Exploitation on Xtensa/ESP

Philipp Promeuschel & Carel van Rooyen

Security analysts (Compass & ex-Compass)

Overview

- JS abstraction caveat
- Mitigations are not in use on new devices
- Mongoose OS / OS layering
- Attack surface exploration… without JTAG, no further inspection

- IoT Cyber killchain
- Paper - origins of the talk
- Exploit / ROP / Persistence / CnC (remote patching)

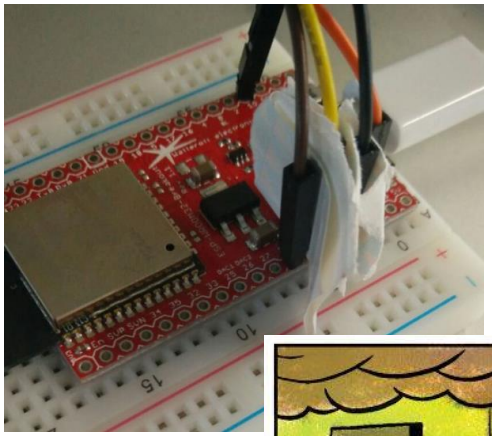
- $2 security
- Approaches for protection of your projects

- Q&A

$ ls /home/philipp

- IT Security Analyst at Compass Security
  - Web
  - Mobile
  - IoT
- Likes to break stuff (not repairing)
- Cannot solder (fumes man)

- Low attention span, easily distracted



Contact: https://twitter.com/nks0ne

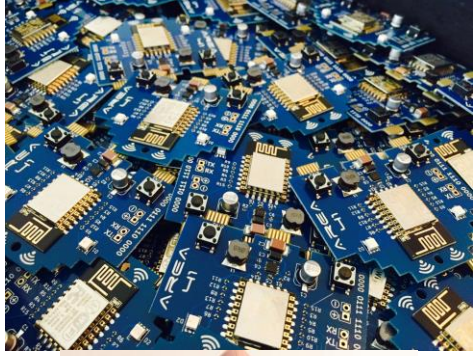
/dev/null

- Previously
  - a lecturer in web dev
  - security analyst at Compass Security Switzerland
- Likes puzzling over broken things
- Can solder

- Bass guitar, jazz, death metal, death jazz

A bit of a storyline

DC4131 Area41

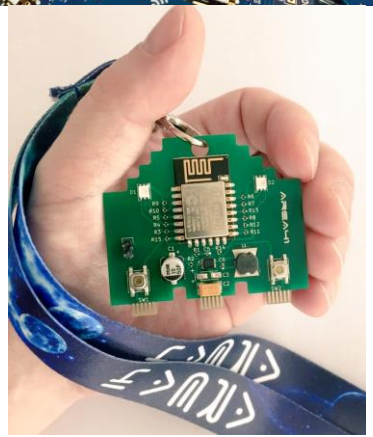


It all started at Area41

Got the badge

Did some reading on the ESP8266 - got excited

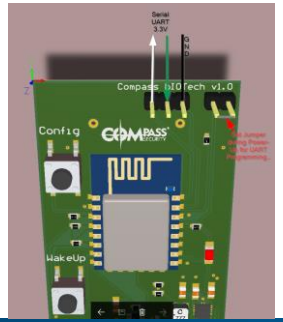
Order the follow up ESP32 to build an “I’ll be home at xx:xx” e-ink thingamajig



Colleague built bIOTech device for Swiss Cyber Storm (caveat)

Real world customer claiming:

“You cannot exploit this there is no OS” - meanwhile...



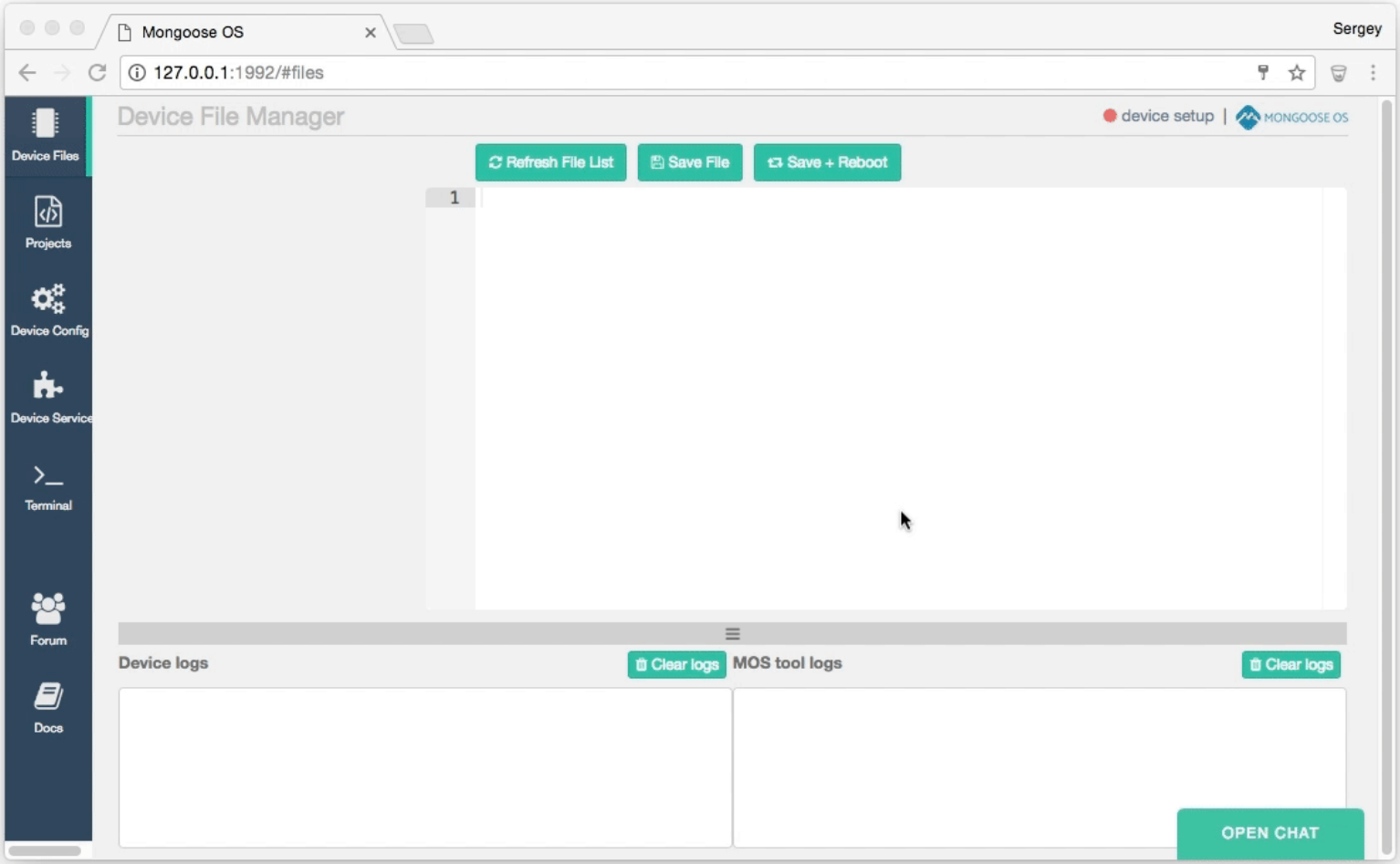
“Security and privacy are silent prerequisites. However, not too many stakeholders will talk about them.

But everybody takes them for granted.”...

...“The problem:
Securing a solution usually
needs time

- a strong contradiction to the market requirements”

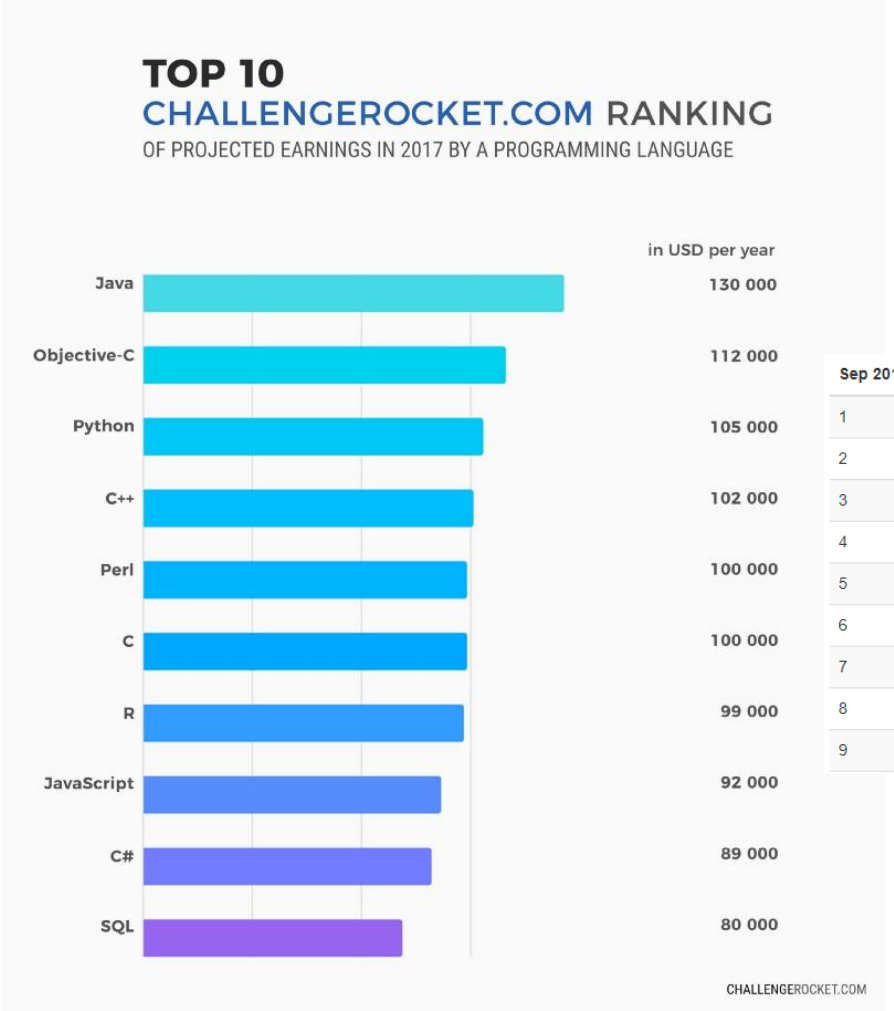
Stefan Grasmann
Zühlke Engineering Gmbh


Mongoose OS
Sergey
127.0.0.1:1992/#files
Device File Manager
device setup | MONGOOSE OS
Device Files
Projects
Device Config
Device Service
Terminal
Forum
Docs
Refresh File List
Save File
Save + Reboot
1
Device logs
Clear logs
MOS tool logs
Clear logs
OPEN CHAT

MONGOOSE OS APPS

This is a collection of apps contributed by the community and Mongoose OS development team. Each app is a ready-to-go firmware, which could be built, flashed and used using the `mos` tool. Also, many apps serve a tutorial purpose - for example, how to use a particular piece of hardware. Each app links to the corresponding GitHub repository, which has an app description and usage instructions. See documentation on how to build and contribute your own app.

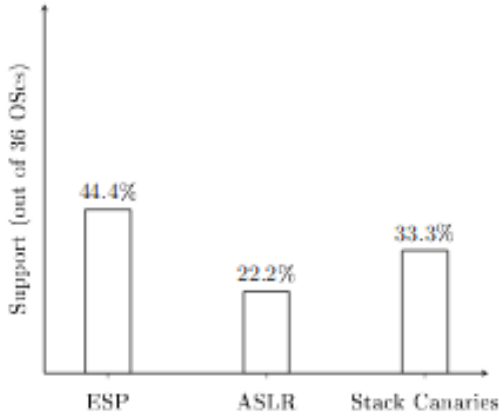
CLOUD INTEGRATION			
alexa-microwave	Control a microwave with ESP32 via MQTT / Amazon Alexa	js cloud	Steve Kasuya
aws-iot-button	Internet button on AWS IoT	js c cloud aws	mongoose-os
aws-iot-heater	Smart heater on AWS IoT	js c aws cloud	mongoose-os
aws-uart	Reading UART0 and sending the data to AWS IoT	js cloud aws example uart	bravokeyl
blynk	Using Blynk mobile app with Mongoose OS	js c mobile example cloud	mongoose-os
blynk-bme280-js	Using BME280 sensor with Blynk mobile app	js mobile cloud arduino	Rumen Nikiforov
losant-motion-sensor	Losant - detecting Motion Using a PIR sensor	js cloud	Taron Foxworth
losant-mqtt	Losant cloud service integration	js cloud	Taron Foxworth
neopixel-aws-iot	Control Neopixels from AWS IoT and an Android Companion App	js aws cloud neopixel	anelson
sonoff-basic-openhab	Sonoff Basic firmware to work with openHAB	cloud c	Michael Fung



Sep 2017	Sep 2016	Change	Programming Language	Ratings	Change
1	1		Java	12.687%	-5.55%
2	2		C	7.382%	-3.57%
3	3		C++	5.565%	-1.09%
4	4		C#	4.779%	-0.71%
5	5		Python	2.983%	-1.32%
6	7	▲	PHP	2.210%	-0.64%
7	6	▼	JavaScript	2.017%	-0.91%
8	9	▲	Visual Basic .NET	1.982%	-0.36%
9	10	▲	Perl	1.952%	-0.38%

Back to the 90's

- Almost no mitigation deployed in IoT
- Wat iz mitigation?
  - ASLR
  - ESP
  - Stack Canaries

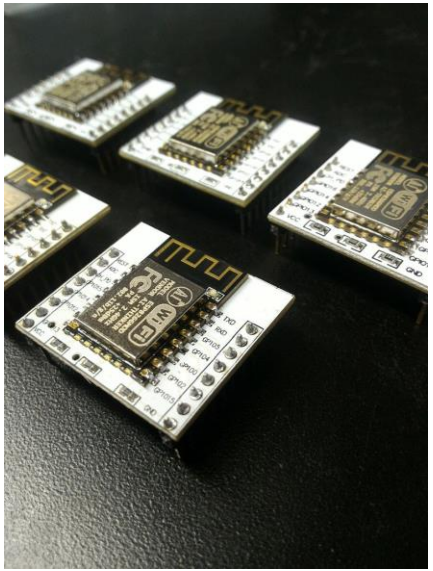


(b) Non-Mobile OSes

- Why are there no mitigations?
  - Limited resources
  - Power saving
- However: .... exploit mitigations are only likely to see widespread industry adoption if the average-case imposed code size, memory and runtime performance overhead is between at most 5 and 10 %.
  - Jos Wetzels

Lost in translation, or when did this become so popular?

“The chip first came to the attention of **western makers in August 2014** with the ESP-01 module, made by a third-party manufacturer, AI-Thinker. However, at the time there was almost **no English-language documentation** on the chip and the commands it accepted.The very low price and the fact that there were very few external components on the module which suggested that it could eventually be very inexpensive in volume, attracted many hackers to explore the module, chip, and the software on it, as well as to translate the Chinese documentation.” - Wikipedia ESP8266



But today…

(picture credits Cesanta)

ESP32 $1

I'm ESPecial...

Easy cloud integration - AWS, Google Cloud, Azure

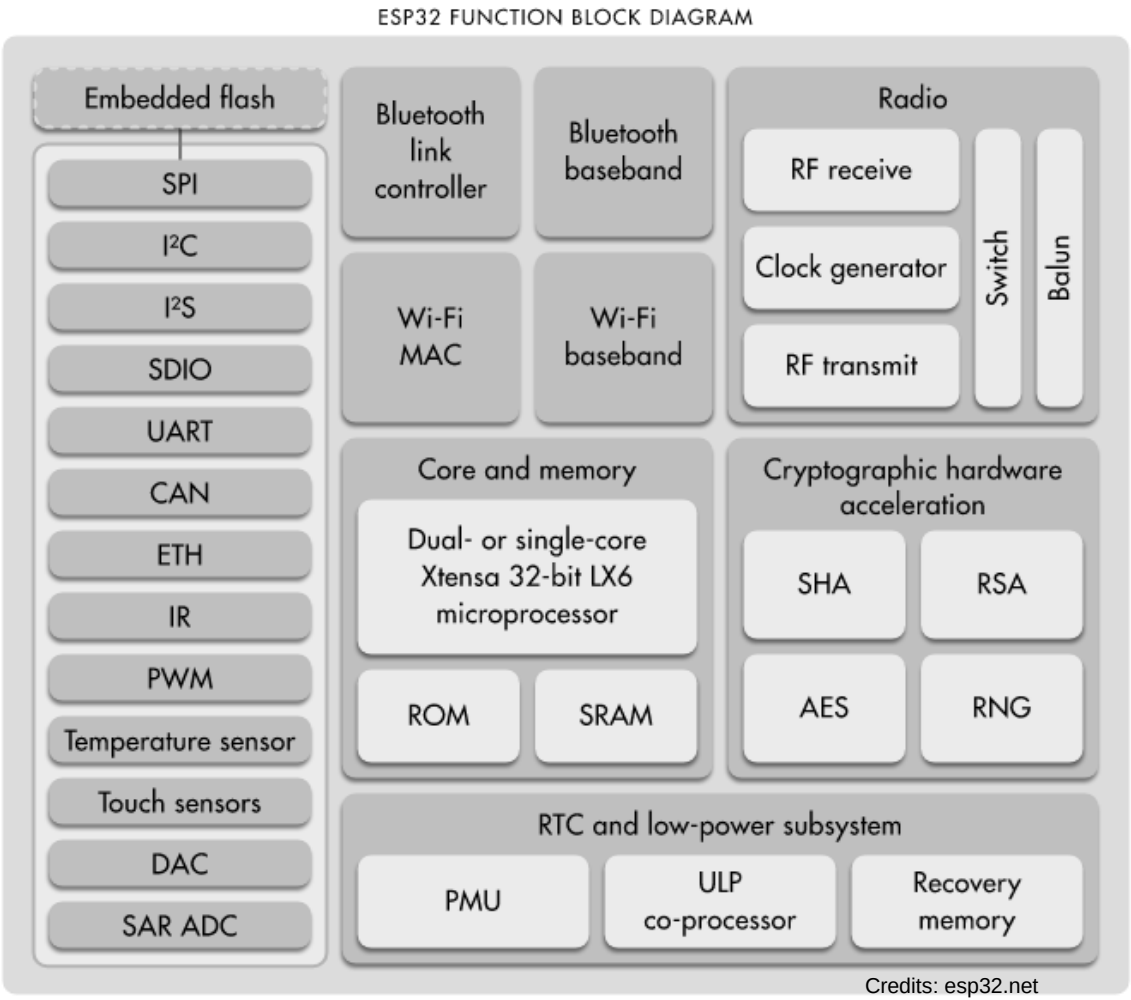
Apps can use Light Weight IP (LWIP)

Dual core (thread for comms, thread for app - FreeRTOS)

Bluetooth

WiFi

GPIO



Credits: esp32.net

Abstraction caveat,

or how to not be Leonard Cohen

Trading infrastructure awareness for decreased learning curve

Nobody knows there is an OS

Nobody knows memory management is handled

Nobody knows about TCP/IP magic

Blackboxes need **safe defaults** - ASLR / code signing / overflow handlers

Credit: WikiCommons

How everything fits together

Turtles all the way down

xtensa OS - XTOS
(hardware abstraction, interrupt and exception handlers)

real time OS - freeRTOS
(preferred OS of Xtensa)

Mongoose OS
(only needed libraries and functionality is compiled in
less reliable)

Attack surfaces

Data at rest

Data in motion

All interfaces, user input

Information leakage - is your avant guard IoT

startup’s intellectual property being pushed to the cloud?

Hardware guys (voltage glitching etc)

Attack Surface	Vulnerability
Ecosystem Access Control	• Implicit trust between components • Enrollment security • Decommissioning system • Lost access procedures
Device Memory	• Cleartext usernames • Cleartext passwords • Third-party credentials • Encryption keys
Device Physical Interfaces	• Firmware extraction • User CLI • Admin CLI • Privilege escalation • Reset to insecure state • Removal of storage media
Device Web Interface	• SQL injection • Cross-site scripting • Cross-site Request Forgery • Username enumeration • Weak passwords • Account lockout • Known default credentials
Device Firmware	• Hardcoded credentials • Sensitive information disclosure • Sensitive URL disclosure • Encryption keys • Firmware version display and/or last update date
Device Network Services	• Information disclosure • User CLI • Administrative CLI • Injection • Denial of Service • Unencrypted Services • Poorly implemented encryption • Test/Development Services • Buffer Overflow • UPnP • Vulnerable UDP Services • DoS
Administrative Interface	• SQL injection • Cross-site scripting • Cross-site Request Forgery • Username enumeration • Weak passwords • Account lockout • Known default credentials • Security/encryption options • Logging options • Two-factor authentication • Inability to wipe device
Local Data Storage	• Unencrypted data • Data encrypted with discovered keys • Lack of data integrity checks
Cloud Web Interface	• SQL injection • Cross-site scripting • Cross-site Request Forgery • Username enumeration • Weak passwords • Account lockout • Known default credentials • Transport encryption • Insecure password recovery mechanism • Two-factor authentication
Third-party Backend APIs	• Unencrypted PII sent • Encrypted PII sent • Device information leaked • Location leaked
Update Mechanism	• Update sent without encryption • Updates not signed • Update location writable • Update verification • Malicious update • Missing update mechanism • No manual update mechanism
Mobile Application	• Implicitly trusted by device or cloud • Username enumeration • Account lockout • Known default credentials • Weak passwords • Insecure data storage • Transport encryption • Insecure password recovery mechanism • Two-factor authentication
Vendor Backend APIs	• Inherent trust of cloud or mobile application • Weak authentication • Weak access controls • Injection attacks
Ecosystem Communication	• Health checks • Heartbeats • Ecosystem commands • Deprovisioning • Pushing updates
Network Traffic	• LAN • LAN to Internet • Short range • Non-standard

Botnet creation through Xtensa exploitation of Mongoose OS - draft

Philipp Promeuschel[1], Carel van Rooyen[1], Dobin Rutishauser[1], Stephan Sekula[2]

***Abstract*—This proof-of-concept IoT botnet implementation is based on IoT devices running Mongoose OS, an open source operating system, and also a framework for the rapid development of IoT projects, built on top of FreeRTOS, which was initially released in 2015[1]. Abusing buffer overflows and other vulnerabilities, a network of adversarially controlled IoT devices can be built, for instance, by executing the Mirai or equivalent malware, which will drive the devices to connect back to an attacker-controlled C&C server.**

In this paper, the issue under scrutiny is the security mechanisms integrated with the Mongoose OS on Xtensa platform systems. Our approach outlines first steps into exploitation techniques tailored to this special target environment. This paper aims at describing employed technologies, techniques, and the preliminary results. It also tries to encourage more research and tool development on this topic. We further aim to share knowledge about the exploitation of new platforms and the approaches taken. Further interpretation shows that overlooked security best practices from the past can again be exploited by future adversaries, and need to be guarded against in IoT environments.

I. Introduction

Recent high-profile botnet attacks[2] already focused on using default passwords, or unauthorized firmware updates, to gain access to and proliferate on devices, in some cases even requiring replacement of the actual devices in order to mitigate issues[3]. However, no worm-like and simultaneously persistent botnet behaviour has yet been detected in the wild, as default password attacks merely rely on the fast reinfection of said devices after reboot. Up to now these attacks have typically targeted printers and routers, for which a possible explanation may be the lack of adequate generic attacks on the firmware and hardware for the devices.

While it has been derided as stunt hacking in the last couple of years, we argue that IoT device hacking is more relevant than ever today. Buchanan *et al.*[4] explains Schacham's return-oriented-programming[5] is *"a technique by which W⊕X-style hardware protections are evaded via carefully crafted stack frames that divert control flow into the middle of existing variable-length x86 instructions creating short new instructions streams that then return. We believe this attack is both more general and a greater threat than the author appreciated"*, and goes further to portend that this is a universal issue, possibly extending beyond SPARC and x86 into other future instruction set architectures (ISA). SPARC has similarities to Xtensa in that it employs register windows, which makes the exploitation thereof different from x86 and ARM exploitation (more later).

[1] Security Analyst, Compass Security Switzerland
[2] Security Analyst, Compass Security Germany

IoT devices with a decent amount of processing power, such as the Espressif ESP8266 and ESP32[6] have recently experienced a major spike in interest from the general public. From an attacker point of view, apart from being powerful, these devices are even more interesting due to their possible cloud integration. Hence, once compromised the connected infrastructure and networks surrounding the device can also be attacked through techniques like pivoting.

Also, much of the development complexity has been abstracted away by frameworks like Mongoose OS, which already provide the necessary development tools, and it has never been so easy to implement hardware-based/embedded products. In the case of Mongoose OS, native Amazon Web Services (AWS) IoT cloud support with message queue telemetry transport (MQTT) is included, and typical services, like a basic web server, can be copied, bootstrapped, and flashed to the device within a couple of minutes. Creator of the dual-licensable Mongoose operating system, Cesanta, also has a notable set of clients from Dell, Samsung, HP, Bosch, Nintendo, Epson, Google, Sky, Qualcomm, and Intel, among others listed on their website. There is some evidence to suggest that the possible deployment base of the target operating system and its top two listed hardware platforms would be potentially large[7] as part of a growing market in home automation and sensor streams[8].

II. The target

The hardware part of our target consists of the current version of the low-cost ESP32 chipset[9] which includes Bluetooth Low-Energy (BLE). The Xtensa ISA focuses on delivering *"a high degree of extensibility, industry-leading code density, optimized low-power implementation, high performance, and a low-cost implementation"*[10]. The Xtensa cores utilized in the ESP32 and their interaction with memory and peripherals are depicted in figure 1 on p. 2.

The software part is Mongoose OS, layered on top of FreeRTOS, an open source IoT operating system supporting low-power connected micro-controllers. It features an integrated web server as well as transparent file system encryption, the easy integration and usage of crypto chips, embedded JavaScript engines (using either mJS, with its strict subset of ES6[11], or v7), and AWS integration.

III. Hello world

Getting a test environment up and running is surprisingly easy and timeous setup was possible, i.e. a couple of minutes. Connecting the device to the development system, installing Mongoose's *mos tool*, and executing the Web UI wizard is enough to get any developer, without the need for hardware knowledge, started.

We started by getting a basic web server up and running, which we tinkered with. This was followed by basic code evaluation. After some initial code and deployment tests, it became clear that that many typical protection mechanisms, like address space layout randomization (ASLR)[12], were not implemented on the system. Thus, a proof-of-concept

Things to do

1. Exploit
2. Remote code execution
3. Persistence
4. Command & Control

Code Redirection - Is it possible?

Xploit Xtensa - Is it even possible on Xtensa? How?

We wanted to prove code execution is possible

```c
#include <stdio.h>
int nevercalled(){
  printf("+ exploit test...");
}
int main(int argc, char **argv)
{
  char buf[16] = "";
    strcpy(buf, "AAAABBBBCCCCDDDDEEEEFFFF");
    return 0;
}
```

What happens?

- Crash happens
- Overwriting local registers A2, A3, A4, A5, A6, A7, A8
- Overwriting A0 and A1
- What happens exactly
  - Buffer write out of bounds, after ret.w the overwritten values will be restored, first bytes of overflow land in A0
- Control over execution flow

Demo: Buffer overflow

http://bit.ly/hw-io_buffer_overflow

CVE-2017-7185 Detail

Modified

This vulnerability has been modified since it was last analyzed by the NVD. It is awaiting reanalysis which may result in further changes to the information provided.

Current Description

Use-after-free vulnerability in the mg_http_multipart_wait_for_boundary function in mongoose.c in Cesanta Mongoose Embedded Web Server Library 6.7 and earlier and Mongoose OS 1.2 and earlier allows remote attackers to cause a denial of service (crash) via a multipart/form-data POST request without a MIME boundary string.

Source: MITRE **Last Modified:** 04/10/2017 + View Analysis Description

Quick Info

CVE Dictionary Entry: CVE-2017-7185
Original release date: 04/10/2017
Last revised: 08/15/2017
Source: US-CERT/NIST

Impact

CVSS Severity (version 3.0):

CVSS v3 Base Score: 7.5 High
Vector: CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H (legend)
Impact Score: 3.6
Exploitability Score: 3.9

CVSS Version 3 Metrics:

Attack Vector (AV): Network
Attack Complexity (AC): Low
Privileges Required (PR): None
User Interaction (UI): None
Scope (S): Unchanged
Confidentiality (C): None
Integrity (I): None
Availability (A): High

CVSS Severity (version 2.0):

CVSS v2 Base Score: 5.0 MEDIUM
Vector: (AV:N/AC:L/Au:N/C:N/I:N/A:P) (legend)
Impact Subscore: 2.9
Exploitability Subscore: 10.0

CVSS Version 2 Metrics:

Access Vector: Network exploitable
Access Complexity: Low
Authentication: Not required to exploit
Impact Type: Allows disruption of service

Mongoose Remote DoS

Crash while handling HTTP multi-part requests

Identified by manual fuzzing http headers

Deep dive in source identified that initially the request is handled properly, but then the exception is not properly handled. Causing the multi-part handling code to read from an uninitialized buffer -> crash

```
5961      struct mbuf *io = &c->recv_mbuf;
5962      struct mg_http_proto_data *pd = mg_http_get_proto_data(c);
5963
5964  +   if (pd->mp_stream.boundary == NULL) {
5965  +     pd->mp_stream.state = MPS_FINALIZE;
5966  +     DBG(("Invalid request: boundary not initilaized"));
5967  +     return 0;
5968  +   }
5969  +
```

MONGOOSE OS

Home Enterprise Downloads Developers Blo

Security advisory on Mongoose networking library

11 APRIL 2017

We have received a notification from security research organisation recently about Mongoose Networking library vulnerability.
The advisory was concerning handling of the multipart upload code:
http://seclists.org/fulldisclosure/2017/Apr/8 .

Prior to making that disclosure public, we have updated our customers and then released the public patch and a stable branch https://github.com/cesanta/mongoose/tree/6.7.1.

The advisory tells about denial of service on Mongoose OS. However it should be noted that on low-power microcontrollers which Mongoose OS targets, it is very trivial to do a denial of service if a microcontroller acts as a server (due to the limited RAM available). Just fire several netcat sessions from your terminal and your microcontroller is down, so there is no need to exploit any vulnerabilities.

Both Mongoose OS and Mongoose Networking library are fixed at this moment. Please make sure you're using the latest stable version.

As a security best practice we recommend to avoid using device in the server mode. Instead make it a client, talking to a backend, reporting data and reacting on commands. That way you will prevent the large class of security attacks.

Sergey Lyubka

Cesanta CTO and co-founder. Former Googler.

Dublin, Ireland https://cesanta.com

Share this post

O rly?

Thanks Dobin...

```
To be released (still awaiting CVE #)
Affected:
---------

Mongoose - Embedded Web Server /
Embedded Networking Library:

Vulnerable:
 * <= 6.9

Not vulnerable:
 * >= 6.10
```

Latest release

6.9

62c896a

Mongoose 6.9

rojer released this 6 days ago

API changes:

- `MG_ENABLE_CALLBACK_USERDATA` - if set, changes signature of event handler function to include user_data argument. Disabled by default for now, in the future this will become the default.
- `mg_set_nameserver()` - specify DNS server to use
- `mg_assemble_uri()` - assemble a URI from parts
- `mg_connect_ws()` now accepts `http://` URLs

Bug fixes:

- Fix parsing of MQTT QoS > 0 PUBLISH messages
- Fix MQTT PUB{ACK,REC,REL,COMP} and UNSUBACK flags
- Properly shut down the SSL connection (send "close notify" TLS message)
- Fix `mg_get_http_var()` return value
- Fix MQTT handshake; change client protocol to version 3.1.1
- Fix Handling of multiple MQTT messages per RECV event
- Update to make lwip_net_if thread-safe
- Use DhcpNameServer on Windows
- Fix MQTT message parsing issues
- Fix DNS name uncompression that could lead to infinite loop
- Fix WS frame reassembly issues

#855

https://github.com/cesanta/mongoose/releases/tag/6.9 - CVE incoming with RCE

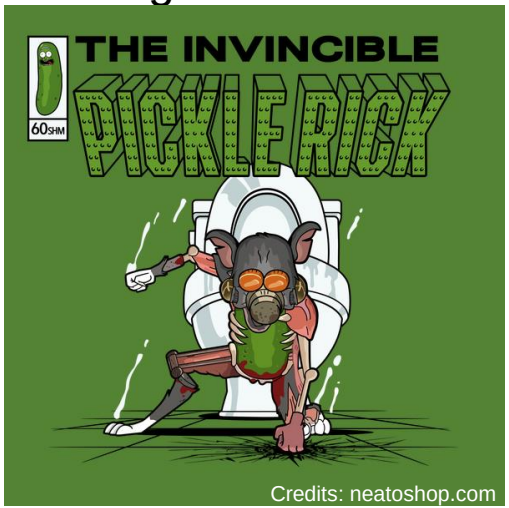
What are ROPchains?

Why? Because the stack is not executable!

Return oriented programming (ROP)

And what does **altering the execution flow** of fractions of existing code **have to do with hardware**?

Build mutations (**repurpose** things)
from existing code (**existing** things)
in order to “be the pickle” (do **interesting** things).



Credits: neatoshop.com

Gadgets: real or fake

What is perfect and what is fake exactly?

load_regs:

```
l32i.n a12, a1, 4;
l32i.n a13, a1, 8;
l32i.n a14, a1, 0xc;
l32i.n a15, a1, 0x10;
l32i.n a0,  a1, 0;
addi   a1,  a1, 0x20;
ret.n
```

Load data from stack into register
A12 = *(A1 + 4)
…
A0 = *(A1 + 0)
A1 = *(A1 + 32)

write_mem:

```
s32i.n a14, a15, 0x0;
l32i   a0,  a1,  0;
addi   a1,  a1,  0x4;
ret.n
```

Store the data in register a14
Into address stored in 15

*(A15+0) = a14
A0 = *(A1 + 0)
A1 = *(A1 + 4)

sync:

```
isync;
isync;
l32i a0, a1, 0;
addi a1, a1, 0x4;
```

Sync (flush caches etc)
So new code is visible

Persistence

In RAM at runtime (non-persistent)
Pew pew

In “ROM” persistent, but destructive
Pew pew pew

In patched “ROM”
Pew ELF pew redirect entrypoint pew pew

Flash:

- SPI Flash
  - Enable Write
  - Flash Patch
  - Reboot

Checksums

- XOR Checksum
- SHA Checksum Bypass (just remove it)

```
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align   Section to Segment mapping
  LOAD           0x000000 0x3f3fff30 0x3f3fff30 0x156a8 0x156a8 RW  0x1         .flash.rodata
  LOAD           0x0156b0 0x3ffb0000 0x3ffb0000 0x01c54 0x01c54 RW  0x1         .dram0.data
  LOAD           0x017308 0x3ffb1c58 0x3ffb1c58 0x00000 0x07d20 RW  0x1         .dram0.bss
  LOAD           0x017304 0x40080000 0x40080000 0x19fc6 0x19fc6 R E 0x1         .iram0.vectors .iram0.text
  LOAD           0x0312cc 0x400d0018 0x400d0018 0x77e19 0x77e19 R E 0x1         .flash.text
```

```
Section Headers: (A only)
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 2] .iram0.vectors    PROGBITS        40080000 017304 000400 00  AX  0   0  4
  [ 3] .iram0.text       PROGBITS        40080400 017704 019bc6 00  AX  0   0  4
  [ 4] .dram0.data       PROGBITS        3ffb0000 0156b0 001c54 00  WA  0   0 16
  [ 5] .dram0.bss        NOBITS          3ffb1c58 017308 007d20 00  WA  0   0  8
  [ 6] .flash.rodata     PROGBITS        3f400010 0000e0 0155c8 00  WA  0   0 16
  [ 7] .flash.text       PROGBITS        400d0018 0312cc 077e19 00  AX  0   0  4
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

$2 IoT solution

Mongoose OS + ESP8266/ESP32 + ATECC508 + Cloud

Crypto chips (ATECC508A - sha-1) - basically your device dies on deployment

IoT devices are basically “dumb unprotected mobile“

- the same rules apply -

IoT & Mobile OWASP top ten (devices without mobile protections)

Securing your project

“you can’t update hardware over the air” - HW.io speaker

Test all the things, **test outside of expectations**

Do risk analysis

Test all **integrations**

Test the ecosystem **holistically** (secure comms power implications?)

Securing communications,

- KeyStore like functionality,
- certificate pinning,
- mutual authentication
- secure OTA fetching

Securing the device

- Libraries
- Other updates

Future Work

Future work

Possibility for intelligent fuzzing

Persistence shellcode

Implement protection:

“**Luckily, most of the mitigation techniques have already been Invented.” - Anonymous**

Our thanks

goes out to

You - for your attention - hopefully feedback

Area41 for the badges

Cesanta for Mongoose OS

JS **Andin** / Neil Kolban for their work and reference work

Colleagues at Compass Security (Dobin Rutishauser, Stephan Sekula) - as wel as our pr0of r3aders

Our employers for the workshop preparation and talk time

People that **pixeled things** for our unsanctioned and out of context reuse here… :)

Where t0 st4rt

Buy ESP32 IoT Dev Kit

Buy Shikra (optional)
Buy Cables (optional)

Wait for delivery

Install esp-idf
Download and flash code

Q&A

Call for Collaboration

Get in touch

We’re still early stages on this topic.

Get in touch if you’re interested, have topic ideas, or want to collaborate on talk ideas.

Link to challenge: https://www.hacking-lab.com/sh/iot - grab a device

Contact details: Philipp.promeuschel@compass-security.com / @carelvanrooyen (twitter DM for email)

Page intentionally left blank

Xtensa Instructions

Load & Store
l32i
l32r
s32i

Jump & Call
j, jx
call0 and call**x0**
ret

Branches (Conditional)
beq
bge
bne
bnez
...

Arithmetic
add
addi
sub
subi
...

Logical
xor
or
and

Move Data & Memory Operations
movi, movz, …
movsp
entry

Buffer overflow - super 1 slide introduction

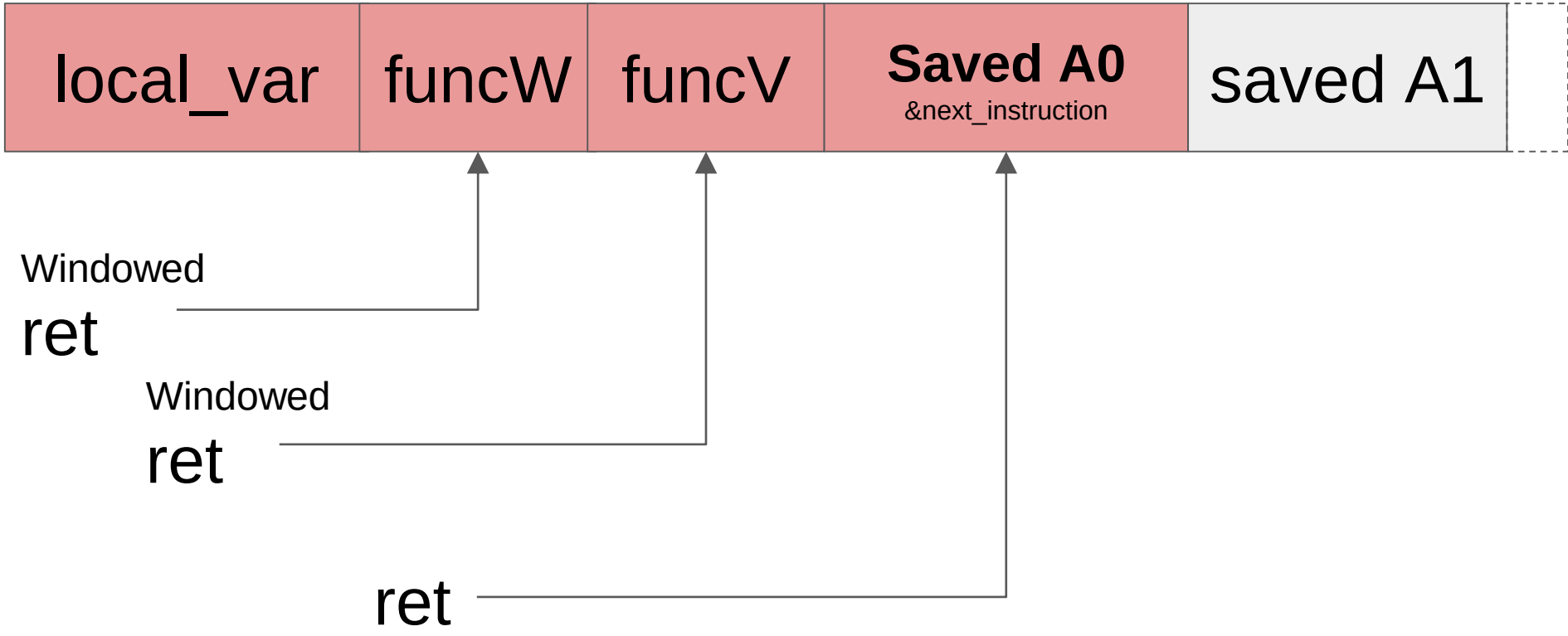
Xploit Xtensa - Is it even possible on Xtensa? How?

```
#include <stdio.h>
int nevercalled(){
  printf("+ exploit test...");
}
int main(int argc, char **argv)
{
  char buf[16] = "";
    strcpy(buf, "AAAABBBBCCCCDDDDEEEEFFFF");
    return 0;
}
```

What happens?

- Crash happens
- Overwriting local registers A2, A3, A4, A5, A6, A7, A8
- Overwriting A0 and A1
- What happens exactly
  - Buffer write out of bounds, after ret.w the overwritten values will be restored, first bytes of overflow land in A0
- Control over execution flow

Register Windowing: Stack overflow layout



Debugging

Demo - crash dumps

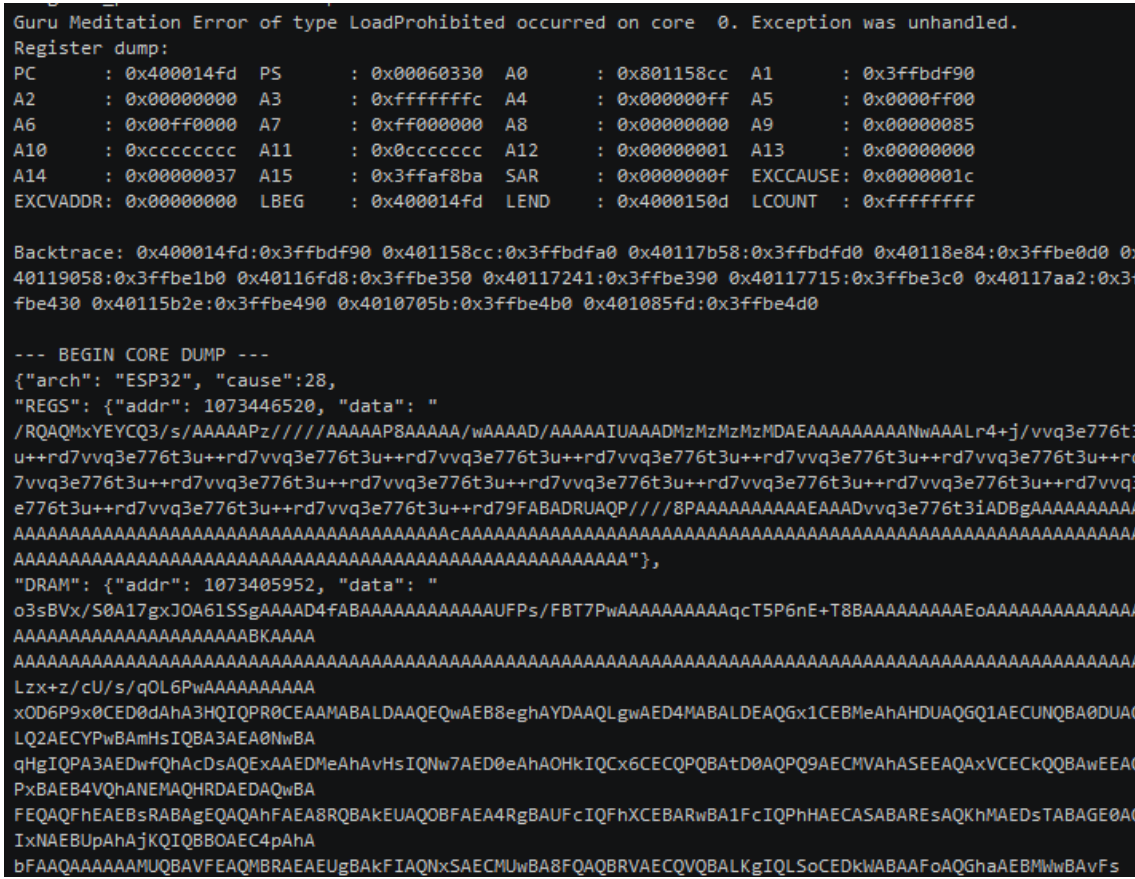
What they are used for?

Crashdumps are not:

“software dynamic testing” approved

Painfully Slow

Only allow **post-mortem** analysis



What we learnt - file system layout

Filesystem is mapped to memory - all ROM parts are executable - whatever you can load into image you can eventually execute - Checksums would be violated on filesystem changes

Start	What	Description	Moar (credit: Dobin Rutishauser)
0x3f3fff30	ELF .flash.rodata		0x3f3f... is not specified anywhere. But loader maps it into memory space (from flash)
0x3ffb0000	ELF .dram0.data		Data RAM
0x3ffb1c58	ELF .dram0.bss	Runtime Stack (SP)	Data RAM
0x40080000	ELF .iram.text		Used for PRO and APP CPU caches. (needs further inspection…)
0x400d0018	ELF .flash.text	Runtime PC points here (Code!)	Really Flash, but also IRAM. Application.