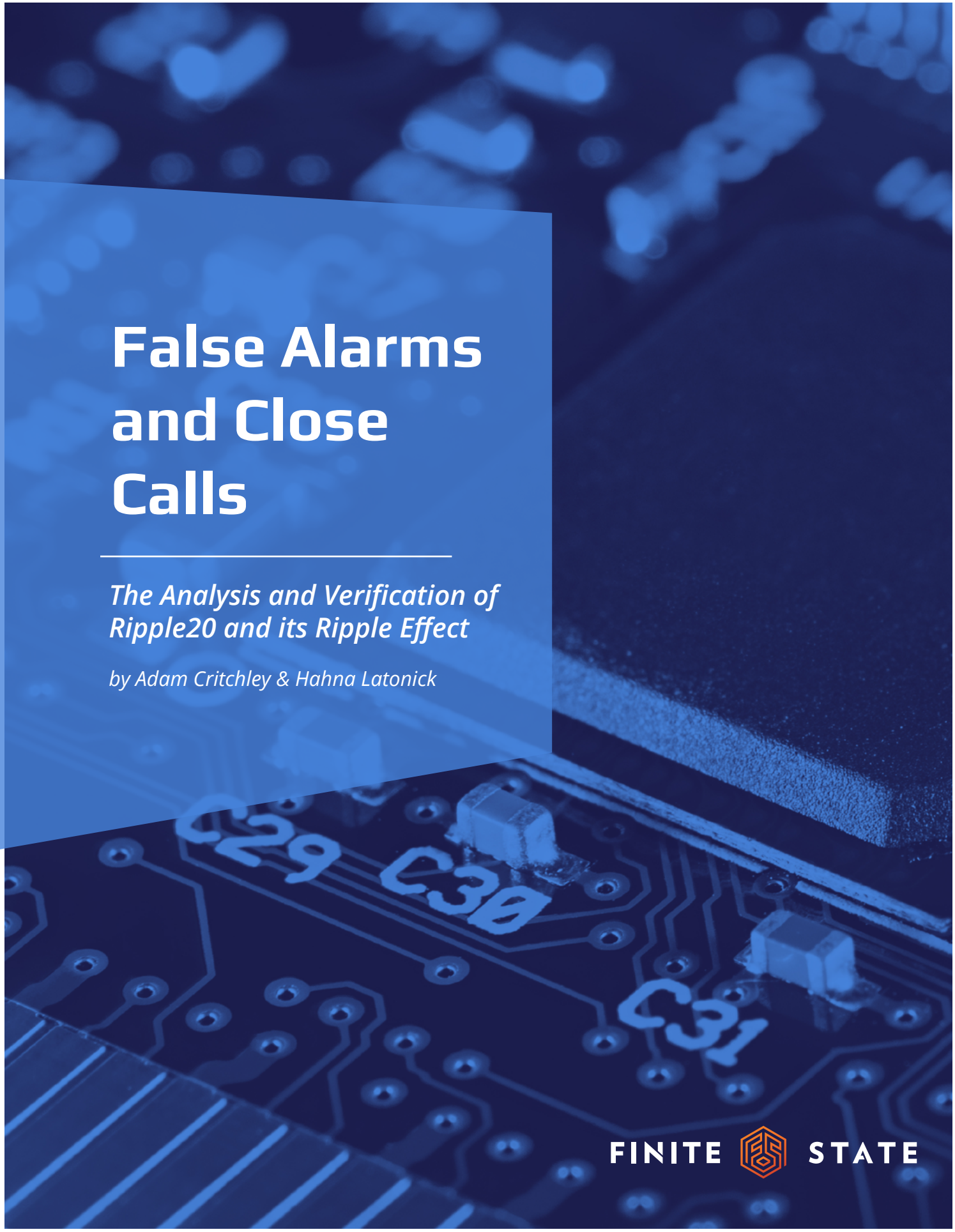


False Alarms and Close Calls

The Analysis and Verification of Ripple20 and its Ripple Effect

by Adam Critchley & Hahna Latonick

FINITE STATE

ABOUT FINITE STATE

Finite State was founded to protect the devices that power our modern lives by illuminating the vulnerabilities and threats within their complex software supply chains. We recognize that supply chain security is the #1 problem in cyber security today. Global software supply chains are opaque and complicated, involving countless developers, vendors, and components. Malicious actors exploit supply chain vulnerabilities to gain access to the networks that power our critical infrastructure and can carry out potentially devastating attacks.

Finite State defends these critical devices, networks, and supply chains by leveraging massive data analysis of device firmware and software to provide transparency to device manufacturers and their customers - enabling them to understand and mitigate their risks before they are compromised.

linkedin.com/company/finitestate
twitter: @FiniteStateInc

Special thanks to Sam Lerner, Octavio Pimentel, Stephanie Pasamonte, Edwin Shuttleworth, and Alex Beigel at Finite State and to Reid Wightman, and Kate Vejda at Dragos.

*** Please note that this paper was updated on 10/21/2020 to reflect additional research, the details of which can be found in our article at https://finitestate.io/2020/10/12/the-aftershock-of-ripple20*

EXECUTIVE SUMMARY

Ripple20 is a collection of 19 CVEs disclosed by JSOF that affect the Treck TCP/IP stack. It has proven to be one of the most widespread vulnerabilities and is elusive to traditional detection techniques due to the many variants spread out over many years of releases. According to JSOF, this series of vulnerabilities affects hundreds of millions of devices and includes multiple remote execution code vulnerabilities, which would allow an attacker to gain complete control over a target device remotely.[1]

Given the serious nature of the vulnerabilities and how they would affect our industry partners, Finite State chose to look into CVE-2020-11896, and CVE-2020-11901, which were the two primary Remote Code Execution (RCE) vulnerabilities presented in the disclosure. They also have the highest CVSS scores of the series. CVSS uses exploitability, scope, and impact metrics to calculate a score between 0 and 10. The scores for the two RCE vulnerabilities were ranked as Critical impact with scores of 10.0 and 9.0 respectively.

Our premier security research team was able to overcome the hurdles to verifying the effects of the two RCE vulnerabilities using a method that we call Focused Emulation. Focused Emulation quickly tests all versions of the device firmware for multiple devices that contain the aforementioned Ripple20 CVEs. This technique also avoids the disruptive nature of testing on a deployed device and the potential for inaccuracy in passive network traffic detection.

Our analysis corpus consisted of 20 firmware images with release dates spanning 2014-2020, different architectures (x86, ARM, MIPS, Coldfire, and SuperH), and operating systems (Quadros 2014/2017, HP OS 2020, Net OS 2017, and GreenHills 2019). We also leveraged JSOF's approach for unpacking HP firmware during our analysis.[2]

What we found is that the CVSS scores for the two devices that JSOF has publicly demonstrated exploits against reflect the scores listed for CVE-2020-11896 and CVE-2020-11901, both of which result in an RCE. However, when evaluating these CVEs against other devices the expected effect varied. For CVE-2020-11896, we have not observed any RCE effects other than those on the devices for which JSOF has published their findings, and have been able to confirm our results in our clients' firmwares. For CVE-2020-11901, we were able to demonstrate a heap overflow on the Digi Connect ME 9210 and the HP OS, which could provide the opportunity for remote code execution (RCE). Our research has shown, however, that most devices we tested that utilize the Treck stack are not affected by the disclosed remote code execution vulnerabilities due to the device's configuration, bringing into question the true widespread impact of Ripple20.

The discrepancies between our results and those originally published underscore a critical need for verification of vulnerabilities across multiple versions of affected devices, but they also indicate that the system for reporting and scoring vulnerabilities may, itself, need to be reworked.

The Ripple20 vulnerabilities are real vulnerabilities that show up in a number of ways—we don't dispute that. In fact, we don't see our results as being in opposition to JSOF's research but rather as an expansion of it, which would not have been possible without the important work that they put in. In this paper, we are disputing the severity of those CVEs, but ultimately the issue is more systemic. The CVE, CPE, and CVSS systems simply don't work for embedded devices, and as a security community we need to find more scalable ways to verify and respond to reported vulnerabilities. In this paper we outline our approach to doing just that.

[1] https://www.jsof-tech.com/ripple20
[2] https://www.jsof-tech.com/unpacking-hp-firmware-updates-part-1/

Verifying the Ripple20 RCE Vulnerabilities

The Treck stack is distributed as source code, giving OEMs the flexibility to modify and select pieces of the code that enable stack functionality. The stack can also target any architecture and device which significantly increases the analysis complexity. Consequently, security teams are required to manually test each device for the possible vulnerability, which is both disruptive and unscalable.

The Ripple20 vulnerabilities are believed to affect a wide range of devices used in every industry. Vendors have released the Treck stack in devices for applications such as medical, transportation, industrial control, enterprise, energy, telecom, retail and commerce which consequently use a diverse range of processors depending on the size, weight, and power constraints of the deployment.

Security teams for both device manufacturers and asset owners have been scrambling to determine whether their devices are affected.

Finding the Right Approach

Rather than relying upon inaccurate network scans, Finite State leveraged our advanced firmware analysis platform to look inside the firmware packages and binaries within them to detect the presence of the vulnerable Treck stack within the assembly code itself.

Finite State has identified the Treck stack in devices utilizing Coldfire, MIPS, ARM, x86, and SuperH processors. Further adding to the complexity of the Ripple20 detection is that updates to the stack can be incorporated piecemeal into the device's source code. Thus, the implementation of the stack for a device will likely be a combination of different versions of the stack. This means that the typical, naive approach of searching for a version string or matching a YARA signature to detect the Treck stack will often be incorrect, as sometimes the version string is inaccurate or not even present in the final firmware.

The Ripple20 vulnerabilities can lead to different effects, such as remote code execution, information disclosure, and denial of service (DoS). Each of these vulnerabilities are important to address and shouldn't be ignored; however, the two remote code execution vulnerabilities, CVE-2020-11901 and CVE-2020-11896, are the most significant of the set due to their CVSS scores of 9.0 and 10 respectively. Of the series, these two CVEs act as enablers to the rest; in other words, without these two CVEs, attackers would be unable to compromise and take full control of the target devices. Thus, Finite State first focused on verifying the severe effects of these two CVEs in an attempt to address the most critical threats facing our customers and industry partners.

Summary of Key Findings

- *Our research has shown that most devices we tested that utilize the Treck stack are not affected by the disclosed remote code execution vulnerabilities, bringing into question the true widespread impact of Ripple20*
- *When verifying the impact on CVEs on other devices, the impact of the CVEs have ranged from Denial of Service to Heap Overflow. The CPE vector should be specific to the devices that were affected, so that the CVSS score is more accurate.*
- *In discussions with our team, Treck confirmed that exploitation results for CVE-2020-11896 were different based on an error checking macro that vendors could choose to enable or not. Defining the error checking macro enabled the "guard code" that was discussed and depicted in our whitepaper. All devices we've encountered, except for the Digi Connect ME 9210, had the guard code. Additionally, another macro could be enabled to support scattered data from the device driver, which completely removes the vulnerable code from the final binary.*
- *For CVE-2020-11901, we developed a new approach to demonstrate a heap overflow on the Digi Connect ME 9210 and the HP OS, which could provide the opportunity for remote code execution (RCE).*
- *Exploitation of CVE-2020-11901 requires DNS to be enabled for the Treck stack. For the firmware that we analyzed, DNS was not enabled and, therefore, the vulnerable code was not present. Even when it was enabled, the Treck DNS code imposed additional constraints which had to be overcome for successful exploitation. Again, the presence, usability, and reachability of the DNS code will vary based on device configurations.*

Figure 1. Summary of exploit results on different Treck variants.

	CVE-2020-11896	CVE-2020-11898	CVE-2020-11901
Quadros 2017	No Effect	No Effect	N/A*
Quadros 2014	DoS	DoS	N/A*
HP OS 2020	DoS	DoS	Heap Overflow, Possible RCE
Net+OS 2017	RCE	Information Leak	Heap Overflow, Possible RCE
GreenHills 2019	DoS	DoS	N/A*

**The DNS feature required for exploitation was not supported.*

Differing Approaches, Differing Results: Why it Matters

JSOF's approach to finding and reporting the Ripple20 vulnerabilities was fairly typical: they discovered a vulnerability in a device and ascertained that it was in the Treck stack. They then found other vulnerabilities in that version, and one other version of the stack, and proceeded to publish those vulnerabilities under the assumption that they affected all versions of Treck. As our research shows, that was not the case.

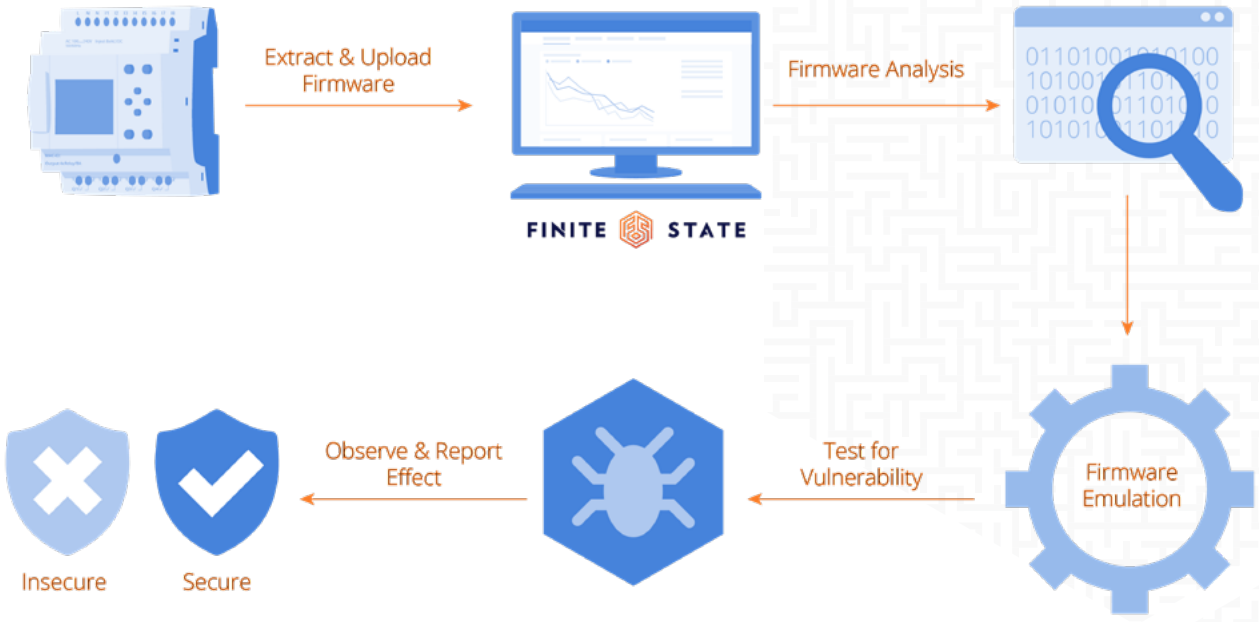
This is significant because publication of the Ripple20 white paper resulted in widespread, ineffective security testing and patching across all of these devices, despite the fact that the two remote code execution vulnerabilities had only been demonstrated on two products.

The inconsistencies between our results and the effects that were initially reported by JSOF highlight the need to verify vulnerabilities that are released to the public. The results of verification, in this instance, drastically change the scope and nature of the necessary responses by security teams.

Unfortunately, the approach demonstrated by JSOF is a conventional one. The discrepancies in their report versus the actual effects are merely a symptom of a larger, systemic issue with the way we analyze and report vulnerabilities in connected devices. As researchers and cybersecurity experts continue to uncover wide spread vulnerabilities like those in the Ripple20 series, it is essential that we are able to quickly and accurately verify these issues across multiple versions of affected devices.

Our approach, detailed in the section below, provides one method for scalable vulnerability verification.

Method: Focused Emulation



Ripple20 has reemphasized the need for automated vulnerability detection and verification solutions that are accurate, reliable, scalable, and fast. Such tools and methods must also be able to analyze binaries without source code which is usually unavailable when evaluating embedded firmware. Static binary analysis has several limitations where it is manually intensive, time consuming, hard to scale, and can lead to false positives. Techniques like signature-based detection (e.g., YARA), instruction matching, or function hashing may result in false negatives and would be ineffective at detecting vulnerable Treck functions because their composition can drastically vary between different platforms, architectures, and tool chains.

To overcome these major obstacles, our research team used their extensive years of collective expertise in reversing, emulation, and vulnerability research to develop a novel capability, which we call Focused Emulation, a comprehensive automated solution capable of detecting and verifying Ripple20 vulnerabilities in target firmware. It allows us to emulate the firmware, execute its binaries and services, exercise the necessary code paths, and test for the actual vulnerability. To ensure successful emulation, the firmware must first undergo automated preprocessing which includes symbol enrichment, function identification, service identification, and initialization.

Using knowledge gained from reversing various firmware with the Treck stack, we identified a way to fully initialize the stack using Focused Emulation for all the potentially vulnerable firmware of interest. Then we supply the malicious packet to the stack for processing and observe the execution state of the firmware to determine whether the stack was vulnerable and the resulting effect. Focused Emulation operates at scale, speed, and efficacy, guaranteeing a high level of confidence with very few to no false positives. It is also fully integrated into our online cloud platform for use today.

Results

Demystifying CVE-2020-11896

Finite State has 20 firmwares with the Treck TCP/IP stack from 5 vendors with multiple versions for most devices. During our reversing efforts, we noticed differences between the vulnerable code present in our firmware and the Digi Connect 9210 firmware from the JSOF whitepaper. Further adding to the confusion was that many vendors were stating the devices for these firmware were vulnerable to Ripple20.

Obviously to get to the bottom of this mystery we needed to exercise the vulnerabilities directly on the firmware to confirm their effect. Armed with the knowledge gained while reversing the stack, we set out to emulate those 20 firmware. We finished the emulation effort for those firmware after rigorous testing and crafted the malicious packet for CVE-2020-11896 with aid from the JSOF whitepaper. We were only rewarded, however, with more questions. When throwing CVE-2020-11896 at those 20 firmwares, we observed that a special check was guarding the truncation code (see Guard Code Figures 2, 3 below), preventing remote code execution. The presence of this guard code is dependent upon whether or not an error checking macro is defined.

Figure 2.
Quadros 2014 based firmware vulnerable to DoS variant of CVE-2020-11896 (Guard Code in green highlight)

```
111   if (ip_total_len_local < pkt_chain_len_local) {
112     if (pkt_chain_len_local == (ts_packet_local->usrPacketPtr).pktuLinkDataLength) {
113       (ts_packet_local->usrPacketPtr).pktuLinkDataLength = ip_total_len_local;
114       (ts_packet_local->usrPacketPtr).pktuChainDataLength = ip_total_len_local;
115       goto LAB_404ae7f4;
116     }
117     fatalLogger(s_tfIpIncomingPacket_404af300,s_Incoming_scattered_data_404af33c);
118     tfPacketTailTrim(ts_packet_local,ip_total_len_local,0);
119     ip_total_len = *(ushort *)((int)device_entry + 0x476);
120   }
121   else {
122 LAB_404ae7f4:
123     ip_total_len = *(ushort *)((int)device_entry + 0x476);
124   }
```

Figure 3.
Net+OS 2020 on Digi Connect ME 9210 with Guard Code in tfIpIncomingPacket which mitigates the CVE effect

```
119             if (ip_total_len_local != pkt_chain_len_local) {
120               if (pkt_chain_len_local == (ts_pkt_offset->pktUserStruct).pktuLinkDataLength) {
121                 (ts_pkt_offset->pktUserStruct).pktuChainDataLength = ip_total_len_local;
122                 (ts_pkt_offset->pktUserStruct).pktuLinkDataLength = ip_total_len_local;
123               }
124               else {
125                 tfPacketTailTrim(ts_pkt_offset,ip_total_len_local,'\0');
126               }
127             }
```

Results

However, even though the RCE was prevented, the alternate branch on that check for some firmware called the fatalLogger function (the actual name may vary) which commonly executed an infinite loop that halted packet processing and effectively led to a DoS condition (see Figure 4 below).

Figure 4. fatalLogger function causing a DoS for CVE-2020-11896

```
0x43b88:        addw    r0, pc, #0x680
0x43b8c:        bl      #0x32c1c
0x32c1c:        push    {r7, lr}
0x32c1e:        mov     r2, r1
0x32c20:        mov     r1, r0
0x32c22:        addw    r0, pc, #0xc4
0x32c26:        bl      #0x1336
0x32c2a:        b       #0x32c2a
Found Halt!
fatal - b'tfIpIncomingPacket': b'Incoming scattered data'
```

Since we suspected that none of our firmware contained the RCE variant of the vulnerable code, we decided to look at a Digi Connect ME 9210 which was detailed in the JSOF whitepaper. Indeed, we found that the suspected vulnerable code (see Figure 5 below) existed in the Digi firmware and then proceeded to emulate it. Finally, we landed CVE-2020-11896 as an RCE on the emulated firmware. Afterwards we re-confirmed that the same malicious packets only caused a DoS, or effects other than RCE, for the other firmware that contained the guard code with the oldest firmware release dating back to 2014.

Figure 5. Unpatched Digi Firmware Vulnerable to RCE variant of CVE-2020-11896 (no Guard Code)

```
103             chain_len = (ts_pkt_local->pktUserStruct).pktuChainDataLength;
104             if (chain_len < ip_len) goto LAB_00069c24;
105             if (ip_len != chain_len) {
106               (ts_pkt_local->pktUserStruct).pktuChainDataLength = ip_len;
107               (ts_pkt_local->pktUserStruct).pktuLinkDataLength = ip_len;
108             }
```

Emulation was so effective at detecting CVE-2020-11896 and its effect that we decided to integrate it into our online platform under our Focused Emulation capability. Since then, we have also added CVE-2020-11898, CVE-2020-11901, and are developing other CVEs as prioritized by customer needs.

Results

Examining the Impact of CVE-2020-11901

CVE-2020-11901 is perhaps the most interesting set of vulnerabilities in Ripple20. JSOF certainly thought so, making it the subject of their BlackHat and DefCon presentations. We were surprised that their reporting didn't mention the testing of both CVEs on the same device, so we decided to investigate this CVE on the Digi Connect ME 9210.

DNS must be enabled in the Treck stack for the device to be affected by CVE-2020-11901 vulnerabilities. Using JSOF's CVE-2020-11901 whitepaper as a guide we set out to test the three vulnerabilities against the Digi and our other firmware. Preliminary reversing of the firmware revealed that none of our firmware had the code for the "bad RDLENGTH" vulnerability, but they did have code for the "Read Out-of-Bounds" and "Integer Overflow" vulnerabilities. This inspired our team to perform a deeper evaluation of CVE-2020-11901 on our firmware.

Examining the DNS response handler, tfDnsCallback, we saw that the vulnerable pair of functions (tfDnsExpLabelLength and tfDnsLabelToAscii) are only called when the DNS record type requested was either MX (0xf) as shown in Figure 6 or PTR (0xc) as shown in Figure 7. Using the JSOF whitepaper as a guide, we crafted the malicious packets and observed their propagation through tfDnsCallback. Using Focused Emulation, we were able to visually observe the code paths taken by the packets which drastically reduced the exploit debugging effort. In our Focused Emulation development roadmap, we plan to use code path exploration to automatically modify the malicious packet for variants of the Treck stack. For example, code path exploration can be leveraged to automatically exploit stacks which randomize the Transaction ID.

Figure 6. MX record handling in the Treck stack

```
186       if (record_type == 0xf) {
187         addr_info = tfDnsAllocAddrInfo();
188         if (addr_info != 0x0) {
189           memcpy(&addr_info->ai_addrlen,resourceRecordAfterNamePtr + 10,2);
190           label_length = tfDnsExpLabelLength(resourceRecordAfterNamePtr + 0xc,dnsHeaderPtr);
191           addr_info->ai_canonname = 0x0;
192           if (label_length != 0) {
193             asciiPtr = tfGetRawBuffer(label_length);
194             addr_info->ai_canonname = asciiPtr;
195             if (asciiPtr != 0x0) {
196               tfDnsLabelToAscii(resourceRecordAfterNamePtr + 0xc,asciiPtr,dnsHeaderPtr);
```

Figure 7. PTR record handling in the Treck stack

```
233       if ((flags == 0xc) && (record_type == 0xc)) {
234         label_length = tfDnsExpLabelLength(resourceRecordAfterNamePtr + 10,dnsHeaderPtr);
235         cacheEntry->dnscRevHostnameStr = 0x0;
236         if (label_length == 0) goto LAB_000f4684;
237         asciiPtr = tfGetRawBuffer(label_length);
238         *&cacheEntry->dnscRevHostnameStr = asciiPtr;
239         if (asciiPtr == 0x0) goto LAB_000f4684;
240         tfDnsLabelToAscii(resourceRecordAfterNamePtr + 10,asciiPtr,dnsHeaderPtr);
```

Results

Analysis revealed that both the Digi and other firmware were vulnerable to both the "Read Out-of-Bounds" and "Integer Overflow" vulnerabilities. The read out-of-bounds leads to an information leak when the answer label is not null terminated in the DNS response. This was straightforward to implement and easily verified within our Focused Emulation solution. However, unlike the read out-of-bounds, the integer overflow did not immediately yield a heap overflow as suggested by JSOF's whitepaper. As a result, we had to craft a new exploit as detailed in our follow-up article.[3] We then evaluated the new exploit using Focused Emulation and verified that the heap overflow can be achieved on our firmware, providing the opportunity for possible RCE.

From the 20 firmware we emulated, we found only two had the functionality required for exploitation of CVE-2020-11901. This is quite surprising because vendors had announced their devices as being vulnerable to CVE-2020-11901 when in reality their device firmware was not affected since it wasn't configured to support DNS. During our investigation, we also noticed that Aruba Networks listed each CVE and the specific effect on their devices.[4] Their PSA confirmed that none of the Ripple20 CVEs had an RCE effect on their devices. All of this suggests more analysis should be performed to verify the effect of Ripple20 CVEs as device configurations may dramatically reduce the actual security impact compared to what was originally reported in the vulnerability disclosure.

---

[3] https://finitestate.io/2020/10/12/the-aftershock-of-ripple20/

[4] https://www.arubanetworks.com/assets/alert/ARUBA-PSA-2020-006.txt

Conclusion

It is important to hold everyone in the cybersecurity community accountable. That's one of the things that this report is attempting to do.

Verifying the true impact of vulnerabilities like those found in Ripple20, while difficult, is not impossible. That the approach taken by JSOF was not atypical indicates that our vulnerability reporting and scoring system itself is flawed. Again, the vulnerabilities that JSOF reported are very real and high impact for the devices that they tested against; however, these vulnerabilities being correlated to a "version" of the Treck stack and reported via NVD, ICS CERT, etc., led to incorrect assumptions which fueled a misguided community response. The impact and presence of the vulnerable condition was not tested on the devices which were deemed to be affected.

As some vendors independently from Treck attempted to develop patches for all of the devices that were presumed to be affected by this highly publicized vulnerability, we saw two problems: devices that were unaffected were patched unnecessarily, and due to these vendors not adequately verifying their patches, new vulnerabilities were introduced in the process (which our team has identified, and which we will detail further after the responsible disclosure process is complete). These unintended consequences are completely avoidable if security teams are able to verify these vulnerabilities before attempting to patch them. Most product security teams, however, lack the proper tooling to be able to verify the effects of these vulnerabilities and patches, which is why it's crucial that we have a system in place that can do so quickly and accurately to prevent this kind of response.

Such tooling would also enable product security teams to properly evaluate CVSS scores and their true severity. CVSS uses scope, exploitability metrics, and impact metrics to calculate a base score. This is primarily within the context of the vulnerable and impacted components of the device. Organizations are encouraged to supplement the Base score with additional information or metrics specific to their use of the vulnerable product to produce a severity score more accurate for their organizational environment. This, however, is outside the scope of CVSS.

As a community we are falling short on vulnerability management for connected devices, and we have to find more scalable ways to verify and respond to vulnerabilities in those devices. We cannot just rely on CVSS, vendor reports, or even security researchers. Users of these devices need to have better risk analysis tools in place in order to make better informed risk decisions. Device manufacturers must have the ability to test their device firmware and security patches at scale before releasing them to their customers.

It is essential that we also create a better system or metric to measure the severity and magnitude of IoT and connected device vulnerabilities and their impact in the real-world. This system must require greater precision and accuracy in writing CVEs. The two RCE vulnerabilities in the Ripple20 disclosure created false alarms in large part because of how vague and broad they were even though they were each only demonstrated against one representative device.

There is an obvious challenge here. While the specificity of CVEs should help to inform and educate vendors, there must still be a balance between revealing too much and allowing attackers to leverage that information.

Still, we as a community need to do our due diligence—on what's being reported, how it's reported, and what's being done about it—much better than we are at present. This would allow us to better prevent, manage, and mitigate future ripple effects in security and the supply chain.