnostic to modern proxies that require authentication via tokens.

WinInet: Best choice, but still janky. Supports reading default proxy and authenticating the user if required.

Your Options: Libraries

Libcurl: Probably the most robust HTTPs client out there today. Easy to use, reliable, and fast.

Others to consider:

Boost/HTTP

Pros:

Usually easier to use and more reliable

Cons:

If the library doesn't exist on the target, you need to either download it, statically link it.

This can make the binary much, much larger. Especially if the client requires OpenSSL (which it probably does). Even mbedTLS/WolfSSL add a few megabytes to the binary.

Other Options

Make a process that reads request information from a named pipe. Use C# HTTP Client that leverages .NET framework. Also has the nice property that it decouples the process doing “sketchy” post exploitation tasks and the process that communicates with the C2.

Use JScript/VBA (uses WinHttp under the hood)

WinRT (Haven't actually tried this myself, but allegedly you can!)

Stealers

They steal stuff.

More formally, stealers in this class will refer to any malware that is

- 1) Part of a Post Exploitation task
- 2) Uploads sensitive victim data to a C2 including but not limited to
 - a) Browser data: Cookies, history, passwords
 - b) Secrets: password manager databases, ssh keys, RDP profiles, Crypto Wallet Private keys, Certificates...etc
 - c) Documents



Example Stealers:

- Racoon
- Azorult
- Vidar
- Loki

Interview Question:

You come across malware that is using a statically linked Sqlite client. What kind of malware is it most likely?

Valid answers:

It is probably a stealer targeting browsers.

More generally, it is doing something that requires interacting with a SQLite database

The most common targets for commodity malware are browsers! Why?

Stealers: Browsers

Most browsers (firefox, Chrome, Edge, Opera, Brave...etc) store all of the user data in an encrypted SQLite database

The path for that database is predictable, and can be read

These values include cookies, usernames, passwords, autofill data, history, ...etc

Luckily, enough people complained that the values were stored in plaintext, and most browsers encrypt the values in the database! So we are SOL right?

Windows: Doing weird stuff with crypto since 1985

Time permitting, we will discuss the Data Protection API (DAPI) in greater depth, but for now, a few important points:

- Most browsers will use the DPAPI to encrypt symmetric keys used to decrypt passwords
- The DAPI is Token based→ ergo if the browser is used by User “John Doe”, then any processes with the “John Doe”’s token can decrypt values that were encrypted with the DAPI that targeted that user’s token
- In other words, if we trick John into running our program, we can recover his encrypted browser passwords and decrypt them!

How to Loot chrome (old)

The data is stored at C:\Users\User\AppData\Local\Google\Chrome\User Data\default>Login Data

The value itself is encrypted with the DPAPI, and can be easily recovered by calling CryptUnprotectData from a process that has the same token as the logged in user.

To do this, we must parse the SQLite database and decrypt each value

How to Loot Chrome (new)

Chrome stores the secret key that was used to encrypt passwords and other data inside of a configuration file

C:\Users\<User>\AppData\Local\Google\Chrome\User Data\Local State

This is a JSON file that contains the secret key, that is encrypted with the DPAPI.

To decrypt the values in the SQLite database, we must

- First parse the encrypted key
- Second, decrypt the key with the DPAPI
- Finally, we decrypt the data inside of the SQLite database

Encryption Libraries

Python: pycryptodome, cryptography (preferred)

C: To name a few: OpenSSL (Pain), Libsodium, NACL, LibHydrogen, Monocypher.

Windows: Crypto API (Sus)

When it comes to crypto, you probably want to use someone else's implementation. We will talk more about this next week.

Read the Documentation!

CryptUnprotectData function (dpapi.h)

12/05/2018 • 3 minutes to read

The **CryptUnprotectData** function decrypts and does an [integrity](#) check of the data in a [DATA_BLOB](#) structure. Usually, the only user who can decrypt the data is a user with the same logon [credentials](#) as the user who encrypted the data. In addition, the encryption and decryption must be done on the same computer. For information about exceptions, see the Remarks section of [CryptProtectData](#).

Syntax

C++



```
DPAPI_IMP BOOL CryptUnprotectData(  
    DATA_BLOB          *pDataIn,  
    LPWSTR              *ppszDataDescr,  
    DATA_BLOB          *pOptionalEntropy,  
    PVOID               pvReserved,  
    CRYPTPROTECT_PROMPTSTRUCT *pPromptStruct,  
    DWORD               dwFlags,  
    DATA_BLOB          *pDataOut  
);
```

Example: Using Python to Recover Passwords

Demo

Other Useful Files

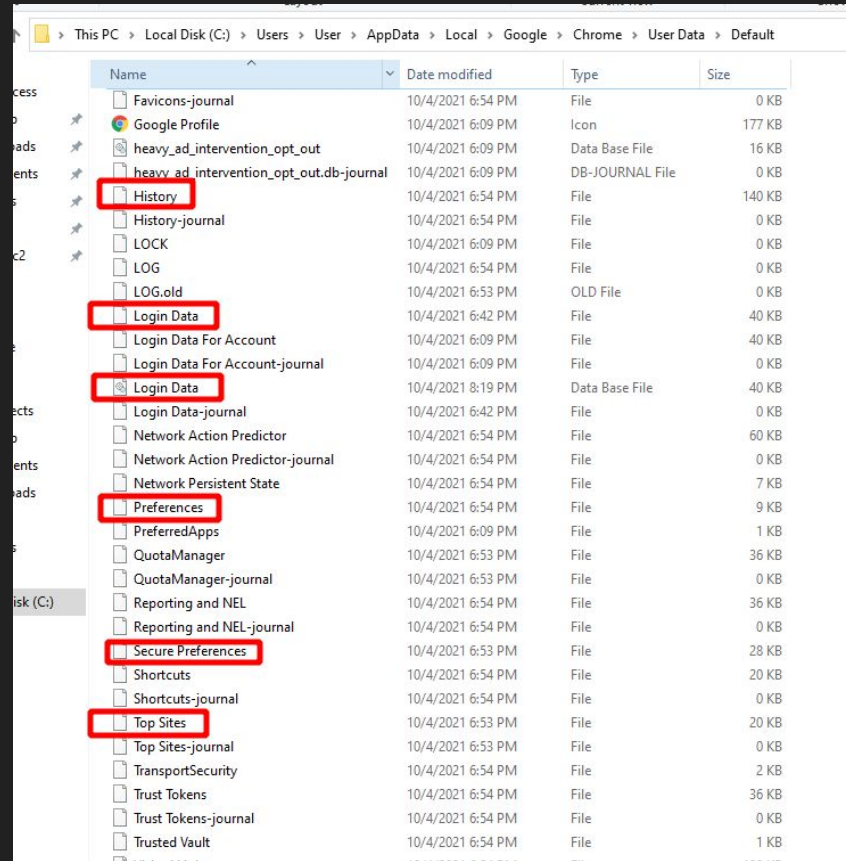
History

Cookies

Top Sites

Preferences

Secure Preferences



This screenshot shows the Windows File Explorer window for the path: This PC > Local Disk (C:) > Users > User > AppData > Local > Google > Chrome > User Data > Default. The table below represents the contents of this folder, with the files highlighted in the image marked with a red box in the 'Name' column.

Name	Date modified	Type	Size
Favicons-journal	10/4/2021 6:54 PM	File	0 KB
Google Profile	10/4/2021 6:09 PM	Icon	177 KB
heavy_ad_intervention_opt_out	10/4/2021 6:09 PM	Data Base File	16 KB
heavy_ad_intervention_opt_out.db-journal	10/4/2021 6:09 PM	DB-JOURNAL File	0 KB
History	10/4/2021 6:54 PM	File	140 KB
History-journal	10/4/2021 6:54 PM	File	0 KB
LOCK	10/4/2021 6:09 PM	File	0 KB
LOG	10/4/2021 6:54 PM	File	0 KB
LOG.old	10/4/2021 6:53 PM	OLD File	0 KB
Login Data	10/4/2021 6:42 PM	File	40 KB
Login Data For Account	10/4/2021 6:09 PM	File	40 KB
Login Data For Account-journal	10/4/2021 6:09 PM	File	0 KB
Login Data	10/4/2021 8:19 PM	Data Base File	40 KB
Login Data-journal	10/4/2021 6:42 PM	File	0 KB
Network Action Predictor	10/4/2021 6:54 PM	File	60 KB
Network Action Predictor-journal	10/4/2021 6:54 PM	File	0 KB
Network Persistent State	10/4/2021 6:54 PM	File	7 KB
Preferences	10/4/2021 6:54 PM	File	9 KB
PreferredApps	10/4/2021 6:09 PM	File	1 KB
QuotaManager	10/4/2021 6:53 PM	File	36 KB
QuotaManager-journal	10/4/2021 6:53 PM	File	0 KB
Reporting and NEL	10/4/2021 6:54 PM	File	36 KB
Reporting and NEL-journal	10/4/2021 6:54 PM	File	0 KB
Secure Preferences	10/4/2021 6:53 PM	File	28 KB
Shortcuts	10/4/2021 6:54 PM	File	20 KB
Shortcuts-journal	10/4/2021 6:54 PM	File	0 KB
Top Sites	10/4/2021 6:53 PM	File	20 KB
Top Sites-journal	10/4/2021 6:53 PM	File	0 KB
TransportSecurity	10/4/2021 6:54 PM	File	2 KB
Trust Tokens	10/4/2021 6:54 PM	File	36 KB
Trust Tokens-journal	10/4/2021 6:54 PM	File	0 KB
Trusted Vault	10/4/2021 6:54 PM	File	1 KB

Questions?