

Automatic Control and Industrial Informatics Department Faculty of Automatic Control and Computers

Improving the security of a webservice: best practices and attack simulations

Table of content

- 1. Abstract & Introduction
- 2. Objectives
- 3. Related work
- 4. Security methods
- 5. Experimental results
- 6. Conclusions

Abstract & Introduction

- Most of the nowadays applications are client-server based
- Main target: expose a webservice from a low-level programming language
- Focus on best practices and security methods

Objectives

- 1. Developing a webservice using C++
- 2. Gradually improve the security of the webservice
 - 1. Check internal and external dependencies
 - 2. Automate the management of resources
 - 3. Encrypt the communication between client and server
- 3. Analyze all data from empirical approach
- 4. Generate a series of conclusions

Related work

• Developed webservice:

- Armhf architecture
- Raspberry PI
- HTU21D humidity and temperature sensor
- Get all data over I2C interface

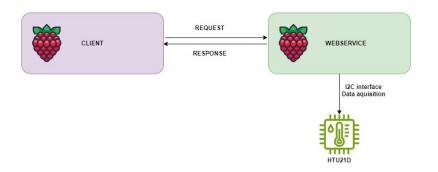


Fig. 1. Data flow from client to server

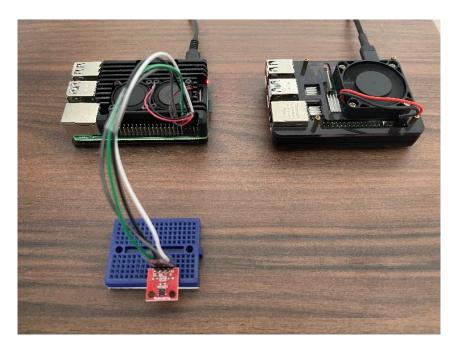


Fig. 2. Physical systems – server and client

Security methods

Checking internal and external dependencies

- How to: check for all exploits found up to this point
 - National Vulnerability Database NIST
 - CVE Numbering Authorities
- **Result:** Avoiding potential exploits / Accepting and analyzing some possible consequences
- Concrete example webservice dependencies:
 - Restbed: no exploits have been reported
 - RapidJSON: no exploits have been reported
 - OpenSSL: all versions have vulnerabilities
 - Ex: version 3.0.11 CVE-2022-1434 (related to RC4-MD5 algorithm)

Security methods

Automating the resource management

- How to: Smart pointers & RAII idiom
- **Result:** Avoiding memory leaks
- Why is important to avoid memory leaks?
 - Such a small and seemingly harmless mistake can represent a significant opportunity for the attacker
 - Talk about a **concrete example**: a webservice which collects data from pressure, temperature and humidity sensors within an automated pipeline for the automotive industry

Encrypting the communication between client and server

- **How to:** digital certificates and HTTPS
- **Result:** encrypted communication end to end (confidentiality and integrity of data)

Experimental results

- Empirical approach for validating the security methods
- All tests were conducted into a controlled environment
- Everything was done solely to validate some results and discover new possible exploits
- Man in the middle attack simulation
 - Method: ARP poisoning
 - Purpose: to validate the effectiveness of the TLS protocol

System	MAC address	
Client	E4:5F:01:45:D5:4C	
Server	D8:3A:DD:19:9B:A8	
Attacker	08:00:27:1E:36:4A	
Table 1. Systems used and their MAC addresses		

Experimental results

ھ File <u>E</u> dit <u>V</u> iew <u>Go</u> <u>C</u> apture <u>A</u> nalyze <u>S</u> tatistics Telephony <u>W</u> ireless <u>T</u> ools <u>H</u> elp	
$\boxed{\blacksquare} \ \boxed{} \ \boxed{ \ \boxed{} \ \boxed{} \ \boxed{ \ \boxed{} \ \boxed{} \ \boxed{} \ $	
http && ip.addr == 192.168.100.56	
No. Time Source Destination Protocol Length Info 2631 30.265727 192.168.100.56 192.168.100.72 HTTP/JS 836 HTTP/1.1 200 OK , JSON + 3582 208.90080 192.168.100.72 192.168.100.76 HTTP 171 GET /getHtu21DataTimes 6062 210.65953 192.168.100.56 192.168.100.72 HTTP/JS 836 HTTP/1.1 200 OK , JSON	eries HTTP/1.1
<pre>> Frame 6062: 836 bytes on wire (6688 bits), 836 bytes captured (6688 bits) on interfad > Ethernet II, Src: RaspberryPiT_19:9b:a8 (d8:3a:dd:19:9b:a8), Dst: PCSSystemtec_1e:36: > Destination: PCSSystemtec_1e:36:4a [08:00:27:1e:36:4a] > Source: RaspberryPiT_19:9b:a8 (d8:3a:dd:19:9b:a8) Type: IPv4 (0x0800) > Internet Protocol Version 4, Src: 192.168.100.56, Dst: 192.168.100.72 Transmission Control Protocol, Src Port: 3600, Dst Port: 41156, Seq: 1264105, Ack: 16 > [truncated]1845 Reassembled TCP Segments (1264874 bytes): #3589(1448), #3590(1448),</pre>	
- Member: operation [Path with value: /operation:getHtu21DataTimeseries] [Member with value: operation:getHtu21DataTimeseries] String value: getHtu21DataTimeseries Key: operation	000000410 65 22 3a 32 34 2e 36 36 32 33 39 32 39 31 39 e":24.66 23992919 00000420 39 32 31 38 39 22 66 69 64 69 74 79 22 92189, "h umidity" 00000430 3a 34 37 2e 39 37 30 31 36 34 37 39 34 39 :47.9970 01647949 00000440 32 32 7d 2c 7d 69 6d 65 22 3a 22 30 32 22}, {"ti me":"202
	Frame (836 bytes) Reassembled TCP (1264874 bytes)

Fig. 3. Man-in-the-middle attack, HTTP protocol, no digital certificate and TLS disabled

Experimental results

د File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help		$\bigcirc \bigcirc \bigotimes$
$\boxed{\blacksquare} \ \boxed{\textcircled{a}} \ \boxed{\textcircled{b}} \ \boxed{\textcircled{b}}$		
R tls && ip.addr == 192.168.100.56		×
No. Time Source Destination Protocol Length Info 103 425.40690 192.168.100.56 192.168.100.72 TLSv1.3 1514 Application Data 103 425.42280 192.168.100.56 192.168.100.72 TLSv1.3 1514 Application Data 104 425.50834 192.168.100.56 192.168.100.72 TLSv1.3 1514 Application Data 104 425.50834 192.168.100.56 192.168.100.72 TLSv1.3 1514 Application Data 104 425.50840 192.168.100.56 192.168.100.72 TLSv1.3 1514 Application Data 105 425.50840 192.168.100.56 192.168.100.72 TLSv1.3 1514 Application Data 105 425.54377 192.168.100.56 192.168.100.72 TLSv1.3 1514 Application Data 105 425.61530 192.168.100.56 192.168.100.72 TLSv1.3 1514 Application Data 105 425.61532 192.168.100.56 192.168.100.72 TLSv1.3 1514 Application Data 105 425.61532		
<pre>> Frame 10576: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on inte > Ethernet II, Src: RaspberryPiT_19:9b:a8 (d8:3a:dd:19:9b:a8), Dst: PCSSystemtec_1e:36:4 > Destination: PCSSystemtec_1e:36:4a (08:00:27:1e:36:4a) > Source: RaspberryPiT_19:9b:a8 (d8:3a:dd:19:9b:a8) Type: IPv4 (0x0800) > Internet Protocol Version 4, Src: 192.168.100.56, Dst: 192.168.100.72 > Transmission Control Protocol, Src Port: 4334, Dst Port: 49776, Seq: 1051795, Ack: 72! > [25 Reassembled TCP Segments (16406 bytes): #10564(294), #10565(1448), #10595(1448), # > Transport Layer Security > TLSv1.3 Record Layer: Application Data Protocol: Application Data Opaque Type: Application Data (23) Version: TLS 1.2 (0x0303) Length: 16401 Encrypted Application Data [truncated]: 85c4ee267379d62900ba0405aeafc64eae863cf6c9</pre>	0010 05 ae af c6 4e ae 86 3c f6 c9 4b c9 be 70 16 02 0020 de 8b d2 95 30 e9 44 5d 70 a2 42 7b 30 b5 6e 99 0030 6b 85 2a 3f 7a 5e 34 d2 6e fd c8 d1 bb 83 f0 dd 0040 83 c2 4b fb 8e cc ff 94 ca 89 6e 18 2a f3 45 6d 0040 83 c2 4b fb 2d ca f1 92 b6 db 2f oe fb f7 b8	@.
Payload is encrypted application data (tls.app_data), 16,401 bytes	Prame (1514 bytes) Reassembled TCP (16406 bytes) Packets: 11212 - Displayed: 74 (0.7%)	Profile: Default

Fig. 4. Man-in-the-middle attack, TLS protocol enabled, digital certificate on server side

Conclusions

- Combined a theoretical and empirical approach to improve the security of a webservice
- Proposed a series of good practices to be applied for avoiding memory leaks
- Our personal thoughts:
 - Digital certificates play an important role in mitigating MITM attacks
 - However, we did not find any scientific research which validates that this method provides 100% protection against MITM
 - The presence of digital certificates and TLS protocol do not guarantee that such an attack cannot be successfully executed

Thank you for your time and attention!