CobaltStrike External C2 via Microsoft Teams

Leveraging Microsoft Graph API For CobaltStrike Beacon Communications

WHOAMI

- Alex Reid
- OSCP, OSEP, RTJC
- Red Teamer and Tool Developer

- ~2.5 years in cyber / red teaming / development
- 2x contributor to CobaltStrike Community Kit
- Interested in Windows malware development and exploitation
- Primarily develop in C, work in Python3 / Powershell when the opportunity arises
- GitHub: https://github.com/Octoberfest7

What is External C2?

- “External C2 is a specification to allow third-party programs to act as a communication layer for Cobalt Strike’s Beacon payload”[1]
- Abstraction of the network portion of CobaltStrike
  - Keep the CobaltStrike GUI, keep all of the features and functions of Beacon, change how Beacon data gets to the Teamserver.
- External C2 goes far beyond traffic customization possible in Malleable Profile
  - MP allows operator to obfuscate beacon data in GET/POST requests to an attacker controlled IP/Domain (set in Listener)
  - External C2 goes deeper, leveraging a third-party platform (usually their API) to transmit Beacon data. Too complex for MP, requires custom tooling

1. https://hstechdocs.helpsystems.com/manuals/cobaltstrike/current/userguide/content/topics/listener-infrastructue_external-c2.htm

Why External C2?

- The “best” solution is having a custom in-house C2
  - Commercial and/or public C2’s are highly signatured
  - But developing a mature in-house C2 is VERY costly both in time and resources
- External C2 allows deep customization of the network layer while leveraging a full featured, mature implant.
- Customization of the network layer of C2 can pay significant dividends
  - Domain reputation
  - Endpoint telemetry
  - Network proxy’s and whitelists

A Quick Note on Other's Work…

- Lots of examples of External C2 and related work publicly available
- F-Secure / WithSecureLabs C3[2]
  - Framework to leverage different communications channels
    - Mattermost, Asana, GitHub, Dropbox, JIRA, Discord, GoogleDrive, Slack, EWS Tasks, OneDrive 365, MSSQL, UNC Share File, LDAP, Printer Jobs
- Boku7 azureOutlookC2[3]
  - Utilizes Microsoft Graph API for C2 via Outlook Mailbox, but not an External C2; standalone implant

2. https://labs.withsecure.com/tools/c3
3. https://github.com/boku7/azureOutlookC2

Why Teams?

- Lots of third-party apps out there, why Teams?
  - Want a communication channel common to as many organizations as possible; Slack, Mattermost, GitHub are all popular but may be out of place depending on target
- An organization may not use Teams, but they almost assuredly use Windows
  - Windows has all kinds of processes, both background and user-driven, that frequently communicate with Microsoft owned domains/resources
    - This makes it very hard for organizations to block or discriminate against HTTPS traffic headed to Microsoft owned resources; very easy to break things.
    - Abuse this inherent trust in order to get C2 traffic out of the network without raising suspicion
- Avoids lots of problems encountered by conventional C2 related to domain reputation, network proxies, domain whitelisting, etc.

What is Graph API?

- "Microsoft Graph is a RESTful web API that enables you to access Microsoft Cloud service resources. After you register your app and get authentication tokens for a user or service, you can make requests to the Microsoft Graph API."[4]
- Key Point: This project is NOT leveraging or impacting a target's Teams environment or Azure AD account! All calls to Graph API are made to an attacker-owned AAD/O365 instance!
- Graph allows you to interact with far more than Teams; Outlook, SharePoint, OneDrive, Excel, etc.
  - This project happens to send Beacon traffic via Teams messages, but it's the Graph API being leveraged to do so; instead of Teams, could have used different API's in the same framework to send C2 traffic via Outlook drafts a la Boku7's azureOutlookC2.
- External C2 traffic is sent to graph.microsoft.com; NOT teams.microsoft.com

4. https://learn.microsoft.com/en-us/graph/use-the-api

Graph API and Threat Actors

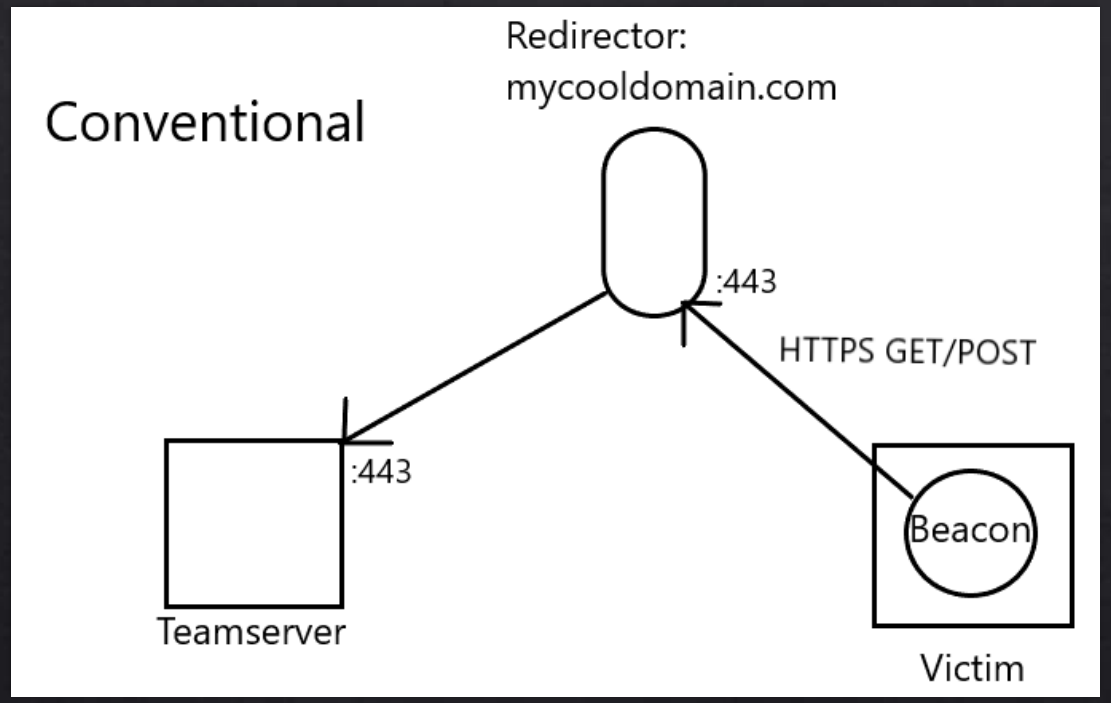
- Graph API malware has been utilized by and attributed to nation-state sponsored APT's
- BLUELIGHT
  - APT 37 / InkySquid / ScarCruft – North Korea[5]
  - Custom C2, utilizes Graph to communicate with OneDrive
- Graphite
  - APT 28 / Fancy Bear – Russia[6]
  - Custom C2, utilizes Graph to communicate with OneDrive

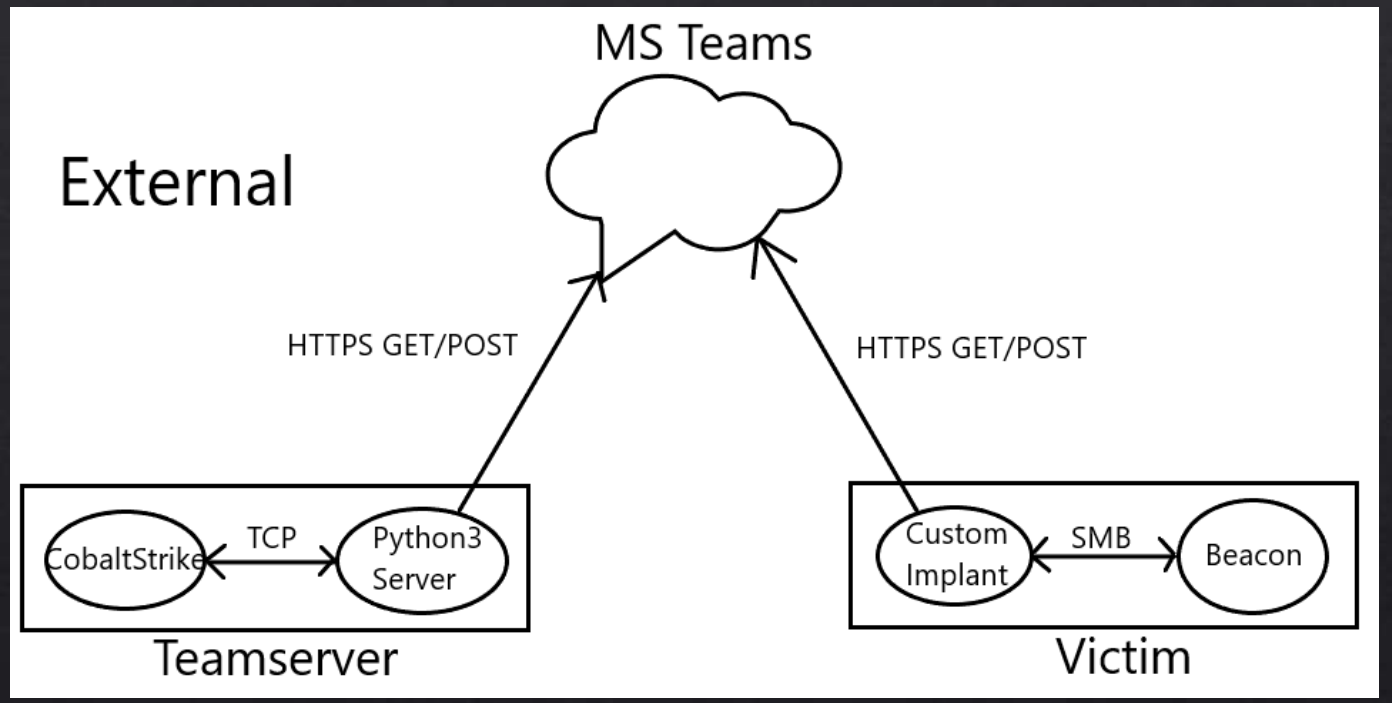
5. https://www.volexity.com/blog/2021/08/17/north-korean-apt-inkysquid-infects-victims-using-browser-exploits/
6. https://www.trellix.com/en-gb/about/newsroom/stories/research/prime-ministers-office-compromised.html

Key Considerations

- We are now leveraging a third-party application/service for C2
  - Attacker's no longer control the "redirector/webserver"
  - Cannot implement IPTables / Apache mod_rewrite / otherwise restrict access to attacker-environment in the same way
    - Most services will use an auth token of some kind in order to utilize API; implant must contain ID's/secrets/tokens in order to communicate with service. If defenders get your implant, they can authenticate to your environment in the third-party application!
  - Third-party likely rate-limit requests; you probably aren't going interactive with your Beacons…
- Security of client data
  - Data is encrypted by Beacon before transmit, but are you comfortable with customer data residing on infrastructure you don't own?
- Asynchronous communications
  - Beacon and Teamserver are no longer synched by Beacon check-in
    - Teamserver must reach OUT to third-party app in order to check for Beacon traffic + send response
    - Another reason you probably aren't going interactive with your Beacons

Conventional VS External Illustrated

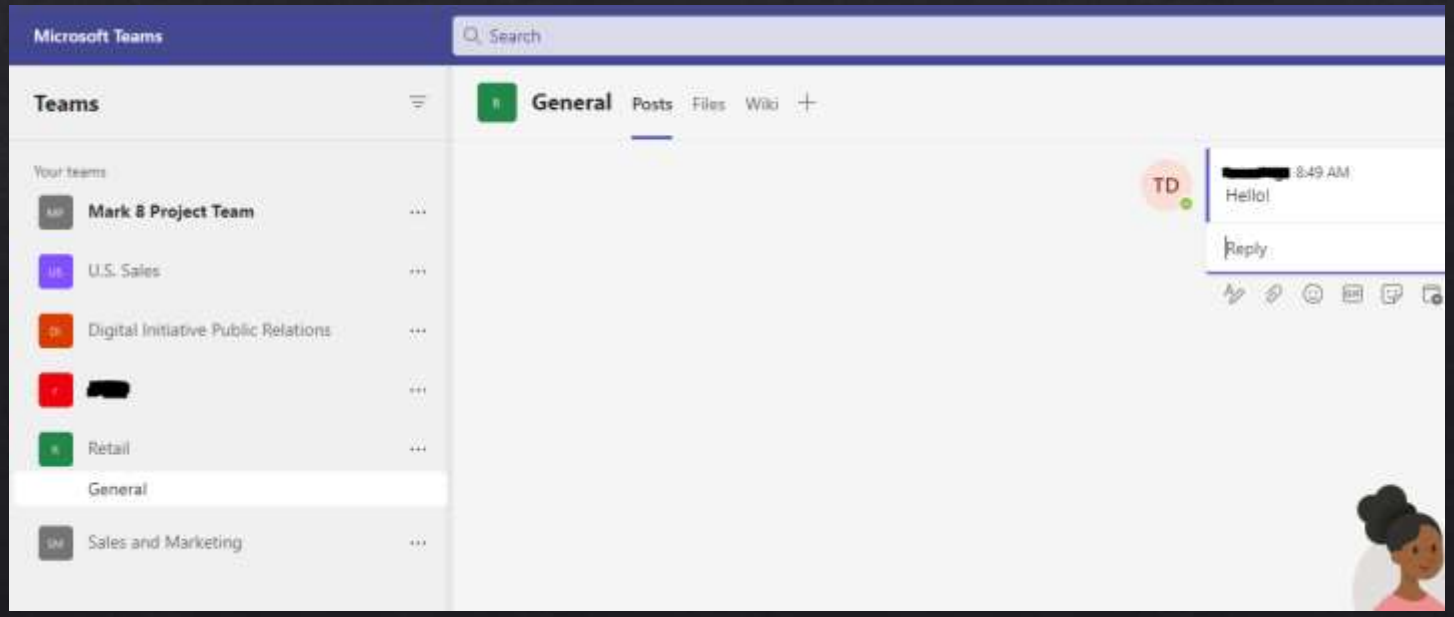




Excuse the MS Paint artwork…

Third Party App Requirements

- What are the requirements of a third-party app to be a good fit for External C2?
  - Receive transmissions from server/implant and be able to differentiate between them
  - Handle multiple simultaneous Beacon connections
  - Capable of transmitting large quantities of data (file upload/download, large command output)
  - Require some kind of authentication / prevent unwanted access to our instance of app



Organizational Structure in Teams

- Microsoft Teams Instance
  - Team A – Mark 8 Project Team
  - Team B – U.S. Sales
  - Team C – Retail
    - Channel 1 – General
    - Channel 2 – Store A
      - Message 1 – From User 1
      - Message 2 – From User 2
      - etc
    - Channel 3 – Store B

- How does Teams organize data?
- How can our Beacon model fit into the Teams data structure?
- Team level – An operation / target environment
- Channel level – A Beacon
- Message level – A check-in from server/implant

- Can have many channels per team
- Can send a lot of messages in each channel

Infrastructure

- We need an attacker-owned/controlled AAD tenant
  - Will need a plan that includes O365 or M365 access in order to use Teams
    - Microsoft plans are horribly complex/confusing… do the research into what features you actually need
  - Implant will be sending Teams messages AS someone; need to make/use user accounts
  - Need to create an Enterprise Application to utilize Graph API and send requests
    - Have to grant appropriate rights to App in order for it to use certain API's
    - learn.microsoft.com has fantastic documentation on Graph API
  - Lots of little settings to tweak and configure. Microsoft documentation makes it all possible, just takes some googling and some work.
- External C2 requires a pretty extensive commitment; hard to automate components of the infrastructure setup
  - Creating Tenant, retrieving data that goes into Server / Implant
  - Even more if you want to go non-attrib and bounce all traffic from attacker through other cloud infra before it touches Microsoft…

Application Permissions

Send chatMessage in a channel or a chat

Article • 09/30/2022 • 35 minutes to read • 7 contributors

Namespace: microsoft.graph

Send a new chatMessage in the specified channel or a chat.

> **Note:** We don't recommend that you use this API for data migration. It does not have the throughput necessary for a typical migration.

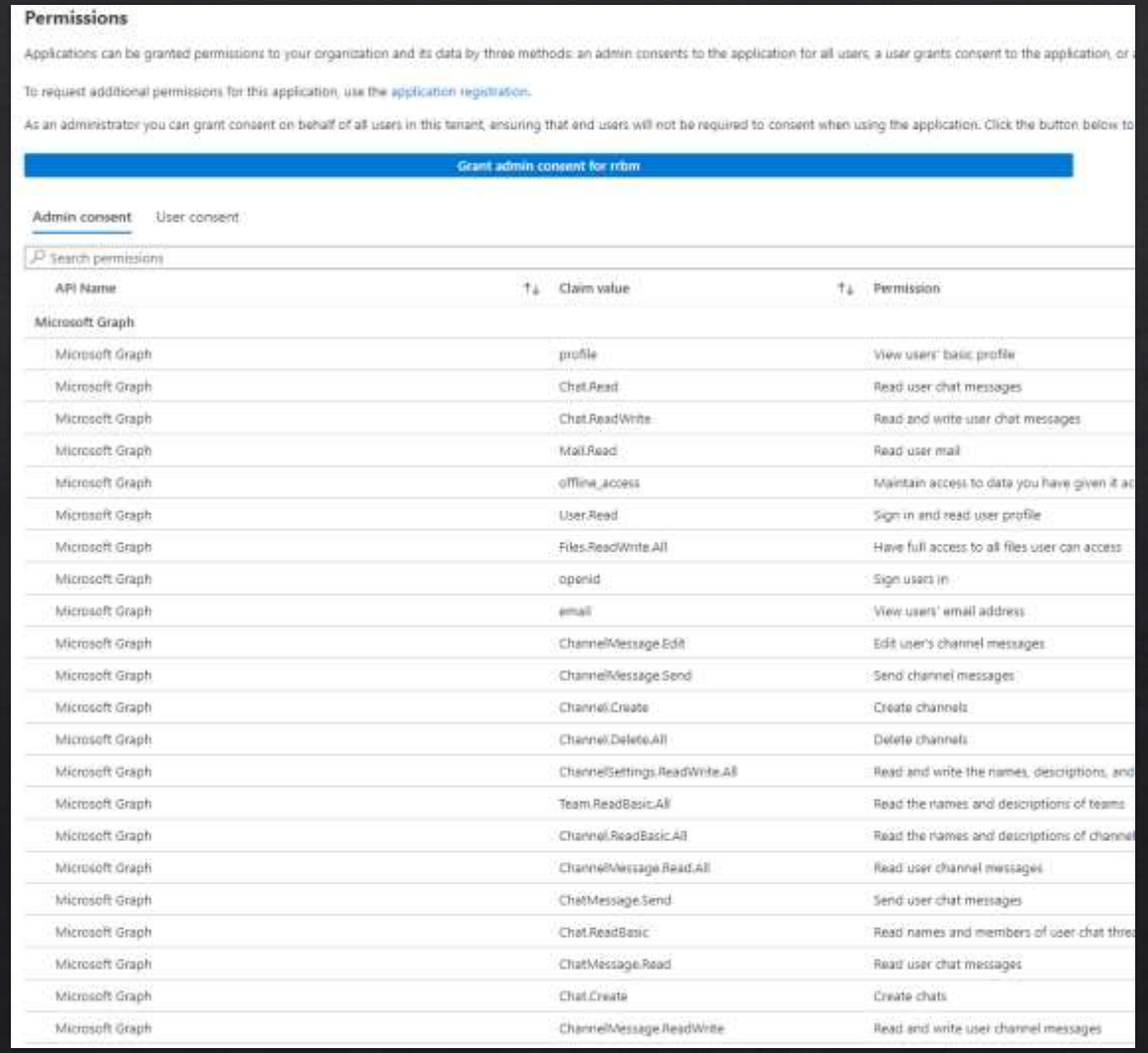
> **Note:** It is a violation of the terms of use to use Microsoft Teams as a log file. Only send messages that people will read.

Permissions

One of the following permissions is required to call this API. To learn more, including how to choose permissions, see Permissions.

Permissions for channel

Permission type	Permissions (from least to most privileged)
Delegated (work or school account)	ChannelMessage.Send, Group.ReadWrite.All**
Delegated (personal Microsoft account)	Not supported.
Application	Teamwork.Migrate.All, ChannelMessage.ReadWrite.All



Implant and Server Overview

Server

- Runs on attacker TS box
- Python3 - Easy to write, no env restrictions
- Interpreter/handler between Teams and TS
  - Authenticate to Teams
  - Retrieve messages from Teams
  - Parse messages
  - Send data to TS -> receive response from TS
  - Format TS data into Teams messages
  - Send TS response to Teams

Implant

- Runs on victim machine
- C - utilizes WinApi, standard malware language
- Interpreter/handler between Teams and Beacon
  - Authenticate to Teams
  - Retrieve messages from Teams
  - Parse messages
  - Send data to Beacon -> receive response from Beacon
  - Format Beacon data into Teams messages
  - Send Beacon response to Teams

Authentication

- How do Server / Implant authenticate and send messages to Teams?
- Graph API utilizes OAuth 2.0
- User authenticates to AAD and receives an Access token
  - Access tokens are short-lived (~1 hour)
  - Access token is required to send/retrieve data from Teams
- User can also request a Refresh token
  - Refresh tokens are long-lived (~90 days)
  - Refresh token can be used to request new Access token
- Embed Refresh token in Server / Implant
  - On runtime, fetch new Access token using Refresh token
  - Every x minutes (before Access token expires), request new Access token -> repeat

Implant Secrets

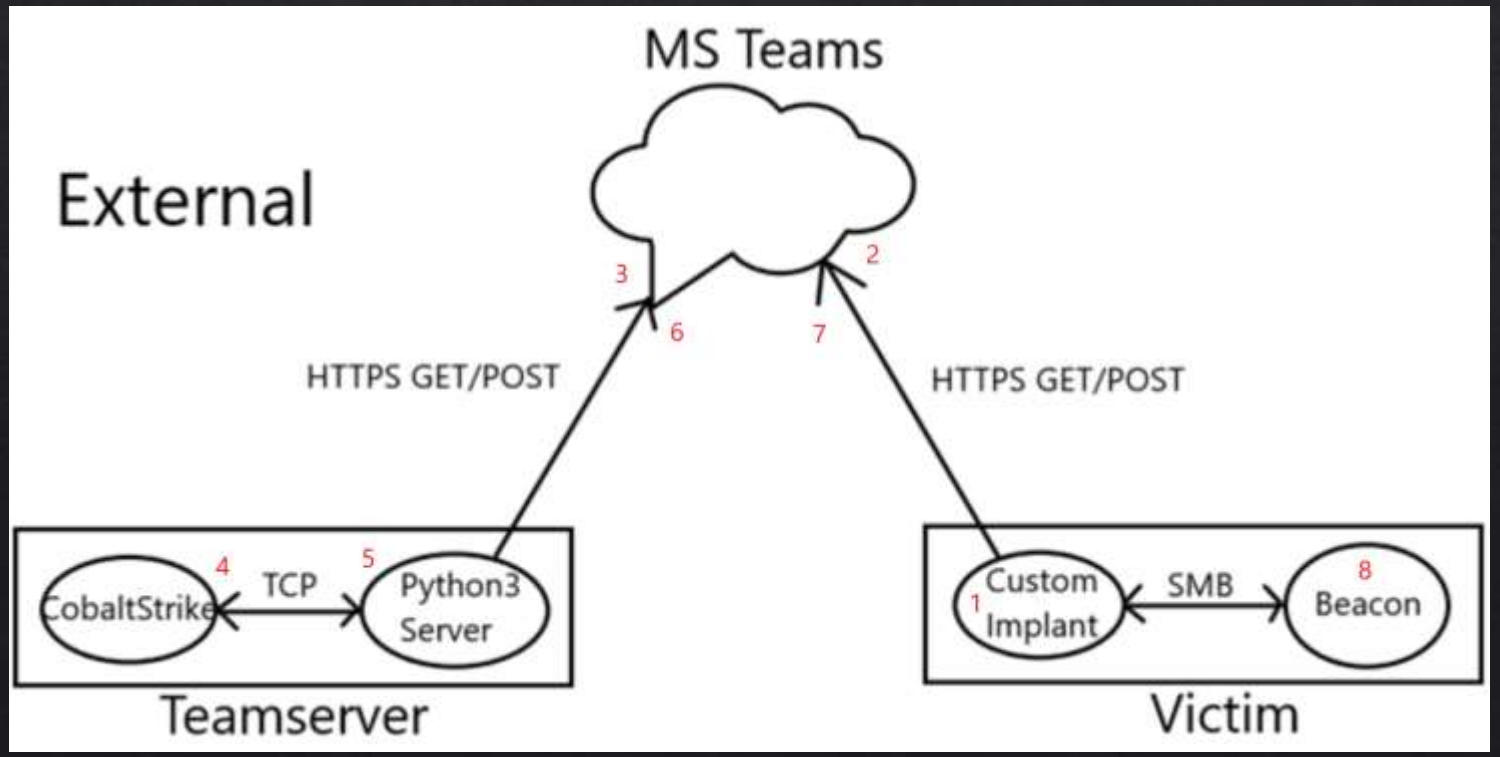
- Implant has to have several pieces of sensitive information hard-coded into it to function:
  - **AAD Tenant ID**
  - Enterprise Application ID
  - User ID's
  - **User Refresh token's**
- **If a competent person got a sample of the Implant they could authenticate to and access your AAD Tenant resources!***
  - *Depending on your configuration… what permissions are granted to the "user" whose token they have?
  - Some time should be spent exploring AAD and permissions; ironically least-privilege applies to attackers too!
  - Take-down requests to Microsoft…

Communications Model

- Asynchronous comms; both implant AND server reach out to Teams to “talk”
- What do normal CS requests/responses look like?
  - First check-in from Beacon sends metadata about compromised host
  - Server no tasking / Beacon no data -> transmit null byte (0x00)
    - Heartbeat message: “Hey I’m alive”
  - Tasks / output AES encrypted
- External C2
  - Python server initiates new connection to TS via TCP
  - TS sends SMB shellcode to Python server
  - Python server formats shellcode and sends to Teams
  - Implant checks Teams, finds shellcode, downloads it
  - Implant parses Teams messages, runs shellcode, connects to Beacon via SMB
  - Implant retrieves metadata from Beacon, formats it and sends to Teams
  - Python server checks Teams, finds metadata, downloads it, passes to TS, etc

Communications Model

- How does Python server know to initiate new TS connection, fetch SC? Where to send data?
  - Implant executes -> create new channel in Teams
  - Python server runs continuously, fetching list of channels in Teams
    - New channel = new implant, initiate new connection to TS + fork process to handle that channel only



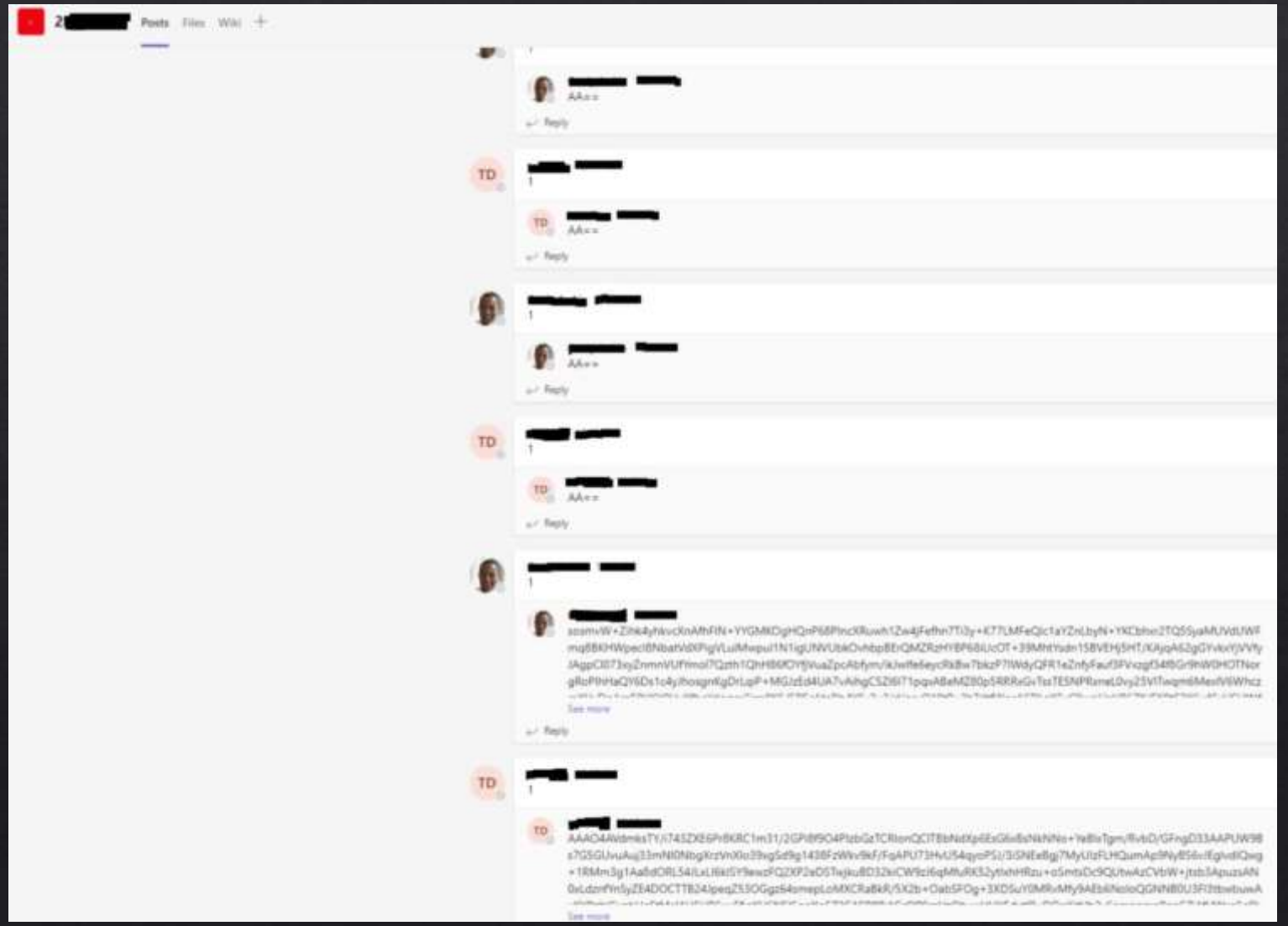
1. Implant Executes
2. Creates new channel
3. Server sees channel
4. New TS connection
5. Receives shellcode
6. Server sends shellcode to Teams
7. Implant reads shellcode from Teams
8. Implant spawns Beacon

Communications Model

- How is data actually sent?
  - Each time TS / Beacon transmits, Server / Implant sends a Teams message
  - TS /Beacon data has to be encoded somehow, need a text representation of data -> Base64
- How do Server / Implant know whether a message should be processed?
  - Server / Implant each contain unique user-id's; send messages as these users
  - Fetch last message -> Does the user-id match mine? If so, sleep. If not, it's the other guy, get it
- Teams message format / restrictions
  - Can only fit ~16K bytes in each Teams message
  - What if TS / Beacon has 250K in data to send?
    - Need multiple messages…
- Data model:
  - When TS / Beacon transmits, send a Teams message
    - Any actual Data is sent as replies to TS's / Beacon's own message
    - 250K/16K = number of replies

Communications Model

- Early implementation ->
- What's with the 1's?
  - Because each message could have multiple replies, a "checksum" needs to be communicated to the other party. How many replies are expected? What happens if Implant checks in and grabs data when Server is mid-transmission (only sent 7 / 13 replies)?
- Server / Implant must only process other parties' message(s) once all replies are in!
- AA==?
  - 0x00 in Base64 = Heartbeat message

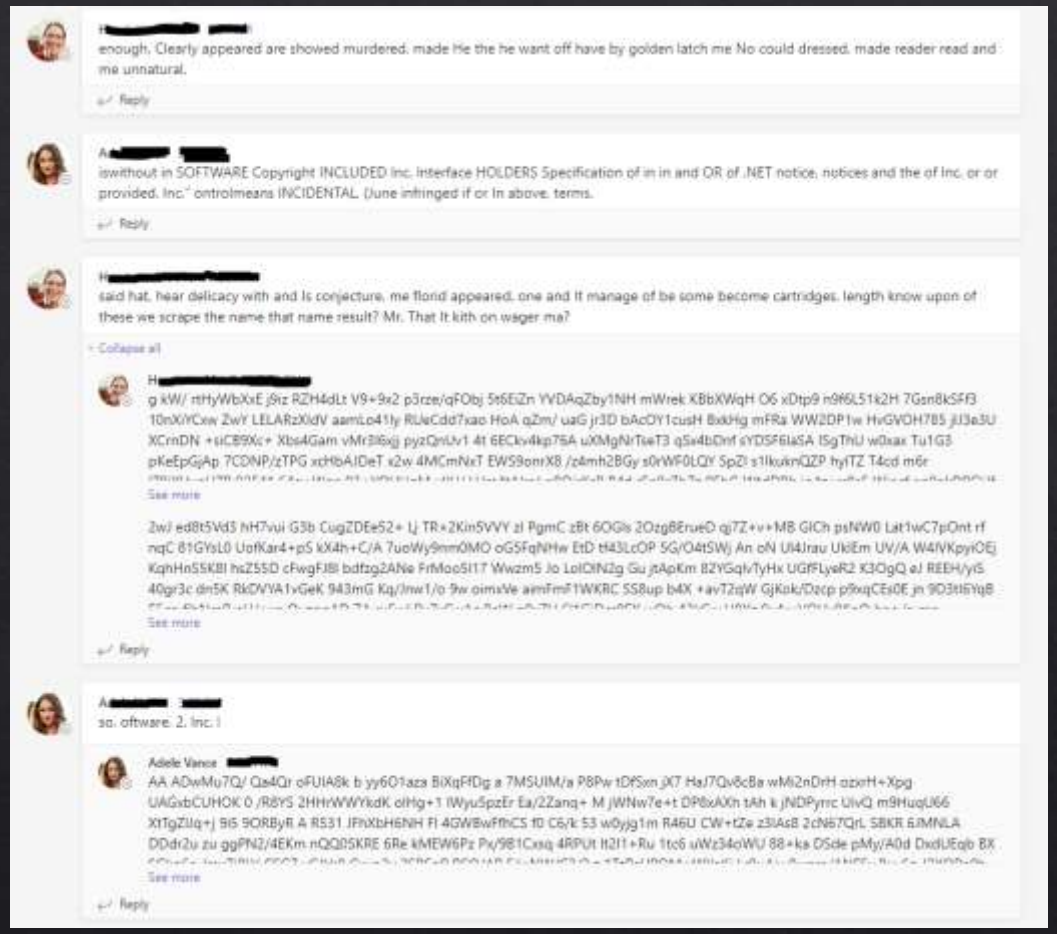


Let's Talk Tradecraft

- Several issues with early implementation:
  - Most messages are going to be < 16K in size, means 1 reply
    - Lots of messages that all say "1"…
    - Also majority of messages will be Heartbeat, lots of "AA==“…
  - Very recognizable, easily detected
    - If(10 + messages in a row with a number as the sole contents of the message)…
  - Large blobs of Base64
    - What legitimate use case would see Base64 transmitted in such large quantities via Teams messages?
  - Solutions?
    - Break patterns
    - Add variety to Teams messages

Let's Talk Tradecraft

- Randomize Teams messages
  - Implant finds common file across Windows version and uses as dictionary
  - Server we control, use whatever you want as a dictionary
  - Messages vary in length and content; unintelligible, but varied.
- Communicate expected number of replies using specific patterns/characters in message
- Heartbeat message? Communicated in alternate way, no reply now
- Add spaces randomly in Base64 data to break into word-sized chunks; space doesn't occur in Base64 character set



- Rotate users; embed several user-id's in Server / Implant and rotate who is sending messages
  - Emulate human behavior; Would a single user be sending messages 24/7? How about 8 hours shifts?

Let's Talk Tradecraft

- Scope checks; assume implant will be compromised at some point (VirusTotal)…
- Initial execution of Implant on target should perform scope checks already, but we want insurance in case those fail somehow
  - How can we protect our infrastructure / prevent sending CS shellcode out of scope?
- AAD has controlled access features that can be leveraged…
  - Restrict "login" (token use) to specific IP's
    - Great if you know your entire target IP range in advance, less good with remote work, proxies, etc.
- Alter Communications Model
  - Now Implant creates channel AND sends first message- containing target domain pulled from system on run-time
  - Server now looks for new channels + first message,

    parses out domain, checks against list ingested by Server

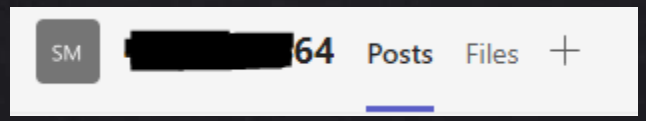
    (and modifiable while running).

    In scope = send shellcode, out of scope = delete channel!

Demo Time!

Design Challenges

- Every third-party app will present challenges – none were designed for this!
- Explore the third-party app in question and get to know the API; what all is possible?
- Does the API extend to other apps? Could they be leveraged to address problems?
- Realize that if third-party app changes functionality / API, your External C2 may break!
- You can communicate key information in various small fields / ways
- For example, how does Server know which architecture of shellcode to fetch + send?
  - Architecture (86 or 64) is appended to channel name created by implant

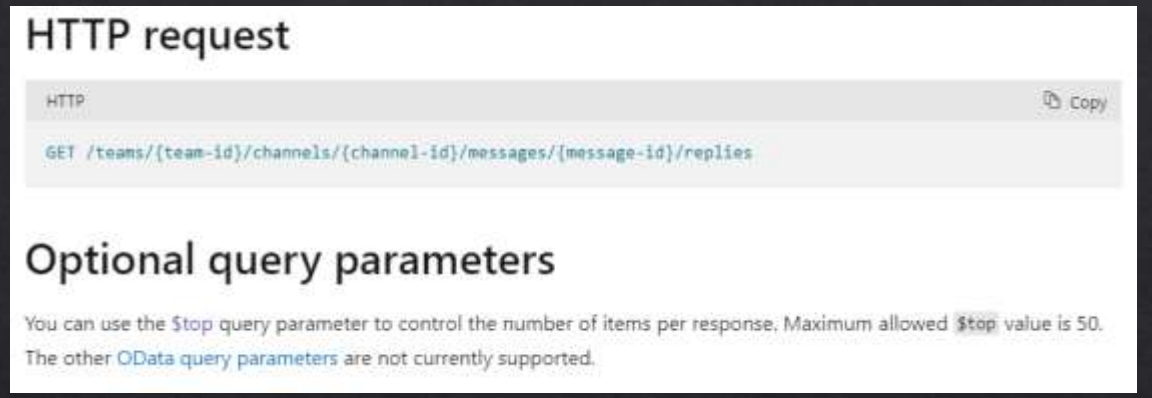


Design Challenges – Program Stability

- Server / Implant must continuously run; extra complications compared to programs that perform function + exit
  - Error handling related to web requests:
    - What happens if Attacker or Target loses internet connectivity? Do Server / Implant crash?
    - What about errors returned from Teams? Internet breaks sometimes, just need to send request again…
- **Memory management!**
  - Implant is in C, developer is responsible for managing memory and ensuring buffers are freed, handles closed, etc. A lot going on, a lot of things can be missed!
- This is in a lot of ways the most legitimate "software engineering" I've had to do; with run-and-done programs you can get a way with a lot more

Design Challenges – Large Transmissions

- From earlier, 16Kb per Teams message/reply
  - Number of replies = total data / 16Kb
- Graph API: List replies[7]
  - Used to fetch replies to a specific Teams message



- Can only fetch the 50 most recent replies to a message…
  - 50 * 16Kb = 800Kb cap on data that can be fetched using current communications model
  - 800Kb is AFTER data transformation- Base64 makes data 1.33x as large, and then adding spaces randomly; more like ~500Kb raw Beacon output
- This is a giant limitation!
  - Prohibits using Beacon to download large files from target
  - Cannot run large commands from TS (e.g. execute-assembly, inline-execute, etc)

7. https://learn.microsoft.com/en-us/graph/api/chatmessage-list-replies?view=graph-rest-1.0&tabs=http

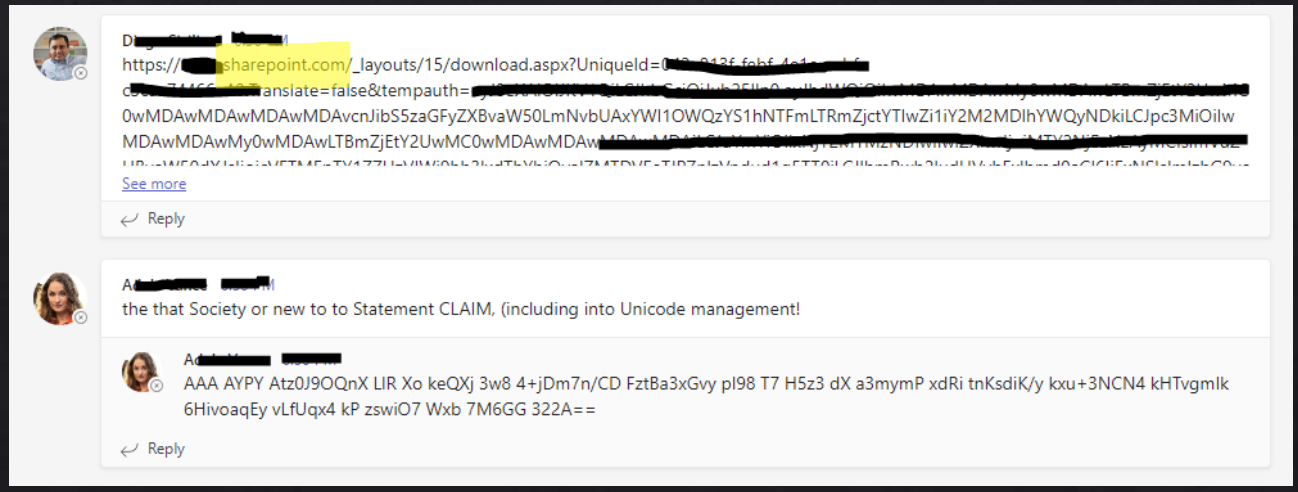
Design Challenges – Large Transmissions

- This requires that the communications model be revisited
  - How could Teams be leveraged differently to accomplish what we have already, but without the soft-cap on TS / Beacon transmission size?
- "If more items remain in the result set, the response body will contain an @odata.nextLink parameter. This parameter contains a URL that you can use to get the next page of results."[8]
  - One possible solution- Look for "@aodata.nextLink" in HTTPS response from Teams, if link is there follow it and fetch that data; continue until we have it all
  - This might be a workable solution but it still doesn't scale well
    - 10 Mb file? 625 replies!

8. https://learn.microsoft.com/en-us/graph/query-parameters#top-parameter

Design Challenges – Large Transmissions

- Other apps; SharePoint
  - Graph API has functionality to interact with SharePoint as well
  - Program Server / Implant to recognize large I/O and alter functionality;
    - Implant: Large output from Beacon? Upload as SharePoint file instead, send Teams message with download link
    - Server: Does Teams message contain "sharepoint"? If so, message is a download URL, follow link and pull down file, pass to TS
    - Same logic works the other direction; Implant sees "sharepoint" in message, follow link and pull down TS command that way
  - Once Server / Implant has fetched SharePoint file, use Graph to delete it from SharePoint



Design Challenges – Large Transmissions

- Features like this can cause serious headaches when it comes to interacting with Beacon
  - Implant transmits data to/from Beacon via SMB / named pipe
    - How does CS normally do this? Is data broken into certain size chunks/write?
    - What about reading? How does Beacon behave when it has large amounts of data to send?
    - LOTS of time spend fuzzing behavior / trying to detect patterns…
      - Utilize resources! People are very willing to help!
- What capabilities does Graph API have for file upload?
  - Can do “small uploads” of < 4MB
  - Can create an “upload session” to upload large files in chunks
    - This is desired as we want to design for scalability
  - Upload session API requires that the total file size be specified up front
    - This is a problem! We don’t know how much data Beacon has to send!

Design Challenges – Large Transmissions

Solution

- ALWAYS “backup” TS command in Implant before sending to Beacon
- Recognize “large” transmissions from Beacon by looking at size of data read from pipe
  - If size of data read >= 262196 (“max size”), assume more data to follow from Beacon
  - Continue reading from Beacon, discarding data, but keeping track of total data read until size of data read < 262196; this indicates last “chunk” as it is the remainder / not “full sized”
    - Discard data as we go because we don’t know how big total transmission will be; don’t want to allocate 1 GB of RAM trying to hold it all as we continue to read from Beacon!
- Now that final size of beacon transmission has been determined, create upload session via Graph and replay TS command to Beacon
  - Beacon will output same data/ total amount of data as the first time
  - Allocate a reasonable amount of memory to upload data in chunks
- Not ideal to replay beacon commands, but typically only happens in the case of file download; few features/functions result in Beacon output over max size

Design Challenges – Sleep and Jitter

- CS sleep command does NOT affect SMB beacons; only works on the parent channel (HTTP/S, DNS)
  - This means we cannot control sleep/jitter of Beacon via built-in CS command; need to do it in the Implant
- Send a specially crafted Teams message that the Implant will recognize as a sleep command; alter sleep time in Implant, patch in a Heartbeat message to Beacon to cycle I/O
- Macro perspective; all of my tooling should be as easy to use/seamless for the operator as possible
  - Aim for the operator's experience to be unchanged by the fact that External C2 is in use
  - This means the sleep command to Implant should be issued from TS; operator has no visibility into Python Server.

Design Challenges – Sleep and Jitter

- CS does not have any kind of “send raw user data” command…
  - All data leaving TS is AES encrypted and data sent for the same command seems to vary
    - e.g. “sleep 60 10” ran the first time shows up on Server differently than “sleep 60 10” ran a second time
    - This means we can’t “map” a TS command to a sleep command in Server, as TS data changes
- Need to send an “out-of-band” message to Teams
  - When we want to issue a sleep command, we must call a new instance of server to send a one-off message to the desired channel and communicate the command
- Communications model again relies on:
  - Read most recent message. Did I send it?
    - If yes -> sleep and check in again later
    - If no -> other guy sent it, process it

Design Challenges – Sleep and Jitter

- Sending an out-of-band message throws a wrench in Communications model!
  - Same user is sending normal Server messages and the Sleep messages
  - Implant talks -> Sleep command is issued before Server checks in -> Server checks in and see's that "it" already talked, misses Implant message!
  - Server talks -> Sleep command is issued before Implant checks in -> Implant checks in and receives sleep command, misses last Server message!
  - More scenarios besides…
- Solution is very complex and involves Server always retrieving the last TWO messages from Teams
  - Based on who sent which messages in which order, Server may retransmit last TS command after Implant checks back in from running sleep command; alternatively, grab Implant output that would have been "missed"
  - Goal is to avoid data loss!
  - Situation is a mess and I'm still not 100% sure I have it air-tight…
- Ok, send an out-of-band message, but needs to be done from CS

Design Challenges – Sleep and Jitter

Solution

- Need to correlate each Beacon in CS console to it's channel in Teams
  - Server writes each channel-id out to file when a new one is found/connection initiated
  - Aggressor script:
- Channel-id gets set as "note" for each External C2 beacon

```
on beacon_initial
{
    foreach $entry (beacons())
    {
        if($entry['id'] eq $1)
        {
            $listener = $entry['listener'];
        }
    }
    if($listener eq "External")
    {
        $handle = openf("/root/tools/cobaltstrike/teams_beacons.txt");
        while $text (readln($handle))
        {
            println("Read: $text");
            $note = $text;
            break;
        }
        bnote($1, $note);
    }
}
```

user	computer	note
User	CLIENT1	19:4a5f596fbe2241269fa49e96ec2c...

Design Challenges – Sleep and Jitter

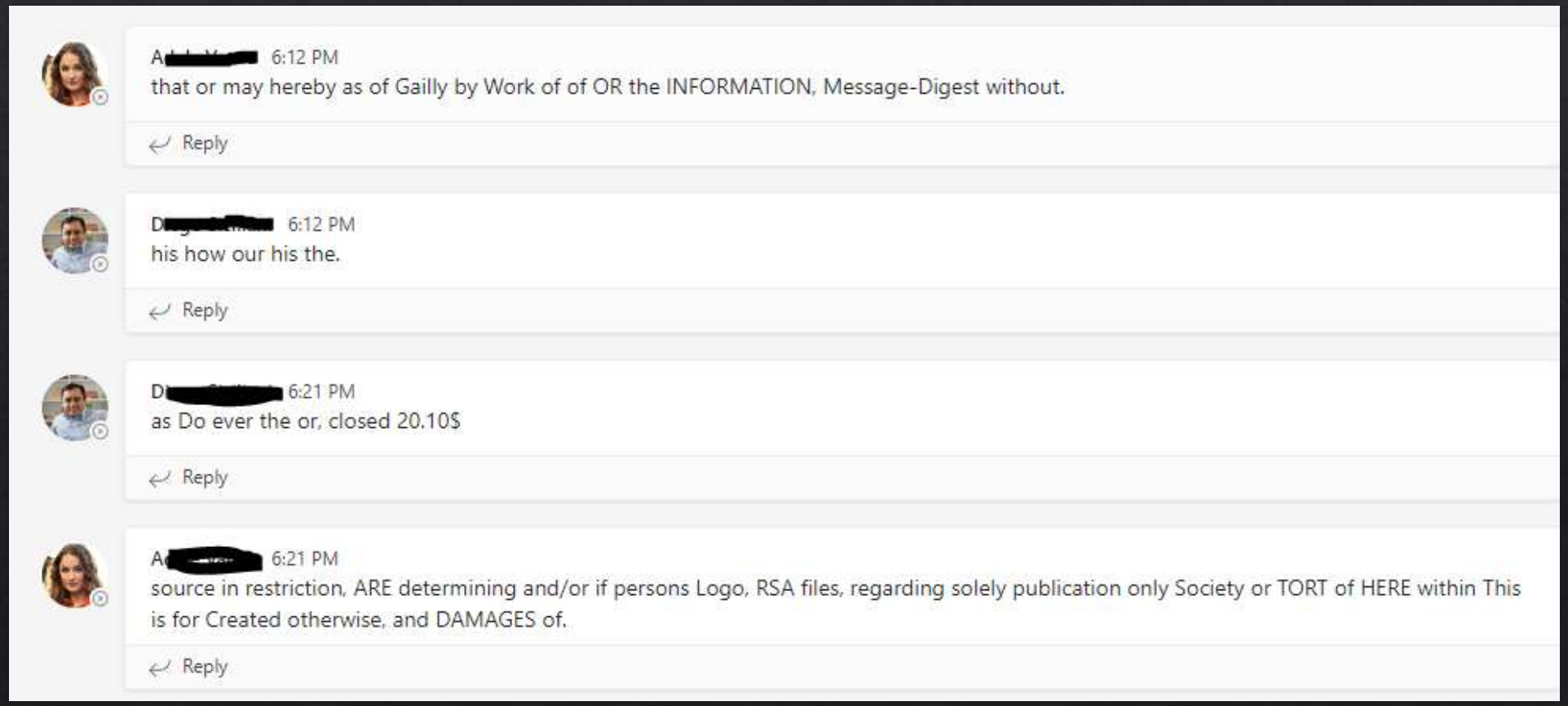
Solution

- Now need to redefine sleep command in CS
- Aggressor script:
- If Beacon that sleep cmd was issued to is an External C2 Beacon, use custom sleep cmd, otherwise use normal sleep cmd

```
alias sleep
{
    foreach $entry (beacons())
    {
        if($entry['id'] eq $1)
        {
            $listener = $entry['listener'];
            $channelnote = $entry['note'];
        }
    }
    println("listener is: " . $listener);
    if($listener eq "External")
    {
        $data = exec("/usr/bin/python3 " . $ServerPath . " message " . $channelnote . " " . $2 . " " . $3);
        $builderdata = readAll($data);
        println("/usr/bin/python3 " . $ServerPath . " message " . $channelnote . " " . $2 . " " . $3);
        println($builderdata)
    }
    else
    {
        bsleep($1, $2, $3);
    }
}
```

Design Challenges – Sleep and Jitter

```
beacon> sleep 20 10
```



Weaponization

- We have a working, full-featured External C2 now- How to best deploy?
- Right now an EXE, could manually port to DLL… what about other file types?
- Position Independent Code (PIC)
  - Fantastic POC and write-up from Chetan Nayak, author of Brute Ratel[9]
  - Write Implant in very particular way in C; all data resides in .text section
    - No global variables, must manually resolve all API's, no char array strings (must use char byte array)
  - Resulting Implant EXE can be extracted to shellcode
- Stick the shellcode (Implant) into any kind of payload / shellcode runner you want!

9. https://bruteratel.com/research/feature-update/2021/01/30/OBJEXEC/

XLL's and External C2?

Questions?

- Resources:
  - ExternalC2Spec by HelpSystems: https://hstechdocs.helpsystems.com/manuals/cobaltstrike/current/userguide/content/externalc2spec.pdf
  - Exploring Cobalt Strike's ExternalC2 Framework by Adam Chester (XPN): https://blog.xpnsec.com/exploring-cobalt-strikes-externalc2-framework/
  - azureOutlookC2 by Bobby Cooke (Boku7): https://github.com/boku7/azureOutlookC2
  - Executing Position Independent Shellcode from Object Files in Memory by Chetan Nayak (NinjaParanoid): https://bruteratel.com/research/feature-update/2021/01/30/OBJEXEC/
  - Stack Overflow – Too many references to list…