void fuzz(char* buf, int& len){

    int q = rand()%20;

    if (q == 7){
        int ind = rand()%len;
        buf[ind] = rand();
    }

    if(q == 5){
        for(int i = 0; i < len; i++)
            buf[i] = rand();
    }

    if(q == 11){
        int l = rand()% MAX_PACKET_LEN;
        *len = l;
    }
}
}
Adventures in Video Conferencing
About Me

- Natalie Silvanovich AKA natashenka
- Project Zero member
- Previously did mobile security on Android and BlackBerry
- Defensive-turned-offensive researcher
Video Conferencing

- Video conferencing has expanded greatly in the past 5 years
  - Browsers
  - FaceTime
  - WhatsApp
  - Facebook
  - Signal
WebRTC
What is WebRTC?

- RTC = Real Time Communication
- Audio and video conferencing library maintained by Chrome
- Used by
  - Browsers (Chrome, Firefox, Safari)
  - Messaging applications (Whatsapp, Facebook Messenger, Signal, SnapChat, Slack, etc.)
- Little security information available
Known Security Bugs

List of Known Security Bugs

- (None so far)
WebRTC Architecture

Signalling Server

Signalling

JS API

Browser

Alice

Media

Browser

Bob
WebRTC Architecture

- **XHR**
- **SSE**
- **WebSocket**
- **HTTP 1.x/2**
- **Session (TLS)**
- **Transport (TCP)**
- **Network (IP)**

- **RTCPeerConnection**
- **DataChannel**
- **SRTP**
- **SCTP**
- **Session (DTLS)**
- **ICE, STUN, TURN**
- **Transport (UDP)**
Packet Decoding Sequence

- SRTP
  - decrypt
- RTP
- RTP
  - error correction
- VP8/VP9/H264
  - payload format decoding
- Audio or video
  - codec decoding
Idea 1: Session Description Protocol

- SDP is the most sensitive interface of WebRTC
  - WebRTC requires parsing untrusted SDP with no user interaction
- Used WebRTC library to create SDP fuzzer on commandline
- Reviewed SDP code
- No bugs!
- Some platforms implement separately
Idea 2: RTP and Media Protocols

- WebRTC has already implemented fuzzers for RTP, media protocols and codecs
  - But what about end-to-end?
- Wrote end-to-end fuzzer for RTP
Evolution of a fuzzer

**Prototype**

- Altered Chrome to add fuzzer
- Had one browser instance ‘call’ another
- **Crashed roughly every 30 seconds**
- Learned that the concept would generally work
- Got very shallow bugs that blocked fuzzing fixed
Evolution of a fuzzer

**Client Fuzzer**

- Wrote C++ client that interacts with browser
  - Lighter weight than browser
  - Can run against any target
  - Pro: crashes are guaranteed to work on browser
  - Con: slow
- Found additional end-to-end vulnerabilities in WebRTC
Evolution of a fuzzer

**Distributed Fuzzer**

- Wrote command line RTP emulator with help of WebRTC team
  - Pro: extremely fast, runs on multiple cores
  - Pro: supports coverage
  - Con: not an exact representation of any WebRTC implementation
- Many bugs!
Results

- 7 vulnerabilities found and fixed
  - CVE-2018-6130 -- out-of-bounds memory issue related to in VP9
  - CVE-2018-6129 -- out-of-bounds read in VP9
  - CVE-2018-6157 -- type confusion in H264
  - CVE-2018-6156 -- overflow in FEC
  - CVE-2018-6155 -- use-after-free in VP8
  - CVE-2018-16071 -- a use-after-free in VP9
  - CVE-2018-16083 -- out-of-bounds read in FEC
std::map<int64_t, GofInfo> gof_info_ RTC_GUARDED_BY(crit_);
gof_info_.emplace(unwrapped_tl0,
    GofInfo(&scalability_structures_[current_ss_idx_],
    frame->id.picture_id));
if (frame->frame_type() == kVideoFrameKey) {
    GofInfo info =
        gof_info_.find(codec_header.tl0_pic_idx)->second;
    FrameReceivedVp9(frame->id.picture_id, &info);
    UnwrapPictureIds(frame);
    return kHandOff;
}
std::map<int64_t, GofInfo> gof_info_ RTC_GUARDED_BY(crit_);
gof_info_.emplace(unwrapped_tl0,
    GofInfo(&scalability_structures_[current_ss_idx_],
    frame->id.picture_id));
if (frame->frame_type() == kVideoFrameKey) {
    GofInfo info =
        gof_info_.find(codec_header.tl0_pic_idx)->second;
    FrameReceivedVp9(frame->id.picture_id, &info);
    UnwrapPictureIds(frame);
    return kHandOff;
}
const_iterator std::map::find ( const key_type & __x ) const [inline]

Tries to locate an element in a map.

Parameters:

x    Key of (key, value) pair to be located.

Returns:

Read-only (constant) iterator pointing to sought-after element, or end() if not found.
WebRTC Security Problems

- WebRTC has billions of users
- WebRTC provided no way to report security bugs
- WebRTC documentation provided no guidance on updates
FaceTime
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

Caller

Encoded AV

encrypt

Internet

decrypt

Decoded AV

log or replay

Callee
**Investigating RTP Packets**

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

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

- **Caller**
  - Encoded AV
  - encrypt
  - Encrypted AV
- **Internet**
  - Full packet
  - add header
- **Callee**
  - Decoded AV
  - decrypt
Fixing headers (send)

Caller

Encoded AV

Encrypt

Encoded AV

Encrypt

Encrypted AV

add header

Full packet

log

Internet

decrypt

Decoded AV

decrypt

Callee
Fixing headers (replay)

Caller

Encoded AV

Encrypted AV

encrypt

Internet

decrypt

Callee

Decoded AV

Copy payload from log

encrypt

sendmsg

Full packet

Copy header from log and fix SSRC

Copy header from log and fix
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

Caller

create cryptor

Encoded AV

encrypt

Encrypted AV

add header

Full packet

sendmsg

Internet

decrypt

Callee

Decoded AV
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)

Caller

Encoded AV

decrypt

Encoded AV

Internet

sendmsg

Full packet

add header

Decoded AV

Callee

log entire packet then encrypt payload
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 ssr with ssr for payload type
- Replace logged sequence number with logged sequence number - starting logged sequence number + starting sequence number for ssr
- 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)

Caller

queue ← create cryptor

Encoded AV

create cryptor

Internet

decrypt

Callee

Decoded AV

sendmsg

Full packet

copy logged packet
fix SSRC and seq num
encrypt payload

Encrypted AV

add header

Encoded AV

Encrypt AV
Demo
Results

- CVE-2018-4366 -- out-of-bounds read in video processing on Mac
- CVE-2018-4367 -- stack corruption
- CVE-2018-4384 -- kernel heap corruption in video processing
  - CVE-2015-7006 (found by Adam Donenfeld of Zimperium) is similar and exploitable
- CVE-2019-6224 -- overflow in splitting RED packets
WhatsApp
WhatsApp

- Looked at Android App
  - Desktop app does not do voice
- No symbols, but log entries from libsrtp and PJSIP
  - PJSIP is a commercial library similar to WebRTC
- Identified memcpy from packet to buffer before encryption (looked for srtp_protect log entries)
● 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]));
    },
    onLeave: function (retval) {
        },
WhatsApp

- 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
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
WhatsApp

- WhatsApp has FOUR RTP streams, even when muted
- Luckily, they have different payload types
- Fixing ssrsc and sending logged packets worked
Crash Detection

- WhatsApp handles signal crashes internally
  - Creates crash reports in unknown format
  - FB Messenger and other apps also do this
- WhatsApp crashes do not get logged by logcat
- Stubbed out signal() and sigset() in library to get around this
- Crashes were logged by Android after this
Result

- CVE-2018-6344 -- Heap Corruption in RTP Processing
WhatsApp Signalling

- While reversing RTP processing, it became clear signalling messages were processed by native code
- Processing was not limited to correct packets for the state
- Reviewed each entry point
- Found boring crashes, but nothing interesting
  - Service respawns
WhatsApp Signalling

- Discovered signalling processes a large JSON blob “voip_params” from the server
- Sets dozens of parameters internally
- Discovered a peer could send this blob in one packet type
- Reviewed the code
- Fuzzed the parser with help from Tavis Ormandy
- No bugs ...
WhatsApp Signalling

- WhatsApp was aware of these attack surfaces
- Was aware of other voip_params issues
  - Fixed the one I reported quickly
  - Considering signing
- Has plans to reduce the attack surface of signalling
Conclusions
****, I Was Supposed To Have Learned Something From Fuzzing RTP, Wasn’t I?

Scott Ippolito
11/03/15 9:51am • SEE MORE: OPINION

When I was diagnosed with cancer 7 months ago, I was horrified. But every 3 years I visited the hospital that death was just around the corner, that it might not be there for my family anymore, that I might never see my kids get married and have kids of their own. But after countless operations and the latest round of my chemo, I was given a clean bill of health. And now I’m back to my normal, everyday routine and... oh, uh, well, this...

I was supposed to learn some profound realization from something cancer, wasn’t I?
Bug Summary

- WebRTC: 7 bugs
- FaceTime: 5 bugs
- WhatsApp: 1 bug
Bug Location

- RTP: 0
- Error correction: 3
- Payload format: 7
- Codec: 2
Timing

- WebRTC: 4 weeks
- FaceTime: 6 weeks
- WhatsApp RTP: 2 days
- WhatsApp signalling: 3 weeks
Conclusions

• Video conferencing contained many vulnerabilities
  ○ Complexity is a cause, but probably necessary
  ○ Patching is a concern

• Video conferencing lacks test tools
  ○ Tooling was time consuming but worth it
  ○ https://github.com/googleprojectzero/Street-Party

• Signaling is a possible area for more bugs

• RTP needs more fuzzing
Questions

https://googleprojectzero.blogspot.com/
@natashenka
natashenka@google.com