Google

Seeing Inside The Encrypted Envelope

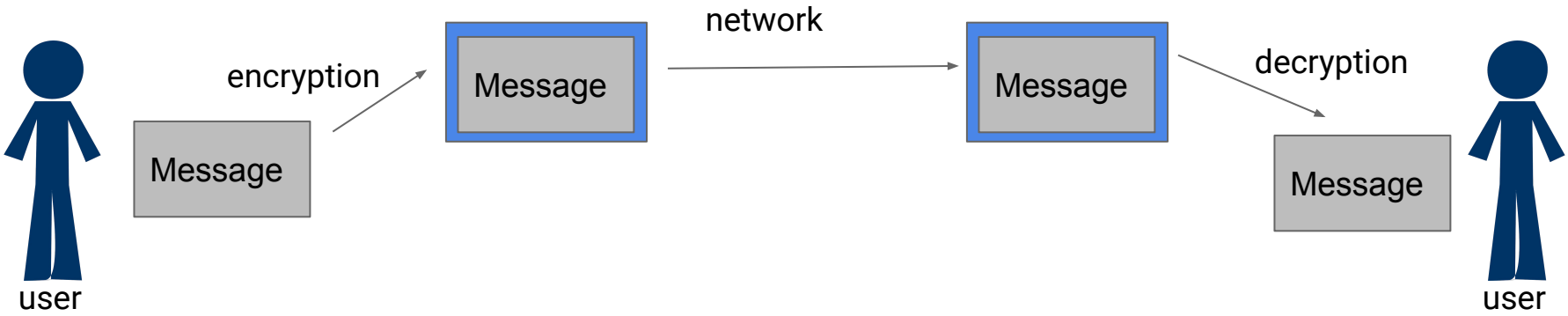
About Me

- Natalie Silvanovich AKA natashenka
- Project Zero member
- Previously did mobile security on Android and BlackBerry
- Messaging enthusiast

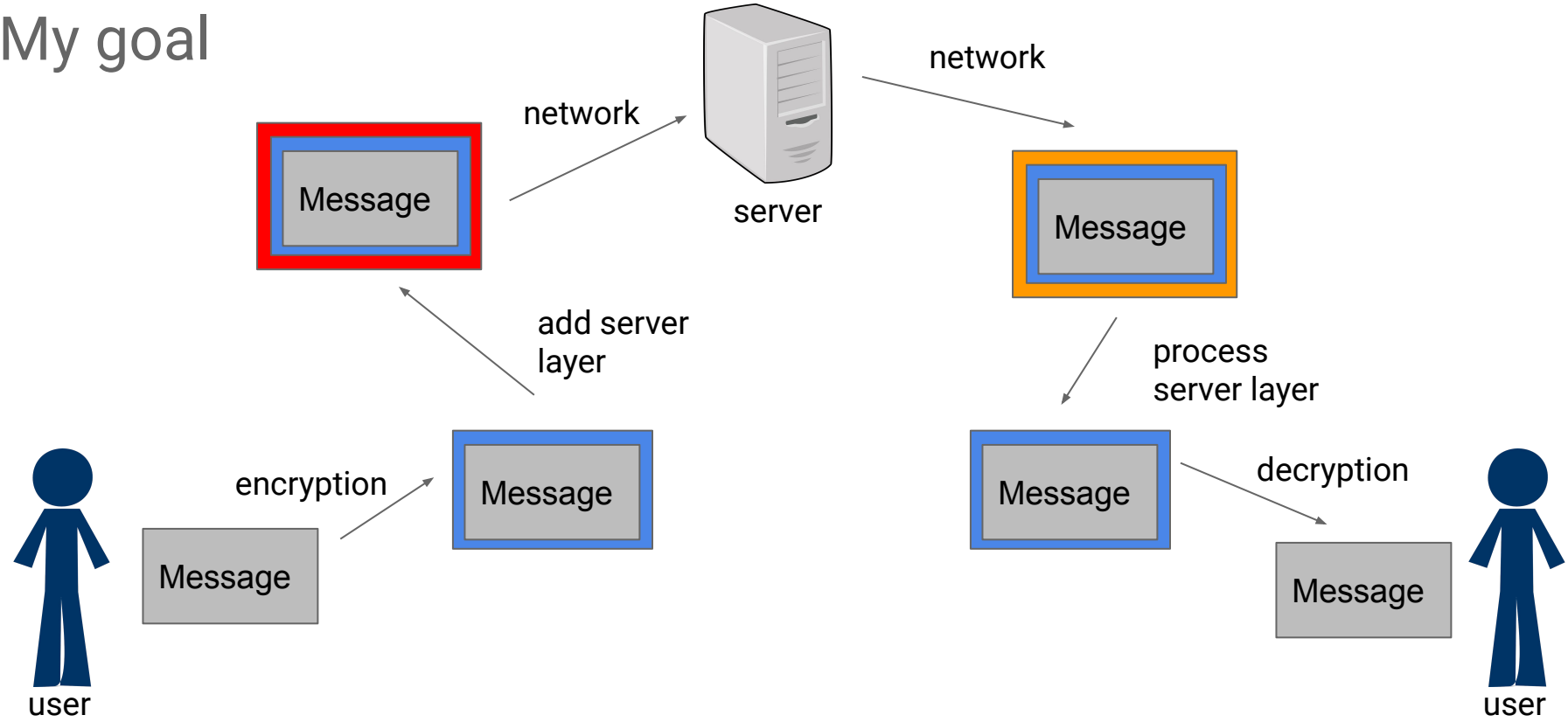
The Problem

- Most remote attack surfaces accept encrypted input
- Attack surfaces that process recently decrypted data are valuable because the server can’t analyze or filter content
- Encryption schemes are usually complicated and/or proprietary

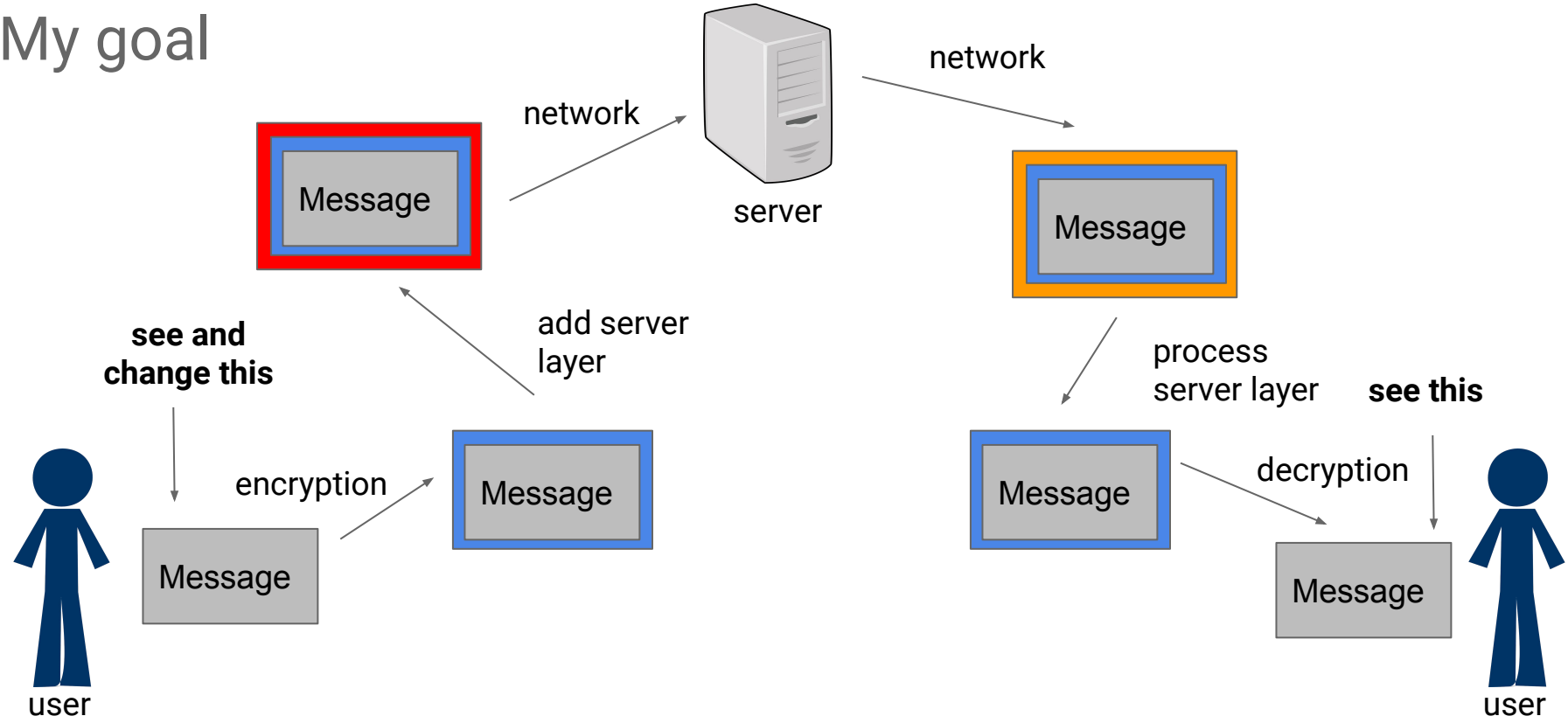
My goal


user
Message
encryption
Message
network
Message
decryption
Message
user

My goal


network
network
Message
server
Message
add server
layer
process
server layer
encryption
Message
Message
Message
decryption
Message
user
user

My goal


network
server
network
Message
Message
Message
see and
change this
add server
layer
process
server layer
see this
encryption
Message
Message
Message
decryption
user
user

Targets

General Considerations

- Platform (mobile versus desktop)
- Open versus closed source
- Custom versus proprietary protocols
- Best effort versus real-time

Strategies

- Intercept over the network and decrypt
- Write or use a client
- Hook the target

Intercepting Traffic

- Generally very difficult strategy unless standard protocols are used
  - Documentation generally lacks details
- Where are you getting the key from?
- Removing crypto function is a possibility
  - Have a ‘special’ libcrypto
  - Make it memcpy or do nothing
  - Can be problematic when reporting bugs

Create a Standalone Client

- This is typically the best and most stable solution
- Heavy development cost
- Easy to distribute
  - Risk of blocking

Using Existing Clients

- This can work if a very good open source client exists
- Most unofficial clients focus on functionality as opposed to coverage
- Changing an open-source client to be suitable for security testing can be time-consuming
- Often use pieces of open source clients for decoding
- Example: Facebook and WhatsApp

Example: WebRTC

- Wrote a standalone client that could start a call with any backend
- Used it to test browsers and Facebook Messenger
- ~ 1 week dev time
- Had difficulty keeping it up to date
- Eventually wrote a command line client that could fuzz on a single device

Hooking

- Hooking functions is a practical low cost solution
  - Often a good way to start to see how buggy software is
- Can be error prone
  - Software updates are a challenge
- Good coverage
- Distribution can be challenging

Hooking

- Two slightly different methodologies
  - Use a debugger-like tool to hook at runtime
  - Modify the binary
- Modification is generally better for performance and stability
- Runtime hooking is generally easier

Examples

- Runtime hooking
  - iMessage
- Application modification
  - Facebook Messenger and WhatsApp signalling (Android application)
  - WhatsApp calling (Android native changes)
  - FaceTime (proprietary all the way down)

iMessage

- Samuel Groß wrote iMessage sending and intercepting client
- Used Frida to hook incoming and outgoing messages

Frida

- Python-based real-time native function hooking framework
  - Can also hook Android Java with limitations
- Works on Android, iPhone, Mac, Linux, etc.
- Just run a binary on the target and attach it to the host via USB
- Actual hooking is written in JavaScript
  - Causes some problems in Objective-C

iMessage Send Script

```
var jw_encode_dictionary_addr =
Module.getExportByName(null,
"JWEncodeDictionary");
send("Hooking JWEncodeDictionary" +
jw_encode_dictionary_addr);
Interceptor.attach(jw_encode_dictionary_addr, {
    onEnter: function(args) {
   var dict = ObjC.Object(args[0]);
```

iMessage Send Script

```
        send(dict.toString())
        var t = dict.objectForKey_("t")
        if (t == "REPLACEME") {
            var newDict =
ObjC.classes.NSMutableDictionary.dictionaryWith
Capacity_(dict.count());
            newDict.setDictionary_(dict);
            newDict.setObject_forKey_("new
message", "t");
```

Android Application Example

- Facebook Messenger
  - Very large, very complicated application
  - It’s usually not necessary to use all of these strategies

Basic Idea

- Find where message is encrypted
- Insert smali code after the message has been serialized, but before it has been signed or encrypted
- Code sends message to remote server, where it can be changed
- Altered message gets sent to test device

Finding the Encryption Point

- Started by decompiling the application APK using apktool
- Get smali files out
- Typically obfuscated
- Android applications contain a lot of unused and rarely used code

```
.method public constructor
<init>(LX/8A2;LX/0Gl;LX/0Gl;LX/89x;LX/1q1;LX/1Xs;LX/0wj;LX/0Gl;
LX/1pr;LX/0wQ;LX/0oS;LX/0dK;LX/0wO;LX/0Gl;LX/1q5;LX/0wm;)V
    .locals 10
    invoke-direct {p0}, Ljava/lang/Object;-><init>()V
    iput-object v9, p0, LX/89y;->c:LX/8A2;
    iput-object v7, p0, LX/89y;->d:LX/0Gl;
    iput-object v6, p0, LX/89y;->e:LX/0Gl;
    iput-object v5, p0, LX/89y;->f:LX/89x;
    iput-object v4, p0, LX/89y;->g:LX/1q1;
    iput-object p4, p0, LX/89y;->h:LX/1Xs;
    iput-object v1, p0, LX/89y;->i:LX/0wj;
    iput-object v0, p0, LX/89y;->j:LX/0Gl;
```

Strategies

- Look for known libraries
  - Libsignal
  - Java crypto
- Focus on natives
- Log entries

Known Libraries

- Most E2E encrypted messengers include libsignal
- Unfortunately, full feature set is not used
- Putting in a stub where libsignal encrypts messages (based on Signal source) did not work on most messengers

Java Crypto Libs

- Cheap trick:
  - Make a build of Android that has a stub in javax.crypto.Mac
  - Make the stub send the digest only when it can access a file in the sandbox of the app you’re testing
  - Will get a lot of stuff that isn’t messages, plus sometimes messages
- Works on about half of messengers

Java Crypto Libs

- Also possible to put log entry that outputs Java stack in Java crypto libs
- Can help you find where the app is encrypting the message
- Relies on the app actually using Java crypto
- Apps often implement their own encryption (wrap a native library), but usually use Java for signing
- Once output stacks in System.arraycopy when I was desperate

Java Crypto Libs

- Can also search smali, but no guarantee stuff gets called
  - Looking for obfuscated functions with byte array parameters worked on WhatsApp
- Can also hook Java crypto with Frida, but doesn’t work well on all devices

Natives (JNI)

- Java Native Interface calls cannot be obfuscated (easily)
- Calls with ‘encrypt’ in the name are good candidates for stub locations
  - Stubs are smali wrappers for the native function
- Messaging encryption is usually native
- Be careful to separate file encryption from network encryption
- Made a script that outputs log entries for every native call

JNI Question

In a Java application, can native code be run without a JNI call?

No.

- JNI can start threads, etc, but native code always starts with a JNI call in an Android Java application

Log Entries

- Some apps have a lot of helpful log entries (and some don't)

```
    const/4 v10, 0x0
    monitor-enter v4
    :try_start_0
    iget-object v0, v4, LX/8B3;->d:Ljavax/crypto/Mac;
    if-nez v0, :cond_10
    sget-object v1, LX/8B3;->a:Ljava/lang/Class;
    const-string v0, "Could not verify Salamander signature -
no SHA256HMAC"
    invoke-static {v1, v0},
LX/00T;->b(Ljava/lang/Class;Ljava/lang/String;)V
    :try_end_0
    .catchall {:try_start_0 .. :try_end_0} :catchall_0
```

Log Entries

- Signature verification failure is a good log entry to look for
- You can add your own log entries

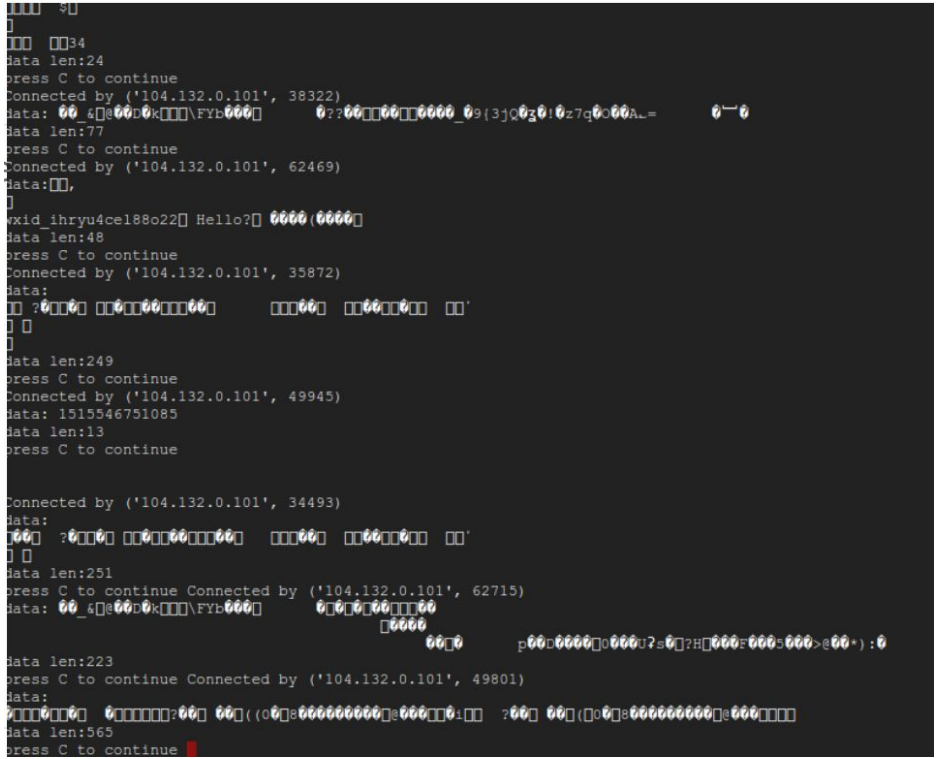
More About Message Encryption

- Apps usually have more than one location where they encrypt messages
  - Messages
  - Attachments
  - Typing/presence indicator
  - Notification content
  - Usually need to add multiple stubs
  - Can add stubs away from encryption too

End Result

- Facebook
  - Added smali stubs in several locations, including wrapping native encryption in smali
- WhatsApp
  - Added smali stub at a single location, far from natives
  - Also altered serialization code at various locations to alter certain message fields without understanding the format (for example, testing directory traversal by changing path generation)

Messages!


data len:24
press C to continue
Connected by ('104.132.0.101', 38322)
data len:77
press C to continue
Connected by ('104.132.0.101', 62469)
data len:48
press C to continue
Connected by ('104.132.0.101', 35872)
data len:249
press C to continue
Connected by ('104.132.0.101', 49945)
data: 1515546751085
data len:13
press C to continue
Connected by ('104.132.0.101', 34493)
data len:251
press C to continue Connected by ('104.132.0.101', 62715)
data len:223
press C to continue Connected by ('104.132.0.101', 49801)
data len:565
press C to continue

Android Native Example

- WhatsApp calling required intercepting messages in the native code

WhatsApp Calling

- Looked at Android App
- No symbols, but log entries from libsrtp and PJSIP
- Identified memcpy from packet to buffer before encryption (looked for srtp_protect log entries)

WhatsApp Calling

- Wrote a Frida script that hooked all memcpy instances
- Frida is awesome!

```
hook_code =""

                    Interceptor.attach (Module.findExportByName (
                "libc.so", "read"), {

                        onEnter: function (args) {

                        send (Memory.readUtf8String (args [1]));

                     },
```

WhatsApp Calling

- Frida is too slow to make a call without a lot of lag
  - Good for debugging binary changes though
- Changed specific memcpy to point to function I wrote in ARM64
- Assembly of my function overwrote GIF transcoder

WhatsApp Calling

- Original branch to malloc was BL instruction
- Used the ARM branch finder to make it point to my function instead http://armconverter.com/branchfinder/
- My function calls dlopen, dlsym and then a function in libnatalie.so

WhatsApp Calling

- Had issues with calls disconnecting, turned out I was corrupting a used register
- After a few fixes could log and alter incoming packets
- Replaying packets by pure copying did not work

RTP Protocol

Bit Offset	0-1	2	3	4-7	8	9-15	16-31
0	Version	Padding	Ext.	CSRC Count	Marker	Payload Type	Sequence Number
32	Timestamp						
64	Synchronization Source (SSRC) Identifier						
96	Contributing Source (CSRC) Identifier						
96+32*CC	Payload						

Interesting Parts of RTP Headers

- SSRC is a random identifier that identifies a stream
  - WhatsApp cannot be limited to a single stream
- Payload type is an identifier that identifies content type, and is consistent

WhatsApp Calling

- WhatsApp has FOUR RTC streams, even when muted
- Luckily, they have different payload types
- Fixing ssrc and sending logged packets worked

FaceTime

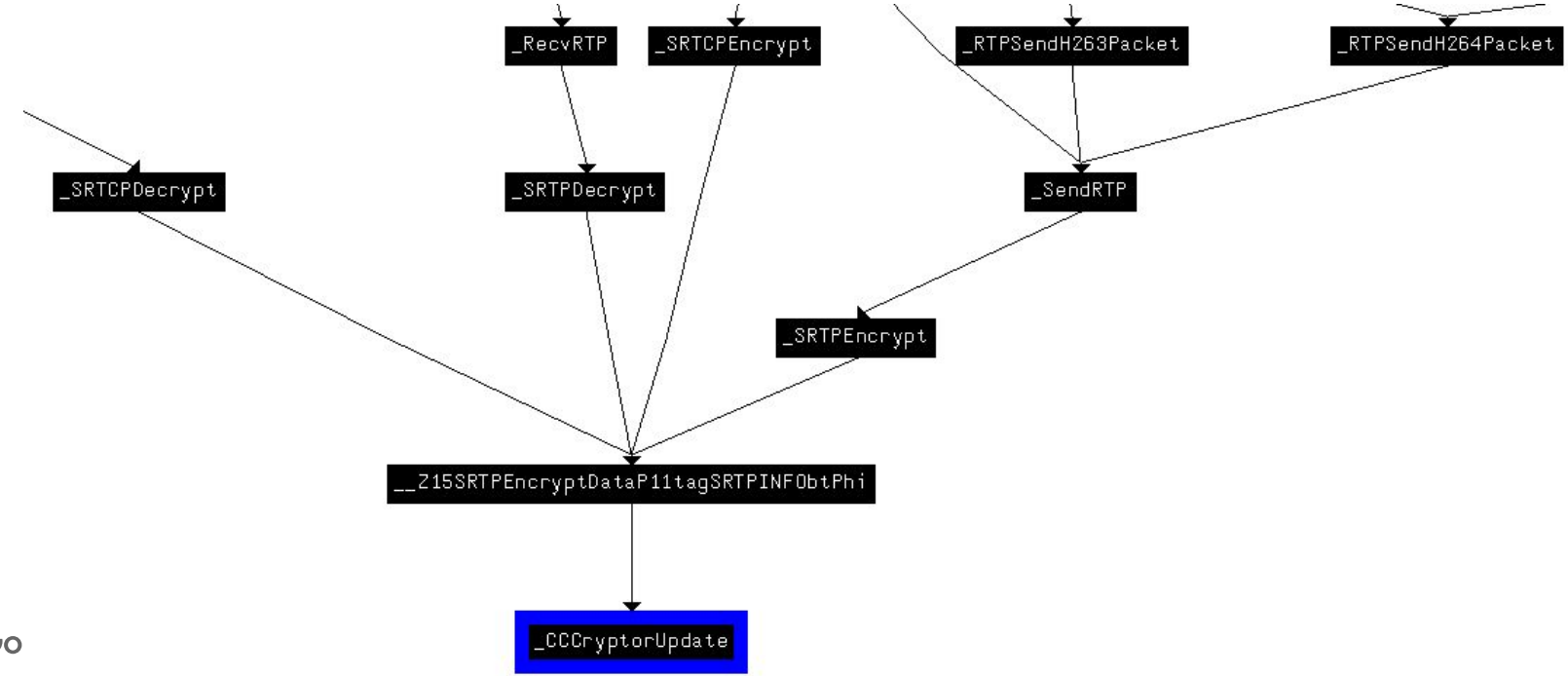
- Limited open-source components
- Runs on Mac
- Needed to modify binary to log packets

FaceTime

- FaceTime is closed-source and proprietary
- Needed to modify binary to log packets

FaceTime Encryption

- Used IDA to identify call to encryption function



Hooking Functions on MacOS

- CCCryptorUpdate seemed a good candidate for recording RTP
- DYLD_INTERPOSE can be used to redirect library calls on Macs
- Requires setting an environment variable
  - This isn’t possible for AVConference, which is started as a daemon

Hooking Functions on MacOS

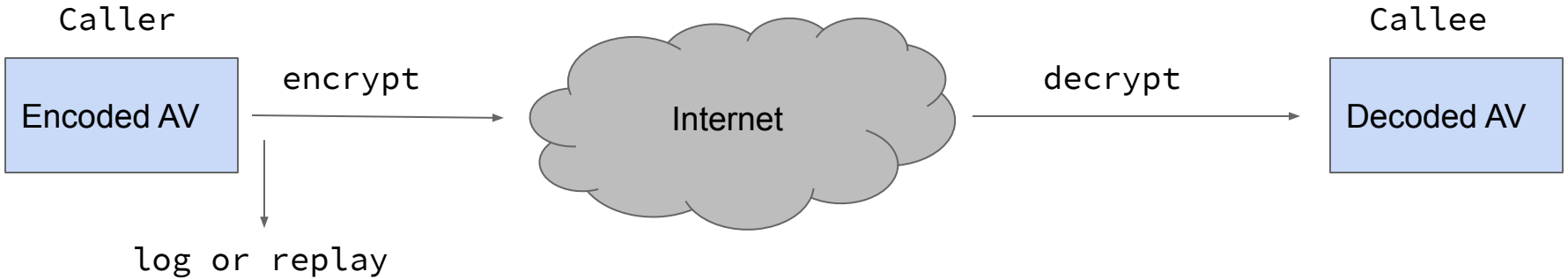
- DYLD_INTERPOSE can also be called in the static section of a library loaded by a Mac binary
- Found insert_dylib on github https://github.com/Tyilo/insert_dylib
- Inserted static library that hooked CCCryptorUpdate

```
DYLD_INTERPOSE(mycryptor, CCCryptorUpdate);

CCCryptorStatus mycryptor(
   CCCryptorRef cryptorRef, const void
*dataIn,
   size_t dataInLength, void *dataOut,

   size_t dataOutAvailable,size_t
*dataOutMoved) {
```

Hooking Functions on MacOS

- Tried making a call
- Needed some refinement
  - Limited hooking to functions that sent RTP
  - Added a spinlock
  - Patched binary to pass length
- Could alter RTP in real time, but replay did not work!

Hooking Functions on MacOS



Investigating RTP Packets

- Read through _SendRTP function to figure out packet generation
- Discovered RTP headers were created well after encryption

Bit Offset	0-1	2	3	4-7	8	9-15	16-31
0	Version	Padding	Ext.	CSRC Count	Marker	Payload Type	Sequence Number
32	Timestamp						
64	Synchronization Source (SSRC) Identifier						
96	Contributing Source (CSRC) Identifier						
96+32*CC	Payload						

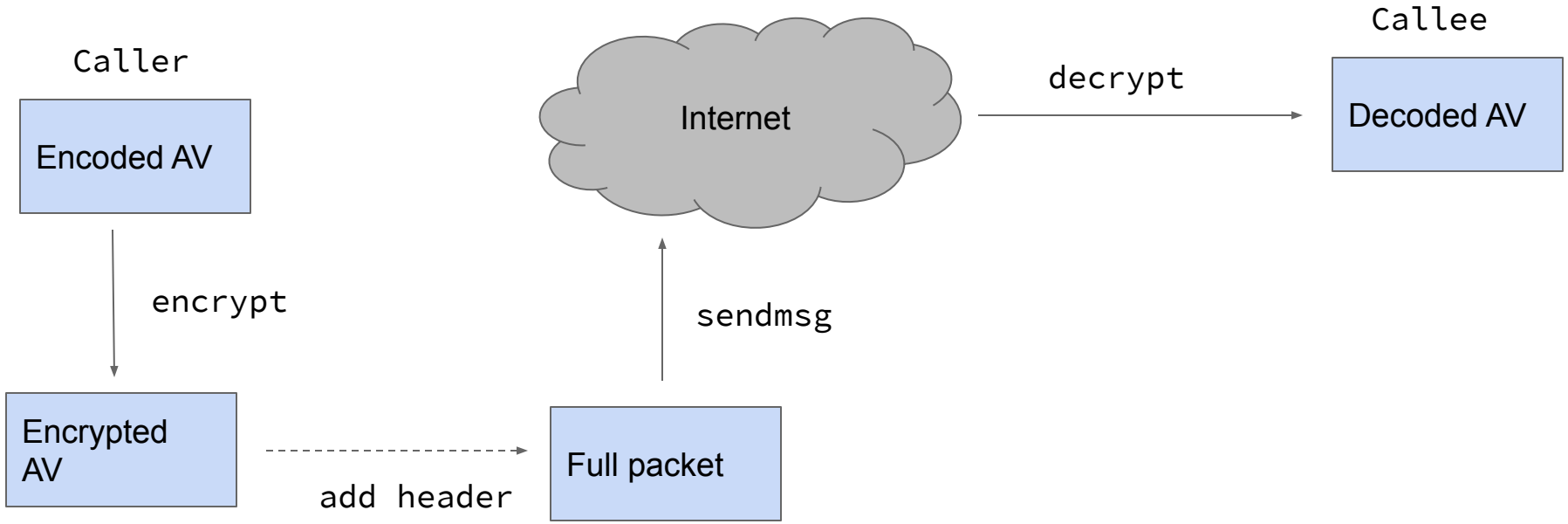
Interesting Parts of RTP Headers

- SSRC is a random identifier that identifies a stream
  - FaceTime cannot be limited to a single stream
- Payload type is a constant that identifies content type
- Extensions are extra information that is independent of the stream data
  - Screen orientation
  - Mute
  - Quality
  - Wait a sec, these totally depend on stream data

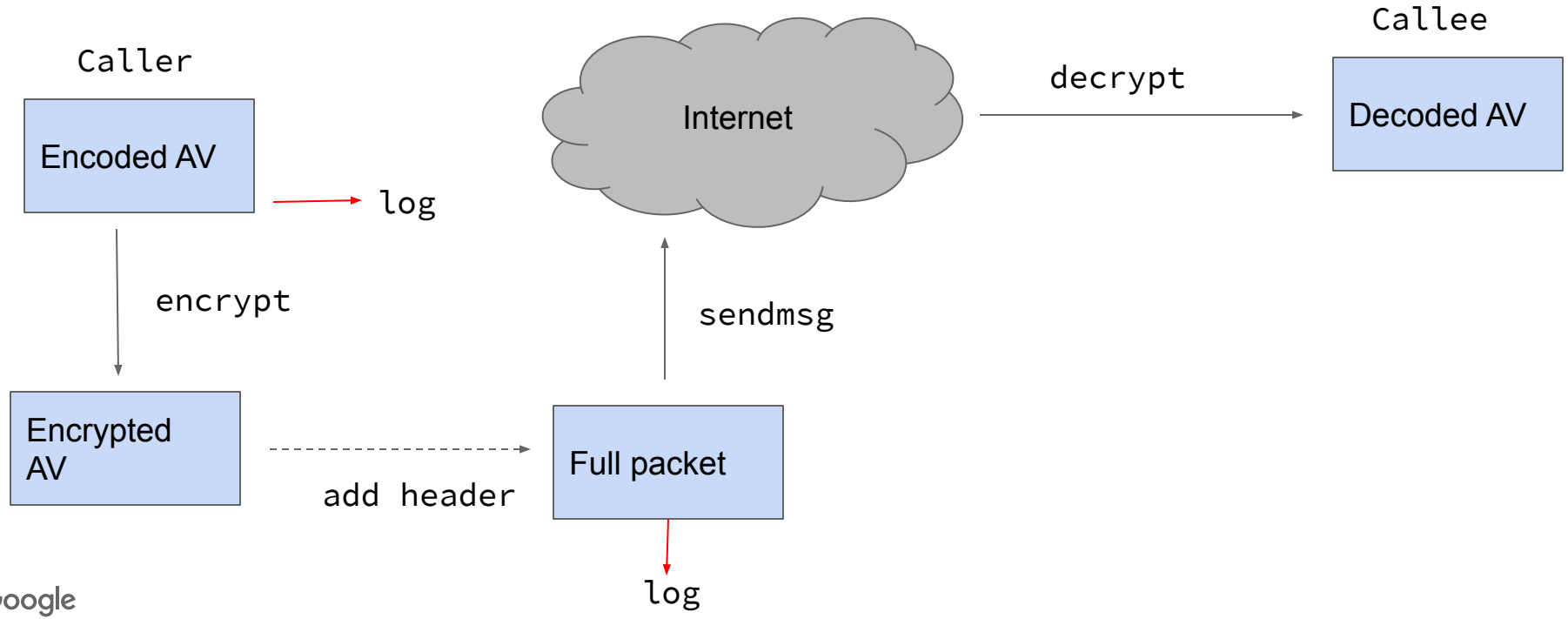
Hooking Headers?

- Tried replaying with existing headers
- Hooked sendmsg to capture and log header
  - Needed to tie encrypted message to header
  - sendmsg NOT called on packets in the same order as encryption (even with a spinlock)
  - Need to ‘fix’ SSRC and sequence number

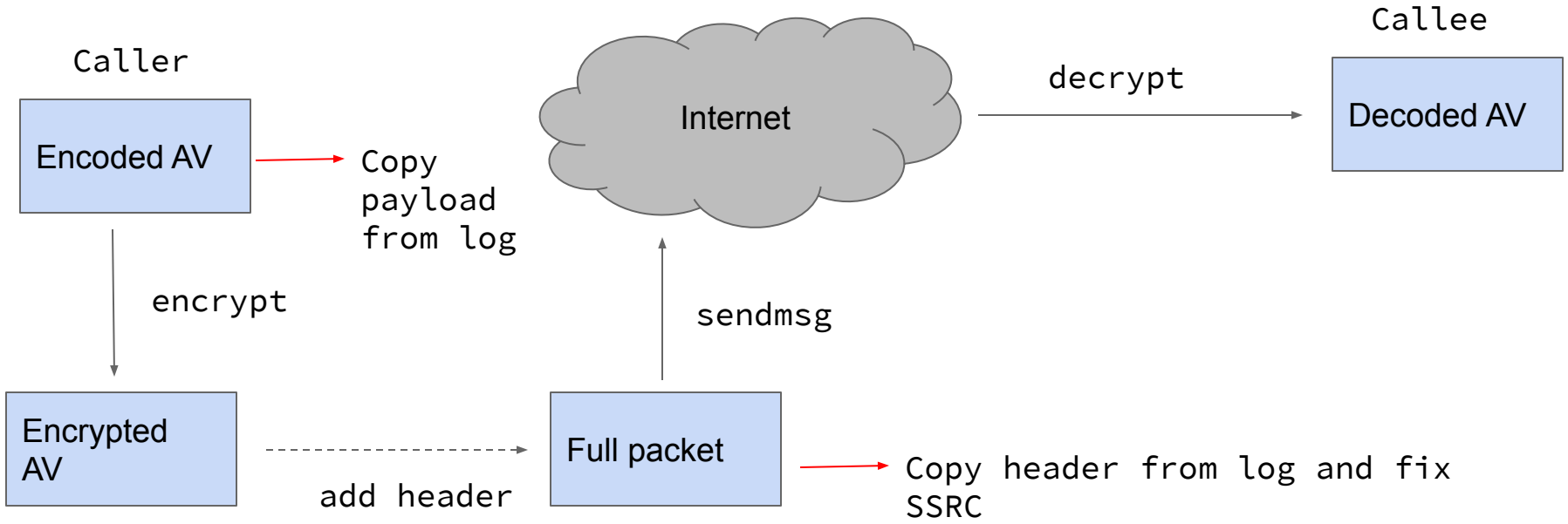
Fixing headers



Fixing headers (send)



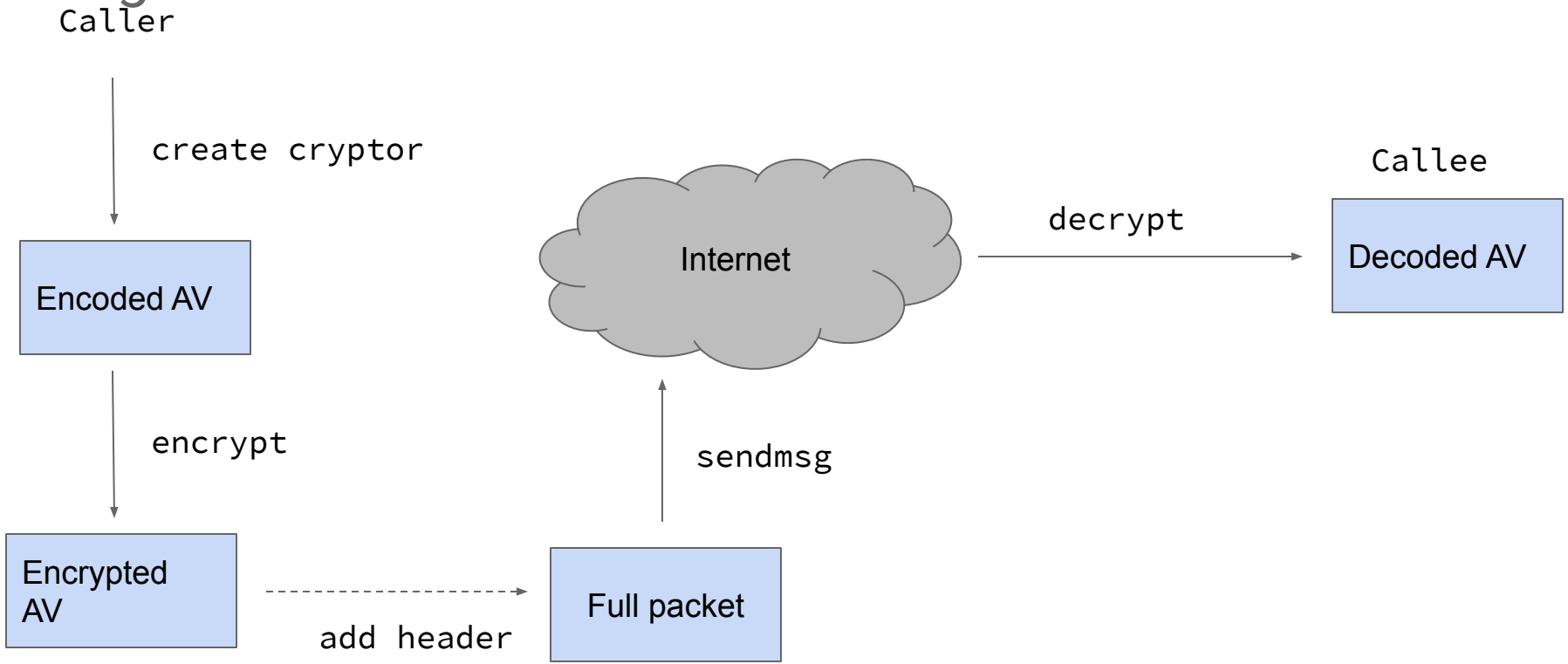
Fixing headers (replay)



Still Didn’t Work

- Patched endpoint to remove encryption
  - This worked, but can’t do it on an iPhone
  - Audio data clearly getting corrupted in decryption
- Created a cryptor queue for each SSRC, and encrypted the data in order
- Discovered encryption is XTS with sequence number as counter
- Fixed seq number counter

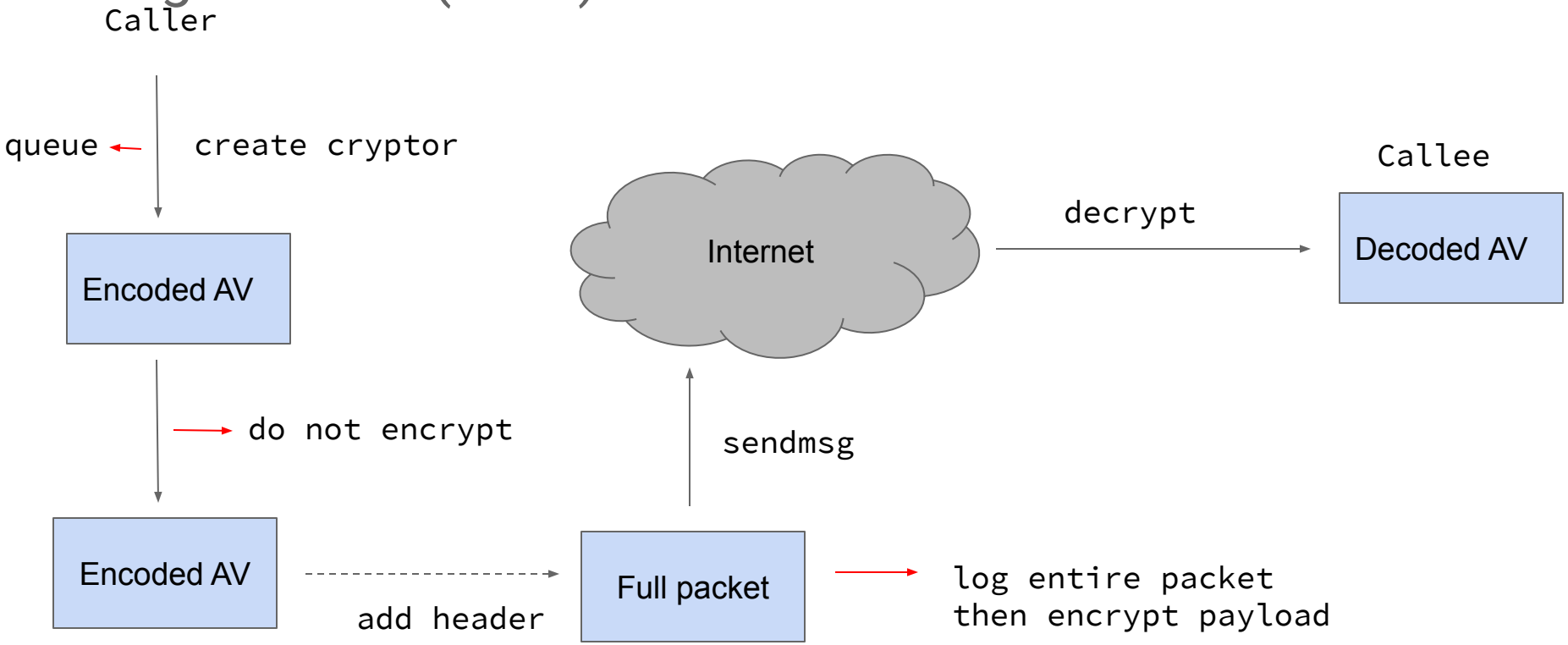
Fixing headers



Steps to Log

- Hook CCCryptorCreate to log cryptors as they are created
  - Store cryptors by thread in queues
- Hook CCCryptorUpdate, and prevent packets from being encrypted
- Hook sendmsg, log unencrypted packet, and then encrypt it using the cryptor from the queue

Fixing headers (send)



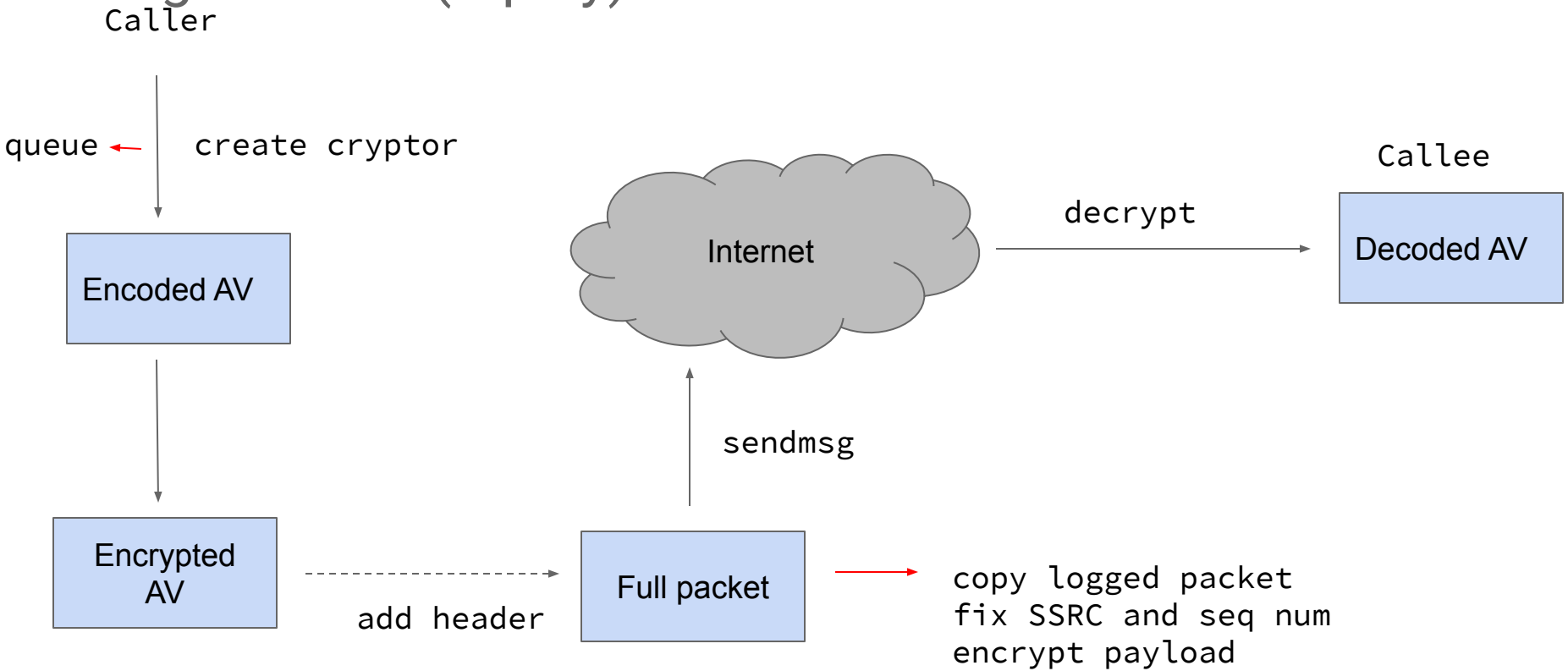
Steps to Replay

- Hook CCCryptorCreate to log cryptors as they are created
  - Store cryptors by thread in queues
- Hook sendmsg, save current ssrc and sequence number if it hasn’t been seen before
- Copy logged packet into current packet

Steps to Replay

- Replace logged ssrc with ssrc for payload type
- Replace logged sequence number with logged sequence number - starting logged sequence number + starting sequence number for ssrc
- Pop a cryptor for the payload type and encrypt the payload
  - If there are no cryptors left, don’t send and wait

Fixing headers (replay)



Demo

Conclusions

- Hooking is generally the best strategy, balancing time investment and functionality
- Stand alone clients and network interceptions are also options
- Tools like Frida can make hooking easy in some circumstances
- Otherwise binary modification is necessary

Conclusions

- Found many bugs with these techniques

https://bugs.chromium.org/p/project-zero/issues/list?can=2&q=label%3AFinder-natashenka

Conclusions

Questions

https://googleprojectzero.blogspot.com/

@natashenka
natashenka@google.com