# T-BONE: Drone vs. Tesla

Ralf-Philipp Weinmann (Kunnamon) <ralf@kunnamon.io>
Benedikt Schmotzle (Comsecuris) <benedikt@comsecuris.com>

KUNNAMON

Comsecuris

# Who are we?

- **Comsecuris**: working on automotive projects since 2014
  - common theme: target device emulation is immensely helpful for assessments

- **Kunnamon, Inc.**: founded in 2019 to modernize embedded automotive development
  - side effect: also useful for security research

KUNNAMON

Comsecuris

# PWN2OWN VANCOUVER 2019: TESLA, VMWARE, MICROSOFT, AND MORE

January 14, 2019 I Brian Gorenc

That's right. We'll have a **Tesla** Model 3 on-site as a target for our automotive category, which has six different focal points for in-scope research (details below). Tesla essentially pioneered the concept of the connected car with their Model S sedan, and in partnership with Tesla, we hope to encourage even more security research into connected vehicles as the category continues to expand. Prizes range from $35,000 to $300,000 depending on a variety of factors including the exploit used. And the first successful researcher can also drive off in their own brand new Model 3 after the competition ends. See the rules section below for specific target categories and awards.

*Automotive Category: Tesla Model 3*

An attempt in this category must be launched against a Tesla Model 3 mid-range rear wheel drive vehicle. The available targets and awards are as follows:

| Target | Escape Option | Prize | Master of Pwn Points | Eligible for Persistence Add-on | Eligible for CAN Bus Add-on |
|---|---|---|---|---|---|
| Modem or Tuner | N/A | $100,000 | 10 | No | Yes |
| Wi-Fi or Bluetooth | N/A | $60,000 | 6 | No | Yes |
| Infotainment | N/A | $35,000 | 3 | No | No |
| | Sandbox Escape | $85,000 | 8 | Yes | Yes |
| | Root/Kernel EoP | $85,000 | 8 | Yes | Yes |
| Gateway, Autopilot, or VCSEC | N/A | $250,000 | 25 | Yes | No |
| Autopilot Denial of Service | N/A | $50,000 | 5 | No | No |
| Key Fobs or Phone-as-Key | N/A | $100,000 | 10 | No | No |

# Public Tesla research up until Pwn2Own 2019

- Tencent KEEN Security lab: *Free-Fall* (2016)
  - Chain from browser exploit (WebKit engine back then) to CAN bus control.
- Tencent KEEN Security lab: *Over-the-Air: How we remotely compromised the gateway, BCM, and Autopilot ECUs of Tesla cars* (2017/18)
  - Browser exploit (Webkit) again, bypassed AppArmor using known kernel bug, broke code signing to deploy own firmware and escalate to CAN bus control
- Tencent KEEN Security lab*: Experimental Research of Tesla Autopilot* (2019 shortly after contest)

KUNNAMON

Comsecuris

# Team Kunnapwn 2019

- Ralf Philipp Weinmann
- Benedikt Schmotzle
- Kevin Redon

# Contest Strategy:

- Try to pop VCSec as main target.
- Backup: Work on Infotainment

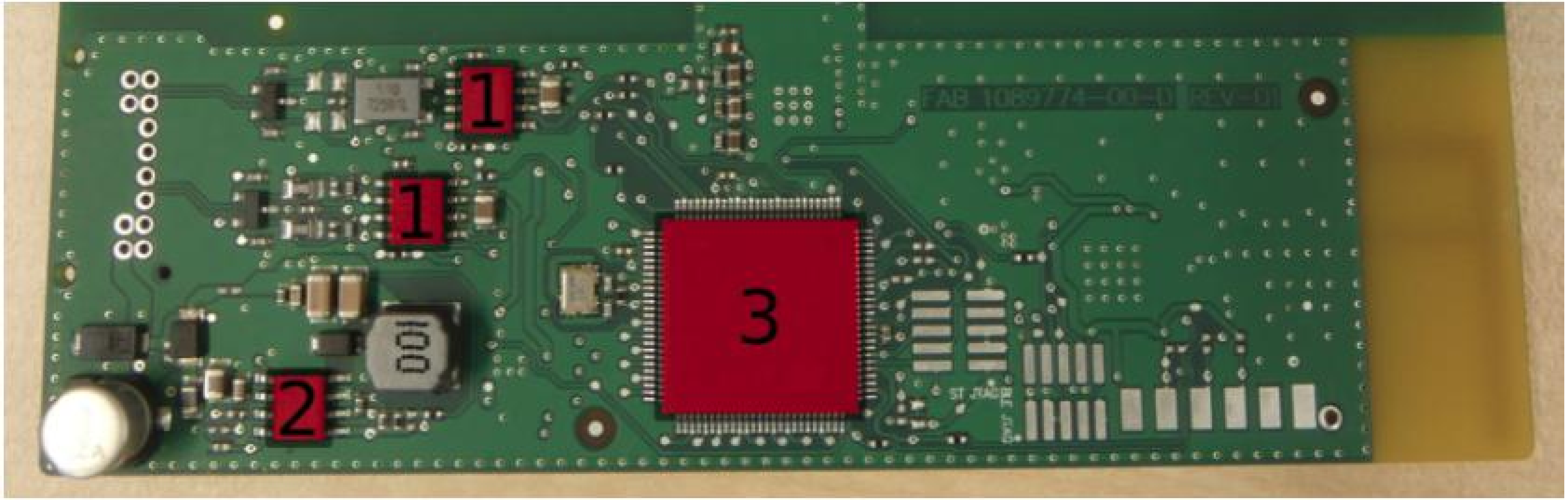
KUNNAMON

Comsecuris

# VCSec overview

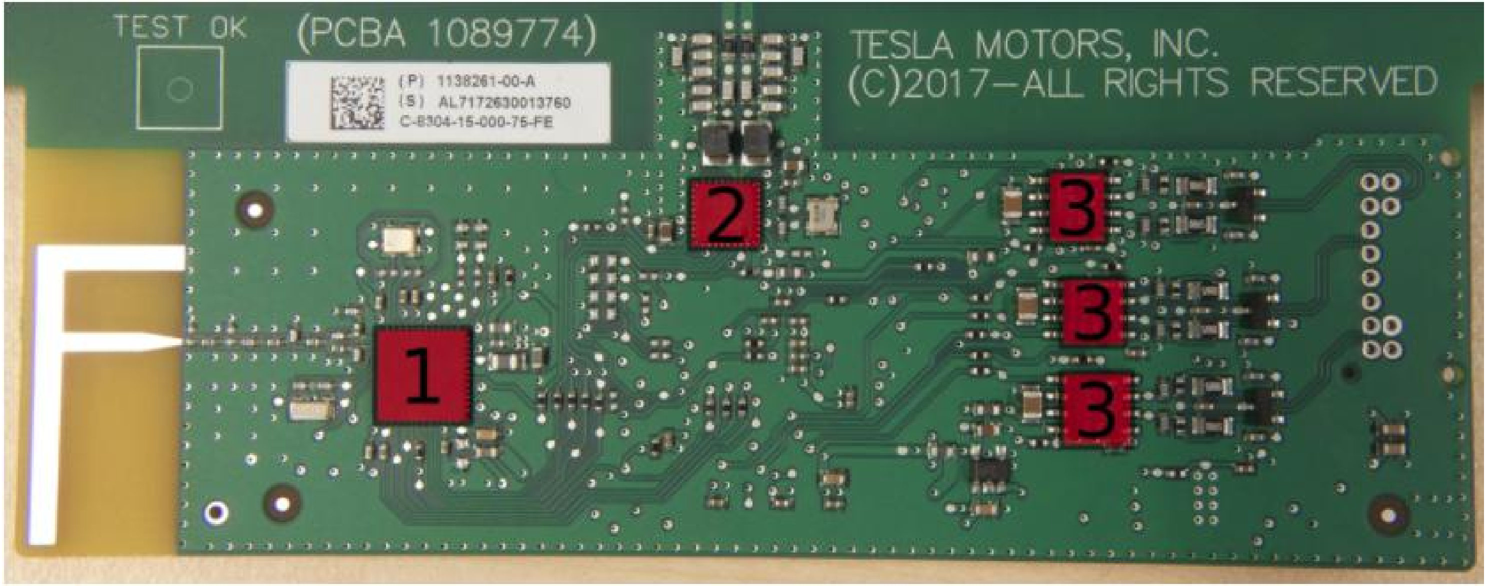
- Tesla security module
- Handles car key cards open/close/start requests using NFC
- Talks to Keyfobs using Bluetooth LE
- Connected directly to the CAN bus
- Reachable from Infotainment only through Gateway

# VCSec (Top) internals




1. CAN transceiver (Texas Instruments TCAN1042V)
2. Voltage regulator (Texas Instruments) 1402SQ
3. MCU (ST SPC560B64L3) -> Firmware running on 32-bit PowerPC (in VLE mode)

# VCSEC (Bottom) internals




1. BLE MCU (TI CC2640R2F) -> Firmware running on Cortex-M3 (32-bit ARM)
2. NFC reader (ST ST25R3915)
3. CAN transceiver (TI TCAN1042V)

# VCSEC internals

- In 2019: No mitigations against memory corruptions: neither on the RF side (CC2640R2F; Bluetooth stack) nor on the CAN side (PowerPC SoC)
    - CC2640 affected by https://www.armis.com/bleedingbit/ RCEs in 2018
- CC2640R2F talks to SPC560B64L3 using a custom protocol
- SPC560B64L3 then issues CAN bus commands

# Approach

- Use JTAG to dump firmware of both CC2640R2F and SPC560B64L3
- Perform static analysis of both firmwares
- Fuzzed emulated CC2640R2F firmware to find vulnerabilities
- Exploit possible bug in CC2640R2F, then find vulnerabilities in SPC firmware to escalate and ultimately issue arbitrary CAN frames
- Ralf developed health issues before contest (slipped disc), attempt failed

# In parallel: Infotainment research (ICE)

- Normal Linux (4.14 kernel at the time) running on an Atom (x86_64)
- Kernel source and buildroot environment publicly available at https://github.com/teslamotors/linux/tree/intel-4.14 and https://github.com/teslamotors/buildroot respectively
- Tesla tries to use least-privilege approach for userland
  - Unique users for different services
  - Kafel for syscall filtering
  - AppArmor to restrict further
- Identified vectors for ICE: Browser (Webkit was changed to Chrome days before the Contest), WiFi, Bluetooth (probably others, such as Spotify; it also parses data from the internet)
- Uses ConnMan for connection management

# ConnMan

- ConnMan (https://git.kernel.org/pub/scm/network/connman/connman.git/)
  - Network connection manager
  - Lots of protocols supported: DHCP, DNS, IPv4, IPv6, NTP, WPAD, …
  - Written in plain C
- Initially written by someone at Intel
- Reachable from WiFi => looks like an ideal target.
- Built harness to fuzz DNS reply parsing with AFL
- Minutes later: first crash due to 90s style memcpy() stack smash
- Discarded bug at first due to stack cookies. While the service restarts, cookie is re-randomized at each restart.

# Result of our PWN2OWN 2019 attempt

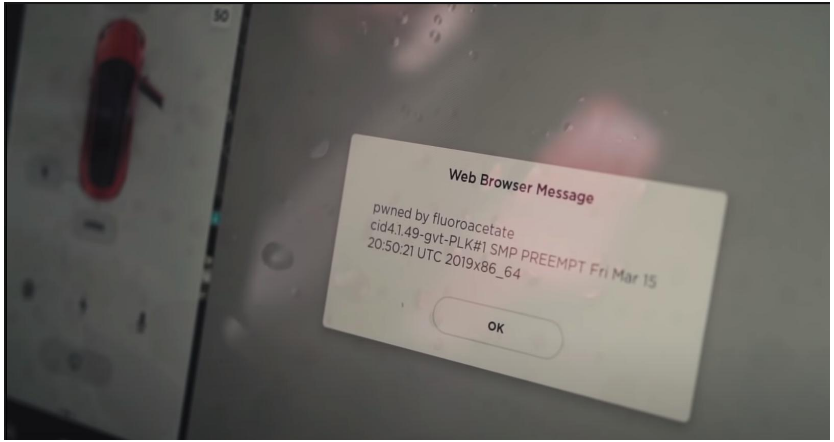
- Made salvaged Infotainment work
- Made salvaged VCSEC work
- Patched Ghidra to correctly handle PowerPC VLE
- Emulated CC2640R2F for fuzzing
- Didn’t find bugs for escalation in time, dubious whether BLE findings really would have been exploitable
- Found memory corruption bugs in ConnMan

# Pwn2Own 2019 Tesla Result

- Fluoroacetate (Amat Cama, Richard Zhu) popped Tesla using Chrome RCE bug on the Infotainment.

1300 - Fluoroacetate (Amat Cama and Richard Zhu) targeting the infotainment system (Chromium) on the Tesla Model 3 in the automotive category.

**Success:** - The Fluoroacetate duo used a JIT bug in the renderer to win $35,000 and a Model 3.



# Fast forward to 2020...

## PWN2OWN RETURNS TO VANCOUVER FOR 2020

January 09, 2020 I Brian Gorenc

| Target: Initial Vector | Target: Intermediate Pivot | Target: Final Stage | Prize Amount | Master of Pwn Points | Additional Prize Options |
|---|---|---|---|---|---|
| Tuner, Wi-Fi, Bluetooth, or Modem | Infotainment | VCSEC, Gateway, or Autopilot | $500,000 | 50 | Infotainment Root Persistence Add-on<br>Autopilot Root Persistence Add-on<br>CAN Bus Add-on |

| Target: Initial Vector | Target: Final Stage | Prize Amount | Master of Pwn Points | Additional Prize Options |
|---|---|---|---|---|
| Tuner, Wi-Fi, Bluetooth, or Modem | Infotainment | $250,000 | 25 | Infotainment Root Persistence Add-on<br>CAN Bus Add-on |
| Infotainment | VCSEC, Gateway, or Autopilot | $300,000 | 30 | Infotainment Root Persistence Add-on<br>Autopilot Root Persistence Add-on |
| Tuner, Wi-Fi, Bluetooth, or Modem | VCSEC, Gateway, or Autopilot | $400,000 | 40 | Infotainment Root Persistence Add-on<br>Autopilot Root Persistence Add-on |

# Team Kunnapwn 2020

- Ralf Philipp Weinmann

- Benedikt Schmotzle

# In the habit of reassessing found bugs from time to time

- During OffensiveCon 2020: “Lets look at these ConnMan crashes again”
  - After some staring at the code, it became obvious that we can indeed use the bug to jump over the stack cookie and perform a partial write on x64
  - This defeats both stack cookies and ASLR
  - However, exploit dev time still estimated to be high without info leak
- Spent some time to find an info leak => found another OOB bug in the ConnMan DHCP stack that leads to a suitable stack information leak

# CVE-2021-26675

```c
static char *uncompress(int16_t field_count, char *start, char *end,
                        char *ptr, char *uncompressed, int uncomp_len,
                        char **uncompressed_ptr)
{
        char *uptr = *uncompressed_ptr; /* position in result buffer */

        debug("count %d ptr %p end %p uptr %p", field_count, ptr, end, uptr);

        while (field_count-- > 0 && ptr < end) {
                int dlen;               /* data field length */
                int ulen;               /* uncompress length */
                int pos;                /* position in compressed string */
                char name[NS_MAXLABEL]; /* tmp label */
                uint16_t dns_type, dns_class;
                int comp_pos;

                if (!convert_label(start, end, ptr, name, NS_MAXLABEL,
                                        &pos, &comp_pos))
                        goto out;

                /*
                 * Copy the uncompressed resource record, type, class and \0 to
                 * tmp buffer.
                 */

                ulen = strlen(name);
                strncpy(uptr, name, uncomp_len - (uptr - uncompressed));

                debug("pos %d ulen %d left %d name %s", pos, ulen,
                        (int)(uncomp_len - (uptr - uncompressed)), uptr);

                uptr += ulen;
                *uptr++ = '\0';

                ptr += pos;
```

```c
                /*
                 * We copy also the fixed portion of the result (type, class,
                 * ttl, address length and the address)
                 */
                memcpy(uptr, ptr, NS_RRFIXEDSZ);

                dns_type = uptr[0] << 8 | uptr[1];
                dns_class = uptr[2] << 8 | uptr[3];

                if (dns_class != ns_c_in)
                        goto out;

                ptr += NS_RRFIXEDSZ;
                uptr += NS_RRFIXEDSZ;
```

[1] [https://www.forescout.com/company/resources/namewreck-breaking-and-fixing-dns-implementations/](https://www.forescout.com/company/resources/namewreck-breaking-and-fixing-dns-implementations/)

# CVE-2021-26676

```
static gboolean listener_event(GIOChannel *channel, GIOCondition condition,
                                                        gpointer user_data)
{
        GDHCPClient *dhcp_client = user_data;
        struct sockaddr_in dst_addr = { 0 };
        struct dhcp_packet packet;
        struct dhcpv6_packet *packet6 = NULL;
        uint8_t *message_type = NULL, *client_id = NULL, *option,
                *server_id = NULL;
        uint16_t option_len = 0, status = 0;
        uint32_t xid = 0;
        gpointer pkt;
        unsigned char buf[MAX_DHCPV6_PKT_SIZE];
        uint16_t pkt_len = 0;
        int count;
        int re;

        if (condition & (G_IO_NVAL | G_IO_ERR | G_IO_HUP)) {
                dhcp_client->listener_watch = 0;
                return FALSE;
        }

        if (dhcp_client->listen_mode == L_NONE)
                return FALSE;

        pkt = &packet;

        dhcp_client->status_code = 0;

        if (dhcp_client->listen_mode == L2) {
                re = dhcp_recv_l2_packet(&packet,
                                        dhcp_client->listener_sockfd,
                                        &dst_addr);
```

```
...
        switch (dhcp_client->state) {
        case INIT_SELECTING:
                if (*message_type != DHCPOFFER)
                        return TRUE;

                remove_timeouts(dhcp_client);
                dhcp_client->timeout = 0;
                dhcp_client->retry_times = 0;

                option = dhcp_get_option(&packet, DHCP_SERVER_ID);
                dhcp_client->server_ip = get_be32(option);
                dhcp_client->requested_ip = ntohl(packet.yiaddr);

                dhcp_client->state = REQUESTING;
```

```
static int send_request(GDHCPClient *dhcp_client)
{
        struct dhcp_packet packet;

        debug(dhcp_client, "sending DHCP request (state %d)",
                dhcp_client->state);

        init_packet(dhcp_client, &packet, DHCPREQUEST);

        packet.xid = dhcp_client->xid;
        packet.secs = dhcp_attempt_secs(dhcp_client);

        if (dhcp_client->state == REQUESTING || dhcp_client->state == REBOOTING)
                dhcp_add_option_uint32(&packet, DHCP_REQUESTED_IP,
                                dhcp_client->requested_ip);

        if (dhcp_client->state == REQUESTING)
                dhcp_add_option_uint32(&packet, DHCP_SERVER_ID,
                dhcp_client->server_ip);
```

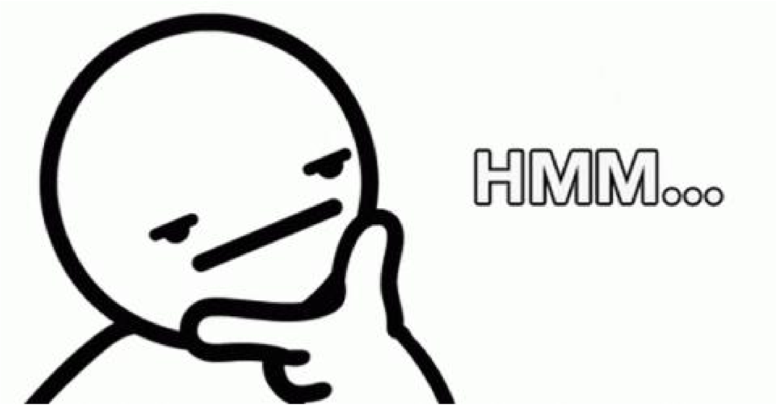
```
uint8_t *dhcp_get_option(struct dhcp_packet *packet, int code)
{
        int len, rem;
        uint8_t *optionptr;
        uint8_t overload = 0;

        /* option bytes: [code][len][data1][data2]..[dataLEN] */
        optionptr = packet->options;
        rem = sizeof(packet->options);
```

# How to exploit these bugs with 0 clicks:

- Tesla Service WiFi network.
  - Parked Tesla vehicles scan for and connect to the SSID “Tesla Service”. The WPA2-PSK credentials can be obtained from firmware or Twitter :-)
- Use wpad to forward requests to local ConnMan DNS
- Use DHCP to leak stack
  - allows to determine libc base and stack base
  - also allows to fingerprint software version running
- Trigger bug to get RCE on the Infotainment
  - Reverse TCP connect fetching 2nd stage payload. Pages containing this payload are made executable and 2nd stage executed.
- *Sidenote: Attack should also have been possible over cellular network*

# Still a month of time until the contest...

- Lets see how far we can take this…




We actually overshot target: Exploit ended up working against all Tesla models (S, 3, X, Y) produced post mid-2018

# How to root the infotainment

- Connman running under its own user
- All processes restricted via:
  - Kafel: syscall filtering
  - Apparmor: resource access limiting
- Connman cannot launch /bin/sh
- But can execute modprobe with restrictions
  - Only Tesla signed modules allowed
  - But some modules are loading firmware
  - For example: BCMDHD driver

# Escalation ping-pong

- Idea: Load a malicious WiFi firmware to the SoC and attack the Infotainment Kernel from the WiFi SoC side.
- DMA attack: Cannot be done as IOMMU is in place.
- BCMDHD[1] driver has had “some” bugs in the past so let's continue there:
  - Result: promising bcopy() OOB with controlled length.

[1] https://googleprojectzero.blogspot.com/2017/04/over-air-exploiting-broadcoms-wi-fi_4.html

# Result of our PWN2OWN 2020 attempt

- Gained 0-click RCE on the Infotainment
- Found possible way to escalate to root
- Even tho we were assured the contest would happen, the automotive category was canceled some days before the contest.
- Frustrated we stopped working on the escalation vector

KUNNAMON

Comsecuris

# But why drones?

- Fun
- Launch attack (stealthy) from up to 100m above
- Fly drone to (Super)charger...
- or other spots with a high Tesla incidence rate

# TBONE: *First public Tesla 0-click attack

# What to do with these bugs/exploits?

- Sit on them until PWN2OWN 2021?
- Will there be bug collisions before March 2021? OTOH, these bugs have been in ConnMan since the beginning.
- With no end of the pandemic in sight: will the event even happen?
- In October 2020 we decided to submit to Tesla's bug bounty.

# Reporting aftermath

- Intel PSIRT doesn't think it's responsible
- But no one else is using Connman anyway right?


- Turns out Connman is default on "Automotive Grade Linux" and recommended by GENIVI

# Obsolete: Connman

- If you are using ConnMan on your Projects you should strongly consider switching to an alternative.
- Tesla migrated to dnsmasq


*To be continued...*

# Conclusion

- From the perspective of attackers, infotainment systems have become similar to desktop systems (shift from QNX, VxWorks and other RTOSes to Linux)
- Stack buffer overflows still a problem in 2020; still exploitable despite mitigations
- Understand the bugs you fuzzed lest you miss the real gems
- Automotive research is possible without actual hardware

# Thanks to

- Tesla Security Team
- CERT-Bund (German CERT)
- Team Kunnapwn 2021