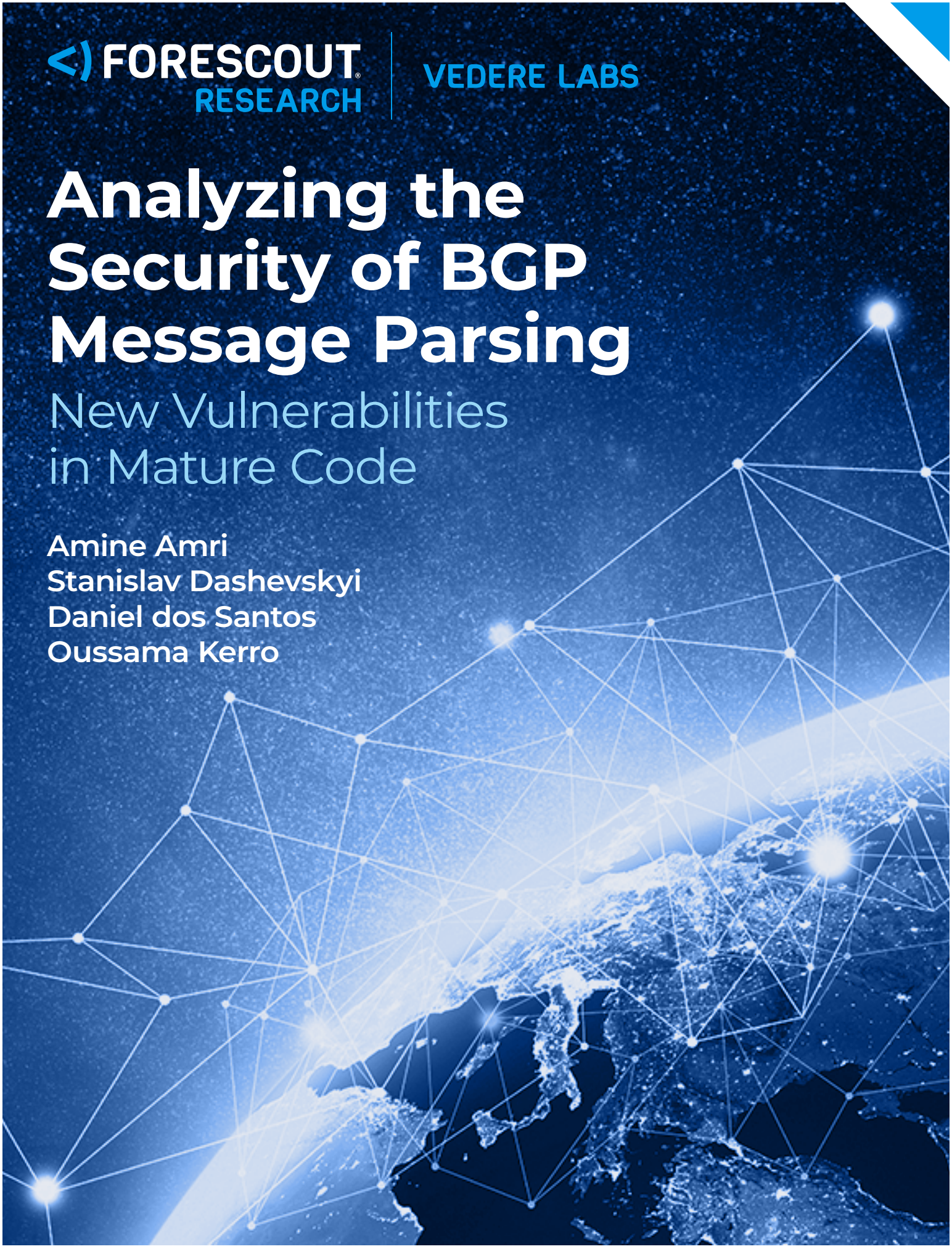
FORESCOUT RESEARCH | VEDERE LABS

Analyzing the Security of BGP Message Parsing

New Vulnerabilities in Mature Code

Amine Amri
Stanislav Dashevskyi
Daniel dos Santos
Oussama Kerro



Contents

1. Executive summary

This report discusses an often-overlooked aspect of Border Gateway Protocol (BGP) security: vulnerabilities in its software implementations. More specifically, vulnerabilities in BGP message parsing that could be exploited by attackers to achieve a denial of service (DoS) condition on vulnerable BGP peers.

Routing is the process of directing data packets to their intended destinations on a network. BGP is the main routing protocol for the internet, as it allows individual autonomous systems (ASes), which are blocks of IPs leased to an organization for a certain time by a registrar, to exchange routing and reachability information. When BGP fails, an AS may become unreachable because others cannot route their packets there and the unreachable AS becomes cut off from the rest of the Internet. When BGP is abused by threat actors, network traffic may be rerouted through unintended locations.

There are both accidental and intentional disruptions of routing on the internet, since BGP was not initially designed with security in mind. Original BGP weaknesses that may lead to major incidents and internet outages have been known for a long time. For example, in a 2018 incident, traffic for Google IP addresses was routed through China Telecom for more than an hour. And in July 2022, the Russian ISP Rostelecom announced routes for parts of Apple's network, resulting in connections to Apple's services potentially being redirected through Russia for more than 12 hours.

Software suites implementing BGP are relied on not only for internet routing but also for functions such as internal routing in most large data centers as well as MPLS L3 VPNs. Following the network function disaggregation (NFD) trend, today many leading implementations are open source.

A lot of (deserved) attention is given to aspects of BGP protocol security discussed in RFC4272, which can be mitigated with the use of RPKI and BGPsec. However, recent BGP incidents show that it might take only a malformed packet to cause a large disruption.

This report presents a quantitative analysis of previously known vulnerabilities in both open and closed-source popular BGP implementations, as well as a new analysis of seven implementations we conducted both manually and automatically using a specially developed fuzzer that we have released to the community. There are two main findings in this research:

- Some implementations process parts of OPEN messages (e.g., decapsulating optional parameters), before validating the BGP ID and ASN fields of the originating router. This means that only TCP spoofing (instead of a complete takeover of a configured peer) is required to inject malformed packets.
- We found three new vulnerabilities in a leading open-source implementation, FRRouting, which could be exploited to achieve denial of service (DoS) on vulnerable BGP peers, thus dropping all BGP sessions and routing tables and rendering the peer unresponsive. These vulnerabilities were found using a fuzzer we developed.

Our research shows that modern BGP implementations still have vulnerabilities that can be abused by attackers.

2. Main findings

2.1. Why analyze BGP implementations?

There has been a lot of research on the (in)security of the BGP protocol itself and, as a direct consequence, many BGP peers now use authentication and encryption mechanisms, such as Resource Public Key Infrastructure (RPKI), although recent works show that BGP is still not secure enough in many real-world networks.

The various projects that implement the BGP protocol have not received the same level of attention in the security community as the protocol itself. Various implementations of these mechanisms may be vulnerable, leaving BGP peers wide open for attacks. It has also been argued that attacking BGP peers might require a lot of resources from the attackers' side: hijacking portions of the internet entails successfully breaching a trusted relationship (e.g., attackers must either compromise an AS or become one). However, the BGP software itself might be vulnerable, and we have seen many such vulnerabilities disclosed over the years. The most recent systematic work we found about security testing of BGP implementations was published 20 years ago.

Therefore, we focused on one of the most overlooked aspects of information security: broken software. **Given the maturity of BGP implementations and the large corpus of patched vulnerabilities in them, are there still any "low-hanging fruit" to be found?**

2.2. Vulnerabilities found

We analyzed seven popular BGP implementations, three open source (FRRouting, BIRD, OpenBGPd) and four closed source (Mikrotik RouterOS, Juniper JunOS, Cisco IOS, Arista EOS), using both manual analysis and fuzzing, as described in Section 3.

We found three new vulnerabilities in the latest release of Free Range Routing (FRRouting) at the time – version 8.4, released on Nov 7, 2022. The vulnerabilities are summarized in Table 1 and detailed in Section 4.

Table 1 – New CVEs

CVE ID	TESTED PRODUCT	DESCRIPTION	CVSSV3.1	POTENTIAL IMPACT
CVE-2022-40302	FRRouting 8.4	Out-of-bounds read when processing a malformed BGP OPEN message with an *Extended Optional Parameters Length* option.	6.5	**DoS**
CVE-2022-40318	FRRouting 8.4	Out-of-bounds read when processing a malformed BGP OPEN message with an *Extended Optional Parameters Length* option. This is a different issue from CVE-2022-40302.	6.5	**DoS**
CVE-2022-43681	FRRouting 8.4	Out-of-bounds read when processing a malformed BGP OPEN message that abruptly ends with the option length octet (or the option length word, in case of OPEN with extended option lengths message).	6.5	**DoS**

The issues were reported to the FRRouting team and fixed in the following versions:

- CVE-2022-40302 and CVE-2022-40318: https://github.com/FRRouting/frr/pull/12043
- CVE-2022-43681: https://github.com/FRRouting/frr/pull/12247

FRRouting is a software suite that implements BGP and many other routing protocols, such as Open Shortest Path First (OSPF). It was forked from another open source project called Quagga in 2016 by developers from several commercial organizations. FRRouting is a prime example of NFD, which is the move away from network appliances that implement all networking functions as closed applications and toward open routing solutions shared by many vendors. FRRouting is currently used in the networking solutions of several major vendors, including nVidia Cumulus, which in turn is adopted by large organizations such as PayPal, Yahoo, Qualcomm and the Dutch National Police; DENT, which is mainly supported by Amazon; and SONiC, which is mainly supported by Microsoft and used in some Juniper routers.

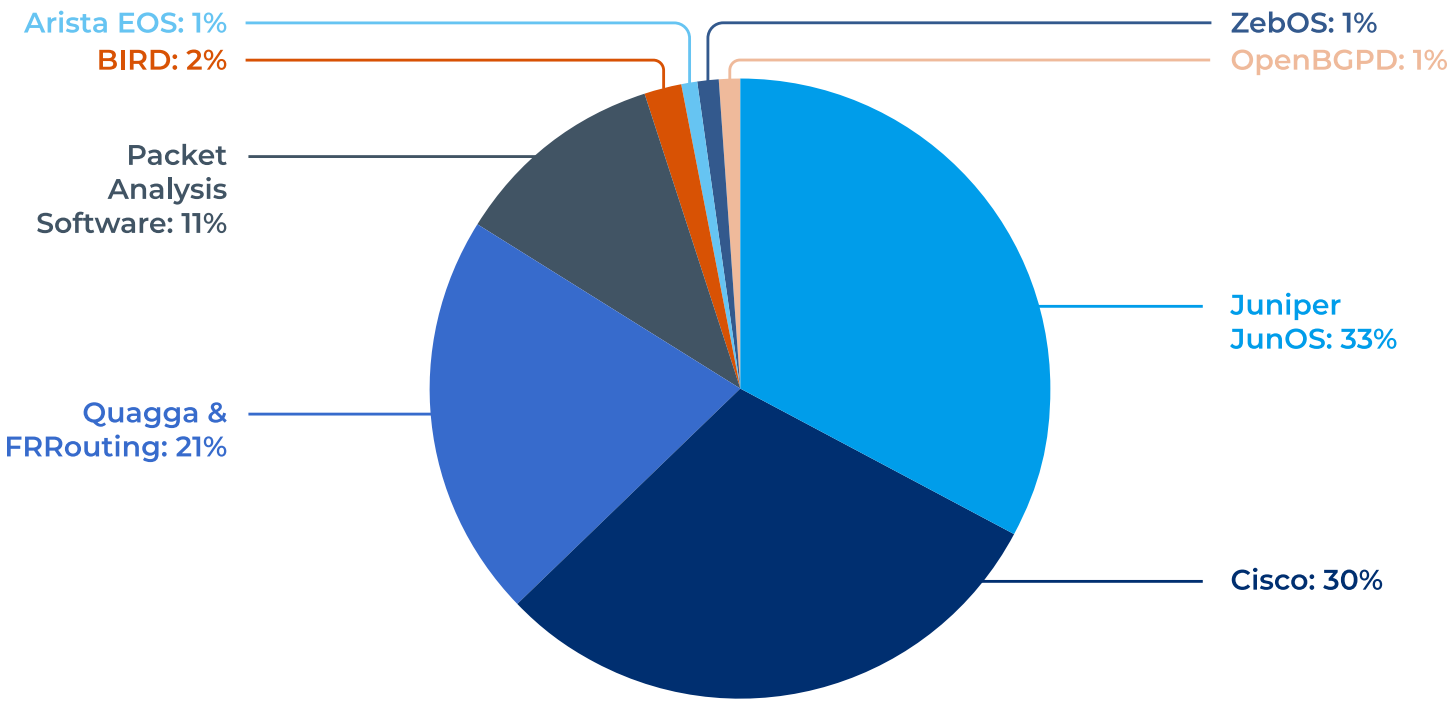
2.3. Impact of the vulnerabilities

Attackers may leverage any of the three vulnerabilities we discovered to achieve a DoS on a vulnerable BGP peer, thus dropping all BGP sessions and routing tables and rendering the peer unresponsive for several seconds (the BGP daemon will automatically restart after a timeout). The DoS condition may be prolonged indefinitely by repeatedly sending malformed packets.
Two of these issues (CVE-2022-40302 and CVE-2022-43681) can be triggered before FRRouting validates BGP Identifier and ASN fields. While FRRouting only allows connections between configured peers by default (e.g., OPEN messages from hosts not present in the config files will not be accepted), in this case attackers only need to spoof a valid IP address of a trusted peer. Another possibility for the attacker is to take advantage of misconfigurations or attempt to compromise a legitimate peer by exploiting other vulnerabilities. Similar DoS vulnerabilities in FRRouting have already caused notable disruptions.

There are over 330,000 hosts with BGP enabled on the internet, and close to 1,000 of those reply to unsolicited BGP OPEN messages. Most of the BGP hosts are in China (close to 100,000), the U.S. (50,000) and the UK (16,000). We also see more than 200,000 hosts running Quagga and more than 1,000 running FRRouting (not all of them with BGP enabled). Again, China comes in on top with more than 170,000 hosts, followed by the U.S. with 15,000 and Japan with close to 4,000.

3. Methodology

We first reviewed historical vulnerabilities in various BGP implementations, specifically related to BGP message parsing. We deliberately excluded other kinds of issues such as logical flaws, access control and password authentication. We searched the NVD and counted 123 such vulnerabilities: the figure below shows their distribution by project/software.



Note that the "packet analysis software" category combines the tcpdump and Wireshark projects (one of the vulnerabilities dates back to when Wireshark was known as Ethereal). It is not surprising to see that network protocol analyzers may be implemented in the same (flawed) way as the actual software that implements the actual protocols. The figure also combines Quagga Routing Suite and FRRouting projects, since they have a shared codebase.

We noticed that nine of these vulnerabilities had remote code execution (RCE) potential, 11 of them could be used by attackers to leak sensitive information and all of them could be used to trigger a variety of DoS conditions via unhandled assertions, memory crashes or infinite loops. Most of the RCE vulnerabilities were identified within the Quagga project, and the RCE impact is specified as potential. (We could not find any evidence that the RCEs are indeed exploitable). However, DoS vulnerabilities may also be very valuable for attackers that aim to disrupt BGP routers and cause major networking outages.

We selected several implementations based on their popularity and the availability of analysis artifacts; that is, open- and closed-source implementations with demo versions that could be installed on a virtual machine. We list these implementations below:

IMPLEMENTATION	OPEN SOURCE?	REMARKS
FRRouting	Yes	Free and open source routing protocol suite for Linux and Unix platforms. FRRouting is a fork of the Quagga Routing Suite, and it implements multiple routing protocols. We analyzed only the functionality related to BGP, both manually and by fuzzing.
BIRD	Yes	A dynamic IP routing daemon primarily targeted to Linux, FreeBSD and other Unix-like systems. We analyzed only the functionality related to BGP, both manually and by fuzzing.
OpenBGPD	Yes	A free implementation of BGP, shipped as a part of OpenBSD Unix-like operating system. We performed both manual code analysis and fuzzing.
Microtik RouterOS v6.x and v7.x	No	An operating system for Microtik routers that includes a component implementing the BGP protocol. We performed fuzzing as well as a lightweight reverse engineering and manual analysis of the BGP-related binary for the latest version 6.x and 7.x.
Juniper JunOS	No	A FreeBSD-based network operating system used in Juniper Networks devices (routers, switches). We performed fuzzing of a BGP component included in the latest version of JunOS.
Cisco IOS	No	A family of proprietary network operating systems used on several network switches and routers manufactured by Cisco Systems. We performed fuzzing of the BGP-related functionality.
Arista EOS	No	A Linux-based network operating system that is the core of Arista cloud networking solutions. We performed fuzzing of the BGP-related functionality.

Some of the selected implementations have many historical vulnerabilities (FRRouting, JunOS, Cisco) while others have very few of them (BIRD, OpenBGPD, Arista, Microtik). We decided that it would also be interesting to see whether Linus's law applies in this case. In other words:

- Can we find some new vulnerabilities in projects with rich vulnerability histories where it is believed that developers have fixed all of them and there is nothing to look for?
- Would low vulnerability counts indicate the maturity of the codebase or might it indicate a lack of security audits?

Based on our analysis of prior vulnerabilities, we devised the following checklist for manual analysis of:

- Relevant network packet structures, memory allocation/deallocation and error handling (assertions, watchdogs, etc.).
- Functions that parse incoming OPEN, UPDATE, NOTIFICATION and ROUTE REFRESH messages to identify erroneous bounds checks that may lead to crashes, infinite loops or assertion failures.
- Code that handles various length fields (BGP header length, optional parameters, capabilities, path attributes, route updates/withdrawals, etc.)
- The BGP state machine (e.g., whether a peer would process unsolicited or out-of-order OPEN, UPDATE and other messages).

We also observed how these implementations react to attempts to establish a BGP session with non-configured peers. As was noted in previous research, this behavior is not uniform across all implementations, although four of the seven implementations analyzed behaved the same.

IMPLEMENTATION	BEHAVIOR WHEN A NON-CONFIGURED PEER TRIES TO ESTABLISH A BGP SESSION
FRRouting	Proceeds with a TCP handshake, terminates the TCP session (a TCP Reset packet) after an OPEN packet is received. Performs some processing of OPEN messages before validating the BGP ID and ASN fields.
BIRD	Proceeds with a TCP handshake, terminates the TCP session (a TCP Reset packet) after an OPEN packet is received.
OpenBGPD	Proceeds with a TCP handshake, terminates the TCP session (a TCP Reset packet) after an OPEN packet is received.
Microtik RouterOS	Proceeds with a TCP handshake, terminates the TCP session (a TCP Reset packet) after an OPEN packet is received.
Juniper JunOS	Proceeds with a TCP handshake. Sends back an OPEN message, sends back a Cease NOTIFICATION message with the subcode 5 (Connection Rejected).
Cisco IOS	Does not allow to establish a TCP connection (TCP handshake fails).
Arista EOS	Proceeds with a TCP handshake, terminates the TCP session (a TCP Reset packet) after an OPEN packet is received.

Six out of seven implementations proceed with the TCP handshake before checking whether an incoming OPEN message is sent from a pre-configured peer. This is because the BGP daemon process is running in user mode (i.e., connection filtering not happening on the kernel level), unlike with Cisco IOS, where it seems that the filtering happens on the kernel level. We also found that FRRouting begins to process OPEN messages (e.g., decapsulating optional parameters) before it gets a chance to verify the BGP Identifier and ASN fields of the originating router. At a glance, the most "permissive" implementation is Juniper JunOS, as it replies with BGP messages identifying itself.

No implementation allowed us to establish a BGP session from a non-configured peer – **this means that TCP spoofing (or a complete takeover of a configured peer) is required to inject malformed packets**. Note that we tested the behavior inherent to the default configurations, so custom (mis-)configurations might allow the attackers to peer with BGP instances without any restrictions.

While performing manual analysis, we did not focus on subtle performance issues such as excessive CPU consumption and memory leaks, as it would be extremely difficult to identify them manually. As we learned with Project Memoria, even though implementations may vary widely with respect to resource management, their packet parsing functions can still be quite similar and exhibit similar bugs

Finally, we realized that BGP is a complex protocol and it may prove very difficult to fully analyze the parsers even in open-source implementations (let alone to reverse-engineer and analyze the closed-source ones). Since we could not find an open-source off-the-shelf BGP fuzzer, we implemented a lightweight one based on the BooFuzz framework. For each of the BGP messages that may be processed by a peer (OPEN, UPDATE, NOTIFICATION, ROUTE REFRESH), we created a stateful fuzzer that will: (1) establish a BGP session with a peer (except for OPEN messages that initiate these sessions); (2) send a malformed BGP message with a specific payload (we had a number of test cases based on the manual analysis checklist and past vulnerabilities, as well as test cases with random fuzzloads); and (3) test the target for crashes via a custom RPC monitor (we used the corresponding base class of BooFuzz).

4. Technical details of new findings

We only performed manual analysis on the implementations with open-source code bases (FRRouting, BIRD and OpenBGPD), we also looked at the relevant binary artefacts from the two major versions of Microtik's RouterOS (versions 6 and 7).

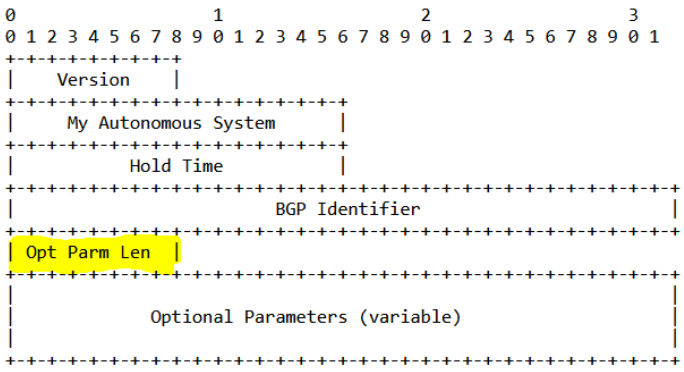
After completing the manual analysis phase with no results, we ran our fuzzer against each of the selected implementations, with a time limit of five hours per test case. Almost immediately, we could identify two closely related issues in FRRouting. We could also identify a third similar issue by doing some manual analysis of a code fragment reused within the same project's codebase. These issues are as follows:

CVE ID	DESCRIPTION
CVE-2022-40302	Out-of-bounds read when processing a malformed BGP OPEN message with an *Extended Optional Parameters Length* option.
CVE-2022-40318	Out-of-bounds read when processing a malformed BGP OPEN message with an *Extended Optional Parameters Length* option. This is a different issue from CVE-2022-40302.
CVE-2022-43681	Out-of-bounds read when processing a malformed BGP OPEN message that abruptly ends with the option length octet (or the option length word, in case of OPEN with extended option lengths message).

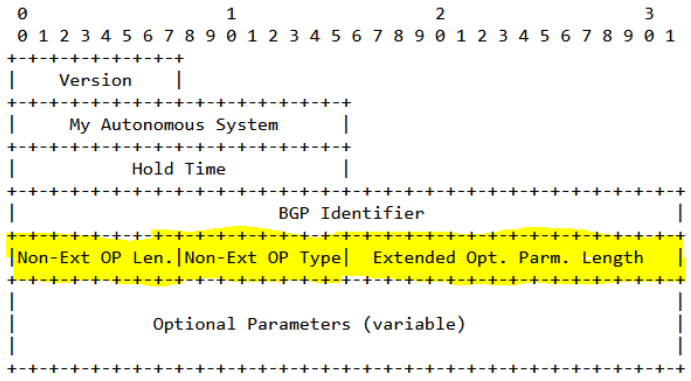
These vulnerabilities are related to the Optional Parameters Length field of the BGP OPEN messages. The main root cause is the same vulnerable code pattern copied into several functions related to different stages of parsing OPEN messages. All three issues, when triggered, allow to achieve a DoS condition via three different code paths. Two out of three issues (CVE-2022-40302 and CVE-2022-43681) can be triggered before FRRouting validates BGP Identifier and ASN fields, making it easier for potential attackers to spoof a configured originating router – in this case they only need to spoof the originating IP address.

We disclosed our findings to the developers of the FRRouting project and they fixed all the issues shortly thereafter.

The original RFC 4271 that details the latest version of the BGP protocol mentions the Optional Parameters Length field that sets the length of optional parameters in OPEN messages (the length of this field is fixed at 1 octet). If an OPEN message does not contain any optional parameters, the value of this field is set to 0. The same RFC document mentions that each optional parameter has a dedicated length field as well, which size is fixed at 1 octet.



RFC 9072 introduced extended option lengths. If the Optional Parameters Length field is set to a special value of 0xff, the length fields of optional parameters will be extended from one octet to a word (two octets). In case of extended option lengths, the rest of the OPEN message will look as follows: following the Optional Parameter Length field set to 0xff, there will be the Non-extended Optional Parameter Type field (one octet) that must be also set to 0xff; next, there will be the word-wide Extended Optional Parameters Length field, which must contain the total length of extended optional parameters, followed by the actual optional parameters. Each of the optional parameters is expected to have a word-wide length field as well.



FRRouting (unlike some implementations such as both versions of the BGP component from Microtik RouterOS that we analyzed) supports RFC 9072 and processes OPEN messages with both regular and extended lengths for optional parameters. From previous research, we know that because it's tricky to implement, the functionality that processes variable-length fields of network packets is a common source of bugs and security vulnerabilities. Below, we present some technical details about our findings.

4.1. CVE-2022-43681

The root cause of this issue is insufficient bounds checks of extended option length octets in OPEN messages. The following code snippet illustrates the vulnerability:

```
    static int bgp_open_receive(struct peer *peer, bgp_size_t size)
    {
      // [...]

1:    optlen = stream_getc(peer->curr);
2:
3:    /* Extended Optional Parameters Length for BGP OPEN Message */
4:    if (optlen == BGP_OPEN_NON_EXT_OPT_LEN
5:        || CHECK_FLAG(peer->flags, PEER_FLAG_EXTENDED_OPT_PARAMS)) {
6:        uint8_t opttype;
7:
8:        opttype = stream_getc(peer->curr);
9:        if (opttype == BGP_OPEN_NON_EXT_OPT_TYPE_EXTENDED_LENGTH) {
9:            optlen = stream_getw(peer->curr);
10:           SET_FLAG(peer->sflags,
11:                PEER_STATUS_EXT_OPT_PARAMS_LENGTH);
          }
      }
      // [...]
  }
```

At line 1, the *bgp_open_receive()* function will read the option length octet (the *optlen* variable). If *optlen* equals to ***0xff*** (defined as *BGP_OPEN_NON_EXT_OPT_LEN* constant), the function will proceed reading the next octet (*opttype*) by calling the *stream_getc()* function at line 8 – this function reads the next octet value from a raw packet stream. If *opttype* equals to ***0xff*** as well, it is considered that the OPEN packet contains extended optional parameters (as per RFC 9072). In this case, a next word will be read from the raw packet stream (*stream_getw()*, line 9). This word corresponds to the *Extended Optional Parameters Length* field. After these fields are read, the code will proceed with parsing optional parameters.

There is an issue in this code that leads to out-of-bound reads: when a malformed BGP OPEN message that ends with just the extended option length field (***0xff***) is received, the call to *stream_getc()* on line 8 will read one octet beyond raw packet stream. This out-of-bounds read will trigger an assertion that will throw a SIGABRT signal, causing the *bgpd* daemon of FRRouting to be restarted, resulting in a DoS condition.

The same effect can be achieved by sending a malformed BGP OPEN packet that ends with the two octets *Non-Extended Option Length* and *Non-Extended Option Type* (both should be set to ***0xff***). In this case, the code will read a word (two octets) out of bounds when calling the *stream_getw()* function at line 9.

4.2. CVE-2022-40302

The root cause of this issue is improper bounds check for the OPEN message packet when reading the AS4 capability (4-byte AS Numbers, as per RFC 6793). The offending function is called *peek_for_as4_capability()*, which is called before processing all other options. Its purpose is to iterate over all present options and to find and parse the AS4 capability:

```
as_t peek_for_as4_capability(struct peer *peer, uint16_t length)
{
 1:  struct stream *s = BGP_INPUT(peer);
 2:  size_t orig_getp = stream_get_getp(s);
 3:  size_t end = orig_getp + length;
 4:  as_t as4 = 0;

 5:  if (BGP_DEBUG(as4, AS4))
 6:    zlog_debug(
 7:      "%s [AS4] rcv OPEN w/ OPTION parameter len: %u, peeking for as4",
 8:      peer->host, length);
     /* the error cases we DONT handle, we ONLY try to read as4 out of
      * correctly formatted options.
      */
 9:  while (stream_get_getp(s) < end) {
10:    uint8_t opt_type;
11:    uint16_t opt_length;

       /* Check the length. */
12:    if (stream_get_getp(s) + 2 > end)
13:      goto end;

       /* Fetch option type and length. */
14:    opt_type = stream_getc(s);
15:    opt_length = BGP_OPEN_EXT_OPT_PARAMS_CAPABLE(peer)
16:               ? stream_getw(s)
17:               : stream_getc(s);

18:    if (opt_type == BGP_OPEN_OPT_CAP) {
         //[...]
         // Look for the AS4 capability and parse  it
       }
     }
}
```

The bounds check condition at line 12 checks for two bytes in advance, against the option length received (the variable *end* is initialized at line 2). However, if the received OPEN message has optional parameters with extended length (as per RFC 9072), the function would read three bytes from the raw packet stream instead of just two, as anticipated by the bounds check condition (i.e., line 16 instead of line 17 will be executed).

We could then craft such malformed OPEN packet that passes the check at line 12, and achieves an out-of-bounds read at line 16. One additional constraint here is that the optional parameters should not contain the *Capabilities* option of value **0x02** (line 18) – in this way we ensure that the code iterates in the "while" loop triggering the out-of-bounds read on one of its iterations.

4.3. CVE-2022-40318

This vulnerability is similar to CVE-2022-40302; however, it was a bit more difficult to find a payload that will trigger it. The challenge here was to construct a malformed OPEN message that would successfully go through the *peek_for_as4_capability()* function without triggering an issue there, but that would trigger a very similar issue in the *bgp_open_option_parse()* function that will be called after.

We spotted this issue after looking at the usages of the *stream_getw()* function and finding a very similar code pattern to the one in *peek_for_as4_capability()*. The vulnerable code fragment of the *bgp_open_option_parse()* function is shown below:

```
as_t peek_for_as4_capability(struct peer *peer, uint16_t length)
{
 1:  struct stream *s = BGP_INPUT(peer);
 2:  size_t orig_getp = stream_get_getp(s);
 3:  size_t end = orig_getp + length;
 4:  as_t as4 = 0;

 5:  if (BGP_DEBUG(as4, AS4))
 6:    zlog_debug(
 7:      "%s [AS4] rcv OPEN w/ OPTION parameter len: %u, peeking for as4",
 8:      peer->host, length);
    /* the error cases we DONT handle, we ONLY try to read as4 out of
     * correctly formatted options.
     */
 9:  while (stream_get_getp(s) < end) {
10:    uint8_t opt_type;
11:    uint16_t opt_length;

       /* Check the length. */
12:    if (stream_get_getp(s) + 2 > end)
13:      goto end;

       /* Fetch option type and length. */
14:    opt_type = stream_getc(s);
15:    opt_length = BGP_OPEN_EXT_OPT_PARAMS_CAPABLE(peer)
16:               ? stream_getw(s)
17:               : stream_getc(s);

18:    if (opt_type == BGP_OPEN_OPT_CAP) {
         //[...]
         // Look for the AS4 capability and parse  it
       }
     }
}
```

If you look closely at the code, you can spot a very similar pattern to the vulnerable code of CVE-2022-40302: there is a bounds check (line 6) that accounts for only two octets that should be present in the packet with the regular option length. However, the bounds check fails to account for extended option lengths, for which there must be three octets in total. The issue will be triggered in the same way at line 16 while calling the *stream_getw() function.*

5. Mitigation recommendations

Since BGP is such an integral part of the internet, there are several guidelines on how to secure it, such as those from the Internet Society, RIPE NCC, NIST and the NSA. However, those guidelines tend to focus only on the known issues with BGP insecurity and how to deploy RPKI.

It is important to note that BGP is found in unexpected places beyond ISPs and internet exchanges (IXs). For instance, BGP is commonly used internally to route the traffic in large data centers and BGP extensions, such as MP-BGP, are widely deployed for MPLS L3 VPNs. Therefore, organizations should not rely only on their ISPs to handle BGP security.

Also, because of the supply chain effect we have seen in past research, vulnerabilities on open-source components tend to spread widely. The new issues CVE-2022-40302 and CVE-2022-40318, for instance, clearly show how the same vulnerable code may be present in multiple places of a code base and serve as a root cause for several vulnerabilities. Similar (or the same) code could be present in other projects and affect several products using FRRouting or one of the networking operating systems that rely on it, such as Cumulus, SONiC and DENT, mentioned above.

To mitigate the risk of vulnerable BGP implementations, such as the FRRouting issues we found, the best recommendation is to **patch network infrastructure devices as often as possible**. To do so, must first have an updated asset inventory that keeps track of all the networking devices in your organization and the versions of software running on them. This is much easier to achieve with software that provides granular visibility for every device in the network.

6. Conclusion

After reviewing and testing the selected implementations, we can assume that they are quite robust and have good protective measures against malformed packets. This is not particularly surprising, considering that these are mature and actively developed projects/products with many contributors. Nevertheless, we were surprised by our findings in the FRRouting project: while these are not major vulnerabilities, it is interesting to see evidence that BGP message parsing issues can still be found in major projects with a good history of security patches. The fact that FRRouting provides wide support for fuzzing its own code suggests that Linus's law may not always hold and a few "shallow" bugs may still slip through the cracks. On a positive note, the issues were fixed by the FRRouting team very fast, which again shows the project has a high level of security maturity.

While all the BGP implementations we tested are quite resilient against malformed packets, it may be possible that we overlooked some bugs. As noted by Cavedon et al., "routers waiting and replying to OPEN messages might be vulnerable to some kind of attack." Less mature implementations may still contain BGP message parsing issues.

It would also be interesting to analyze logical flaws within the protocol itself, such as BGP state machine issues that lead to undefined/erroneous behavior. RFC 4272 has an extensive list of logical flaws that may be present in various implementations. There is also a lot more work required to develop a scalable BGP fuzzer that considers not only malformed messages but also the state machine of the BGP protocol.

A final worthwhile endeavor would be an audit for bugs within peer authentication mechanisms built on top of the BGP protocol, such as the following flaws in RPKI and MD5 authentication: CVE-2020-12831, CVE-2021-0281, CVE-2022-20694 and CVE-2020-3165. These issues seem to have surfaced only recently, which may indicate that a wider set of implementations might suffer from similar flaws.

FORESCOUT

Forescout Technologies, Inc.
Toll-Free (US) 1-866-377-8771
Tel (Intl) +1-408-213-3191
Support +1-708-237-6591
Learn more at Forescout.com