binarly

LogoFAIL

Six months after disclosure

JUNE 2024

Contents

LogoFAIL — Six months after disclosure

LogoFAIL is a set of vulnerabilities originally disclosed in June 2023 by Binarly REsearch and publicly reported on December 6, 2023. LogoFAIL affects the image parsers used by UEFI firmware to display logos during boot. Device vendors allow for customization of the boot logo, mostly for corporate branding, widening the attack surface of UEFI firmware. Attackers can leverage these customization mechanisms to feed untrusted input into image parsers, exploiting vulnerabilities in those parsers.

As demonstrated by Binarly in this proof-of-concept video demo, LogoFAIL has significant impact on affected systems: code execution during the UEFI DXE phase gives full control to an attacker, which allows subverting modern OS defenses and to bootkit a target device.

But what makes LogoFAIL even more impactful is that any parser used in the UEFI firmware industry -- from common formats such as BMP to esoteric ones such as PCX and TGA -- as per Binarly's testing, contained software bugs. All Independent BIOS Vendors (IBVs) developed a parser containing vulnerabilities. Given the peculiar supply-chain nature of UEFI firmware, this means that any device vendor was affected by LogoFAIL and virtually any device running UEFI firmware contained a vulnerable image parser.

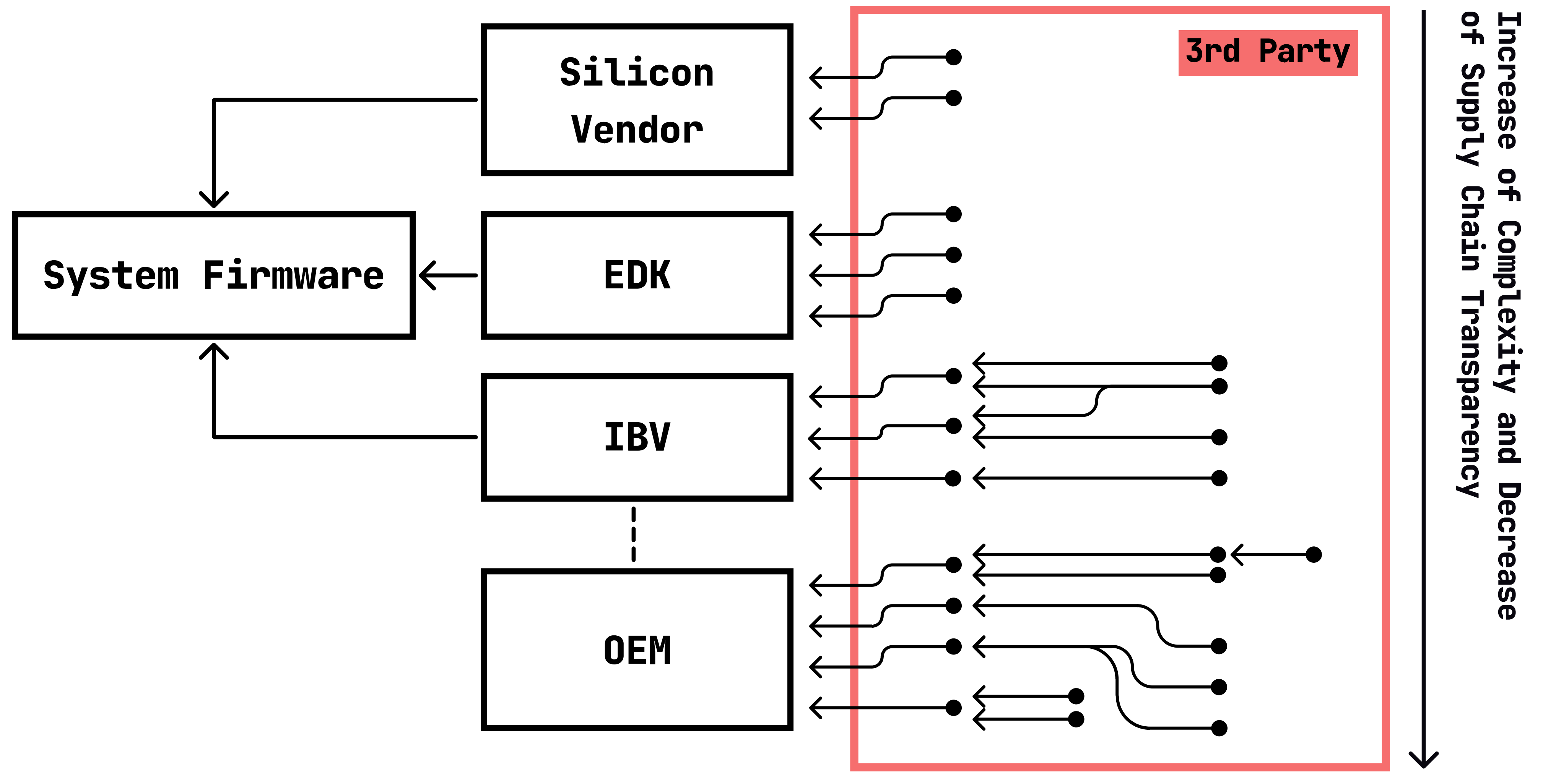
In this report, we provide a retrospective on our LogoFAIL research six months after the public disclosure at BlackHat EU. In the next sections, we analyze how IBVs and vendors responded to LogoFAIL, their updates and patches, and what remains vulnerable. Concurrently to the release of this analysis, we release all advisories detailing all the vulnerabilities discovered during this research. And more importantly, we publicly release our LogoFAIL detection rules on FwHunt, our free firmware vulnerability scanner, so that anyone can check their firmware against LogoFAIL.

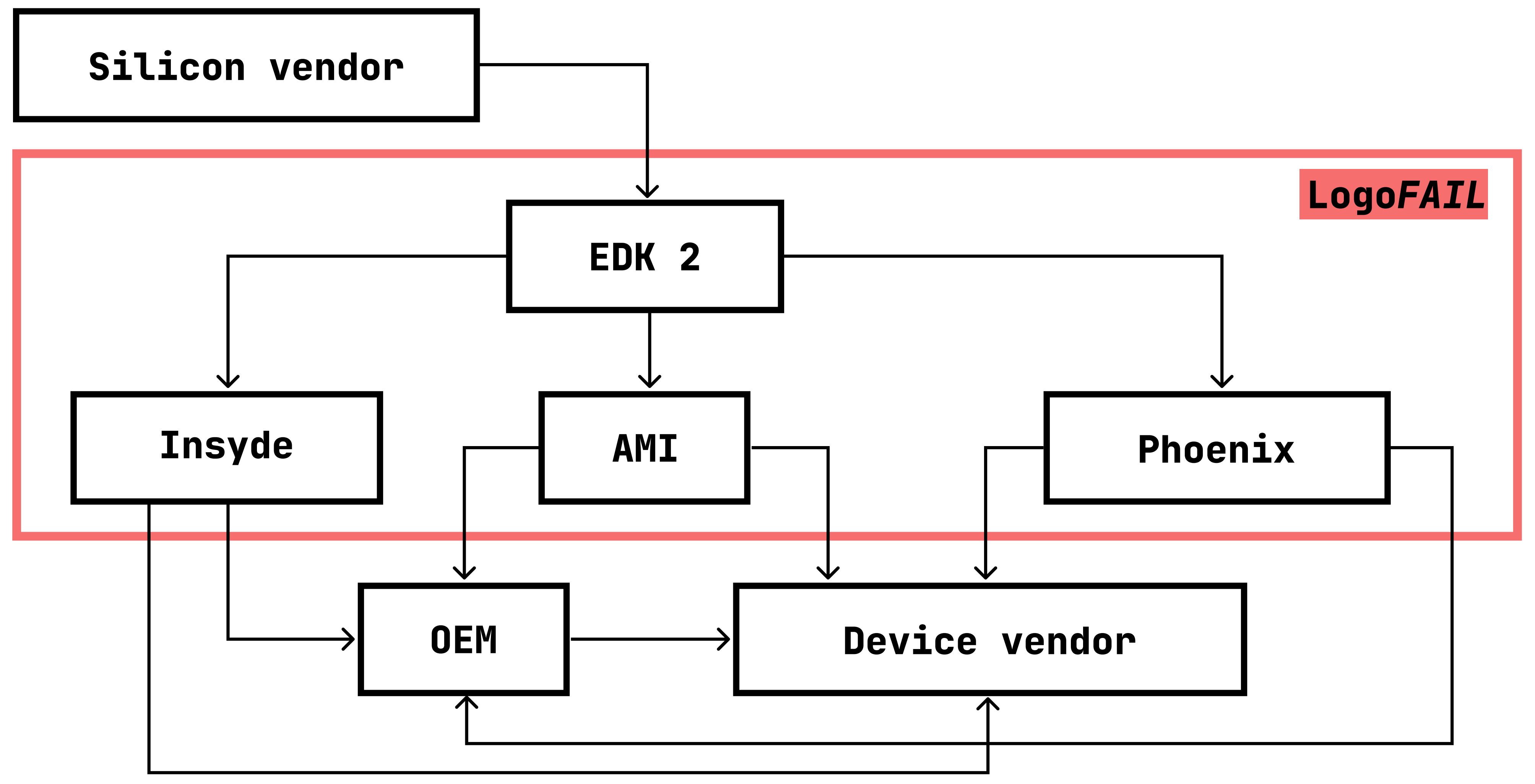
LogoFAIL and the UEFI supply chain

In this research, we dive deeper into the shallow waters of the firmware supply chain confusion and the blind trust in the reference code and firmware developers when we assume everything is fixed several months after disclosure.

In reality, six months after disclosure, LogoFAIL *remains a very concerning issue for the entire firmware ecosystem*. The figure below shows how the accumulation of dependencies from multiple layers of the software supply chain disproportionately increases complexity of this vulnerability on the firmware side. When the firmware code is delivered to the device, it consumes all upper supply chain layers with all the applied implications.

Even as we release 30+ LogoFAIL advisories to the public, we remained worried about the number of unfixed devices remaining in the field. Unfortunately, IBVs like AMI and Insyde assigned only one CVE to multiple advisories, and vulnerabilities have been shared with them during the disclosure process. The Binarly team raised several concerns about this lack of transparency and the need to assign more CVE IDs to the different instances of LogoFAIL vulnerabilities, but our efforts did not change the minds of AMI and Insyde, leading to impact for downstream device vendors.


Silicon Vendor
System Firmware
EDK
IBV
OEM
3rd Party
Increase of Complexity and Decrease of Supply Chain Transparency



LogoFAIL and the UEFI supply chain

Parser vulnerabilities are among the most common and pervasive types of issues and are found virtually anywhere, from personal devices to critical infrastructure. Under the hood, LogoFAIL is just “another” set of parser bugs, so what exactly makes it so difficult to address?

The answer to this question is multifaceted but it’s all connected to how LogoFAIL is deeply rooted within the UEFI ecosystem. The first factor is related to the asynchronous nature of the supply chain UEFI firmware industry, where IBVs provide reference implementations (based on the silicon vendor implementation of EDK2) to device vendors for further customization. As shown in the previous image, since LogoFAIL affects the reference implementations of IBVs, it’s transitively present in the codebases of every OEM and device vendor’s firmware. This tangled supply chain also means that patches produced by IBVs must be shared and merged by device vendors in their own repositories so that updated firmware can be built and shipped to their customers. Given all the parties involved in this process, it’s not unusual in the firmware industry to have a 180+ days patch delivery timeline for end users.

The second factor that makes LogoFAIL challenging to patch is that the affected parsers have been spread throughout the industry for years, making it difficult even for the vendors themselves to know where these parsers are embedded and thus to identify what is vulnerable and what is not.

The last factor is instead related to the sheer number of vulnerabilities that Binarly discovered during this research project, which resulted in 30 unique security vulnerabilities found in the tested parsers. This left IBVs without a clear strategy to address LogoFAIL. Some replaced their parser with supposedly safer third-party implementations, while others addressed each reported bug, occasionally releasing incomplete fixes.

Patching LogoFAIL by disabling logo customization

The most immediate and obvious way to fix LogoFAIL is to disable logo customization: if the image parser used by UEFI firmware cannot be reached with untrusted images, then an attacker cannot exploit the vulnerabilities in the parser itself.

We then revisited the original LogoFAIL advisories shared with IBVs and device vendors, downloading the updated firmware for each affected device. Notably, every firmware update specifically mentioned in its changelog that it had been patched against LogoFAIL.

As shown, ***none of the affected vendors decided to remove the logo customization functionality***, a decision that would effectively mitigate LogoFAIL.

IBV	OEM	Logo customization method	
insyde	acer	\EFI\OEM\AcerLogo.png \EFI\OEM\AcerLogo.jpg	⛔ not removed
	Lenovo	EFI\lenovo\logo + GetVariable(“LBLDESPFN”)	⛔ not removed
ami	intel	via iCHLogo Tool	⛔ not removed
	Lenovo	GetVariable(“LnvOemLogoData”) + User.gif GetVariable(“LnvOemLogoData”) + User.bmp	⛔ not removed
phoenix technologies	Lenovo	EFI\Lenovo\logo\mylogo EFI\Lenovo\logo\mylogo_WxH	⛔ not removed

Patching LogoFAIL by fixing reported vulnerabilities

Fixing these types of supply chain security issues requires developing a certain degree of transparency into all the layers of the software supply chain, specifically when it comes to protecting the foundation of platform security tied to device and firmware security layers.

In reality, the IBVs continue to play the “security by obscurity” game. The industry has learned multiple times that obscurity doesn’t benefit security. Still, in this case, the obscurity is related to the business model of IBVs monopolizing the market and dictating the rules of transparency, including the vulnerability disclosure process and details provided to NVD. That will change in the future with more pressure from a compliance standpoint, and Binarly is committed to recovering firmware supply chain transparency through our Binary Risk Intelligence technologies.

Disabling logo customization should only be treated as a first emergency response, not as a long term solution. Confirmed security vulnerabilities must be properly addressed, either by fixing the underlying software bug or by completely removing the offending code.

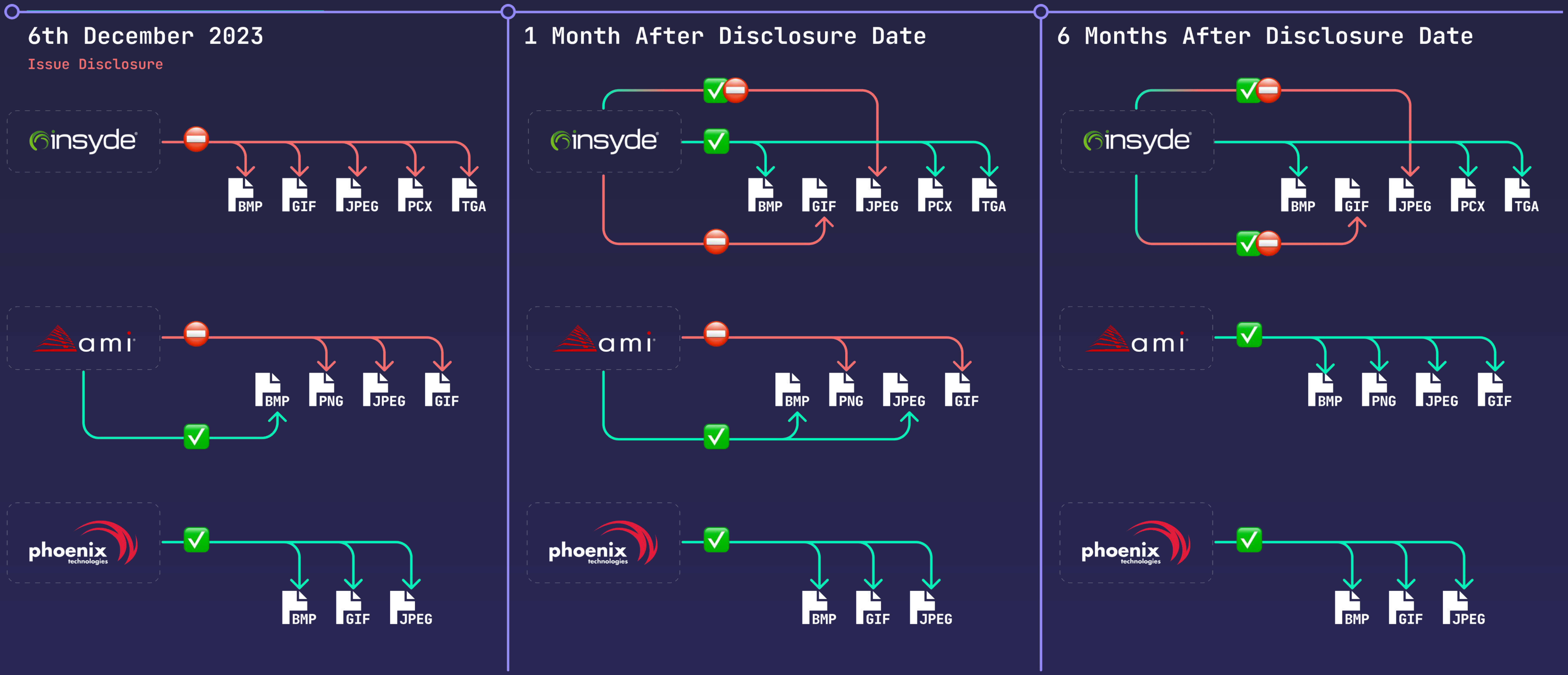
To understand how the firmware industry responded to LogoFAIL, we downloaded from the vendors’ websites three sets of UEFI firmware images, released just after the public disclosure date, one month after disclosure and 6 months after disclosure, respectively.

Referencing the chart on page 8, the first detail that catches the eye is that only one vendor -- *Phoenix Technologies* -- fixed all the bugs by the public issue disclosure date, while AMI only patched bugs related to the BMP parser, and Insyde patched none of the reported bugs.

To be more precise, it could be that the bugs were patched in the IBV’s reference implementation but given the intricate nature of the UEFI supply chain, these patches didn’t promptly reach OEMs, device vendors or end-users. One month after the public disclosure date, more bugs were fixed, indicating that public disclosure might have had a positive effect and pushed IBVs and vendors to accelerate their response to LogoFAIL. Finally, 6 months later, Binarly detected that most of the bugs have been correctly addressed by vendors, with only a few exceptions of fixes that were attempted but we deem incomplete. These incomplete fixes occur either because a vendor does not attempt to fix a reported bug at all, or because the attempted fixes do not address all conditions that can trigger the bug.

Our analysis revealed a second surprising and concerning finding: ***none of the IBVs removed any image parsers from their firmware***. This is particularly alarming as it goes against the recommendations we provided during private disclosure communications. Supporting outdated and obscure formats, like TGA or PCX, seems unnecessary. Similarly, complex file formats such as JPEG, PNG and GIF should not be included in firmware. Binarly’s recommendation remains unchanged: ***graphic elements must be converted to easier-to-parse formats (such as BMP) before being embedded in UEFI firmware***. This approach eliminates the dependency on complex and potentially unsafe third-party parsing libraries.

IBVs fixes comparison



FILE	Parser	🟥	Not fixed	✓🟥	Partial fix	✓	Fixed

IBVs* Independent BIOS Vendors

Current state of devices vulnerable to LogoFAIL

It was assumed that the fixes would be applied to the reference code, such as EDK and IBVs, and would then be consumed by downstream device vendors and OEMs. However, in reality, the situation is far more complex. After the disclosure, we notified all the parties over CERT/CC VINCE regarding our discoveries on assigned CVEs confusion where the number of vulnerabilities didn't match the number of assigned CVEs, and that really played a negative role in the adoption of the security fixes.

Technically, if you fix a vulnerability, the fix matches the CVE, but in reality, not all the security issues related to LogoFAIL can be fixed. Binarly Transparency Platform detects unfixed devices daily, and almost every device still contains a few unfixed vulnerabilities related to LogoFAIL.

LogoFAIL really puts a spotlight on the complexity of the UEFI firmware supply chain ecosystem: while patches were developed by IBVs, it takes time for them to propagate in the ecosystem and to be included in every device's firmware. This can be clearly seen by looking at the results of our latest scan on our internal dataset. Even when only looking at firmware released in 2024, we still have hundreds of products that remain unfixed.

Vulnerable devices by vendor

According to the latest scan of our internal dataset, hundreds of products with firmware released in 2024 remains vulnerable.

Distribution of vulnerable devices

68

IBV

insyde

OEMs

acer	DELL
FUJITSU	Lenovo
GIGABYTE	Others

669

IBV

ami

OEMs

acer	DELL
GIGABYTE	hp
intel	Lenovo
msi	SAMSUNG
SUPERMICRO	Others

15

IBV

phoenix technologies

OEM

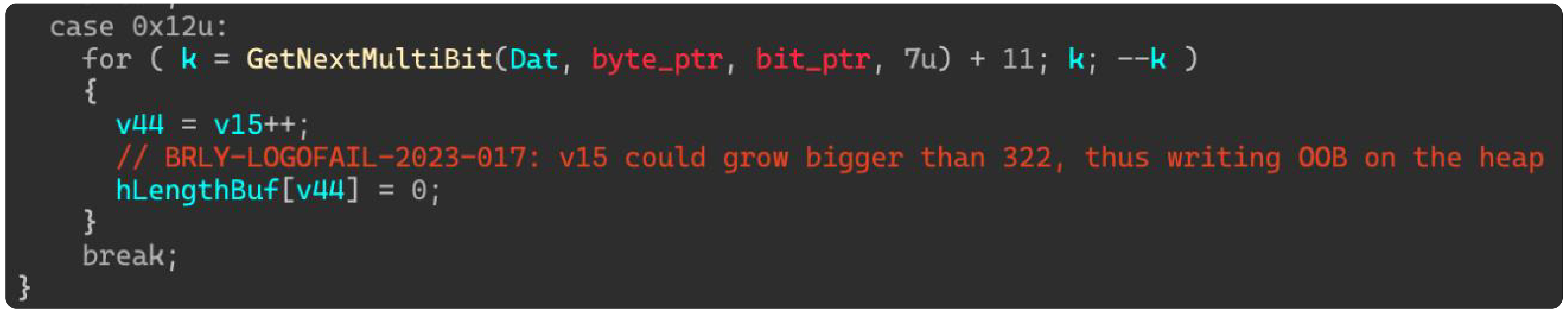
Lenovo	Others

A closer look at LogoFAIL patches

IBVs adopted different strategies to develop patches against the reported bugs. The first strategy, which was adopted by Phoenix, is to replace existing parsing libraries with another library called stb_image. While we fuzzed this image library and didn't find any crashes—a somewhat expected outcome since this library has been already extensively tested by the community—this library still contains parser for complex format and is written in C.

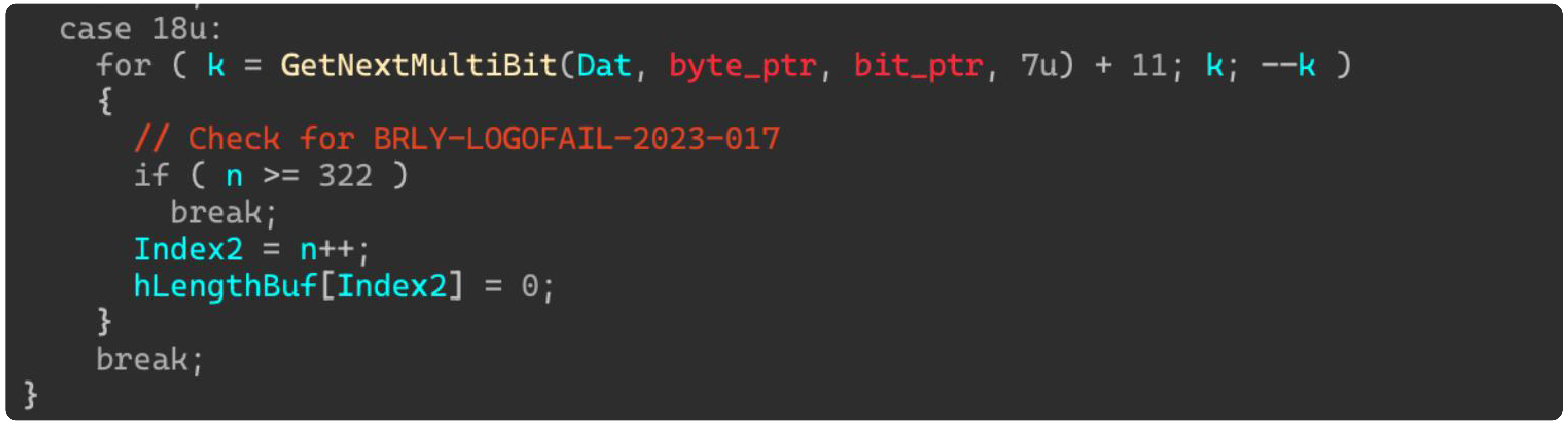
A similar approach has been adopted by AMI, but limited to their existing BMP parser that was swapped with the BMP parser included in EDK2.

On the other hand, the rest of AMI parsers and Insyde's parsers have been patched for each of the reported bugs. Both vendors enhanced security by adding more checks on untrusted inputs and the values derived from them. For example, BRLY-LOGOFAIL-2023-017 originated from the lack of checks on the index used to access a statically allocated buffer, leading to an. OOB memory write.



```
  case 0x12u:
    for ( k = GetNextMultiBit(Dat, byte_ptr, bit_ptr, 7u) + 11; k; --k )
    {
      v44 = v15++;
      // BRLY-LOGOFAIL-2023-017: v15 could grow bigger than 322, thus writing OOB on the heap
      hLengthBuf[v44] = 0;
    }
    break;
}
```

The patch implemented by AMI is straightforward, as it simply checks that the index does not exceed the length of hLengthBuf (322):



```
  case 18u:
    for ( k = GetNextMultiBit(Dat, byte_ptr, bit_ptr, 7u) + 11; k; --k )
    {
      // Check for BRLY-LOGOFAIL-2023-017
      if ( n >= 322 )
        break;
      Index2 = n++;
      hLengthBuf[Index2] = 0;
    }
    break;
}
```

The patching efforts from AMI and Insyde were mostly successful, except for ***one incomplete fix*** in Insyde's JPEG parsers and ***two unpatched bugs*** in Insyde's GIF parser.

A closer look at LogoFAIL patches

BRLY-LOGOFAIL-2023-ID	IBV	Patch Description	Patched	CVSS Score	CWE	Affected Platforms
BRLY-LOGOFAIL-2023-001	Insyde	Added checks on ImageOffset	✅	6.0 | Medium	CWE-200	X86/ARM
BRLY-LOGOFAIL-2023-002	Insyde	Added checks on BMP height and width	✅	8.2 | High	CWE-122	X86/ARM
BRLY-LOGOFAIL-2023-003	Insyde	Added checks on BMP height and width	✅	8.2 | High	CWE-122	X86/ARM
BRLY-LOGOFAIL-2023-004	Insyde	Added check for InitCodeSize before calling LZWDecoder	✅	8.2 | High	CWE-122, CWE-125	X86
BRLY-LOGOFAIL-2023-005	Insyde	Not patched	⛔	3.2 | Low	CWE-125	X86
BRLY-LOGOFAIL-2023-006	Insyde	Not patched	⛔	8.2 | High	CWE-122, CWE-125	X86
BRLY-LOGOFAIL-2023-007	Insyde	Added checks so that GIF width and height are not zero	✅	8.2 | High	CWE-122	X86
BRLY-LOGOFAIL-2023-008	Insyde	Added check for SosPtr, but it's incomplete as the check is not performed before each return	⛔	6.0 | Medium	CWE-476	X86/ARM
BRLY-LOGOFAIL-2023-009	Insyde	Introduced new variable to check that enough data is still present	✅	3.2 | Low	CWE-125	X86/ARM
BRLY-LOGOFAIL-2023-010	Insyde	Check on the BlockQtBuffPtr pointer before dereference	✅	6.0 | Medium	CWE-476	X86/ARM
BRLY-LOGOFAIL-2023-011	Insyde	Added check on index variable	✅	6.0 | Medium	CWE-200	X86/ARM
BRLY-LOGOFAIL-2023-012	Insyde	Added checks on TGA height and width	✅	8.2 | High	CWE-122, CWE-125	X86/ARM
BRLY-LOGOFAIL-2023-013	AMI	Fixed by switching to BMP parser from EDK2	✅	6.0 | Medium	CWE-200	X86/ARM
BRLY-LOGOFAIL-2023-014	AMI	Added validation on ImageSize variable	✅	3.2 | Low	CWE-125	X86
BRLY-LOGOFAIL-2023-015	AMI	Added check on PNG chunk length	✅	3.2 | Low	CWE-125	X86

A closer look at LogoFAIL patches

BRLY-LOGOFAIL-2023-ID	IBV	Patch Description	Patched	CVSS Score	CWE	Affected Platforms
BRLY-LOGOFAIL-2023-016	AMI	Indirectly patched by the fix for BRLY-LOGOFAIL-2023-018	✅	❗ 8.2 | High	CWE-122, CWE-190	X86
BRLY-LOGOFAIL-2023-017	AMI	Added check for array index	✅	❗ 8.2 | High	CWE-122	X86
BRLY-LOGOFAIL-2023-018	AMI	Added check on the multiplication between PNG width and height	✅	❗ 8.2 | High	CWE-122, CWE-190	X86
BRLY-LOGOFAIL-2023-019	AMI	Added check on allocation size is not 0	✅	❗ 8.2 | High	CWE-122, CWE-190	X86
BRLY-LOGOFAIL-2023-020	AMI	Added check that prevents index growing more than buffer size	✅	❗ 8.2 | High	CWE-787	X86
BRLY-LOGOFAIL-2023-021	AMI	Added check on JPEG marker length	✅	⚠ 3.2 | Low	CWE-125	X86
BRLY-LOGOFAIL-2023-022	AMI	Added check on the number of Huffman Table detected	✅	❗ 8.2 | High	CWE-787	X86
BRLY-LOGOFAIL-2023-023	AMI	Added check enforcing fp pointer to be inside the gBltBuf buffer	✅	❗ 8.2 | High	CWE-122	X86/ARM
BRLY-LOGOFAIL-2023-024	AMI	Added check on array index	✅	❗ 8.2 | High	CWE-787	X86/ARM
BRLY-LOGOFAIL-2023-025	Phoenix	Swapped to a new image parsing library	✅	⚠ 3.2 | Low	CWE-200	X86/ARM
BRLY-LOGOFAIL-2023-026	Phoenix	Swapped to a new image parsing library	✅	⚠ 3.2 | Low	CWE-200	X86/ARM
BRLY-LOGOFAIL-2023-027	Phoenix	Swapped to a new image parsing library	✅	❗ 8.2 | High	CWE-122, CWE-190	X86/ARM
BRLY-LOGOFAIL-2023-028	Phoenix	Swapped to a new image parsing library	✅	❗ 8.2 | High	CWE-787	X86/ARM
BRLY-LOGOFAIL-2023-029	Phoenix	Swapped to a new image parsing library	✅	⚠ 3.2 | Low	CWE-125	X86/ARM
BRLY-LOGOFAIL-2023-030	Phoenix	Swapped to a new image parsing library	✅	❗ 8.2 | High	CWE-787	X86/ARM

Conclusion

The discovery and attempts to mitigate the LogoFAIL set of vulnerabilities are perfect examples of complexities haunting the firmware supply chain ecosystem and how different layers could fail independently.

The industry needs a new approach to post-build validation that is guided by code inspection. Any detection logic solely based on vendor-provided information results in incomplete or inaccurate detection logic, as proven by LogoFAIL.

At Binarly, we are investing heavily in our Binary Intelligence Technology to provide post-patch validation based on semantic code properties to provide a deeper understanding of the nature of the code changes. Solving the software supply chain puzzle requires gaining more data insights from every layer of the software supply chain to get more transparency on the nature of the code changes and how vendors keep their promises on addressing high-impact vulnerabilities.

References

The Far-Reaching Consequences of LogoFAIL

Finding LogoFAIL: The Dangers of Image Parsing During System Boot

Inside the LogoFAIL PoC: From Integer Overflow to Arbitrary Code Execution

LogoFAIL: Security Implications of Image Parsing During System Boot

Patch Breakdown and Incomplete Fixes

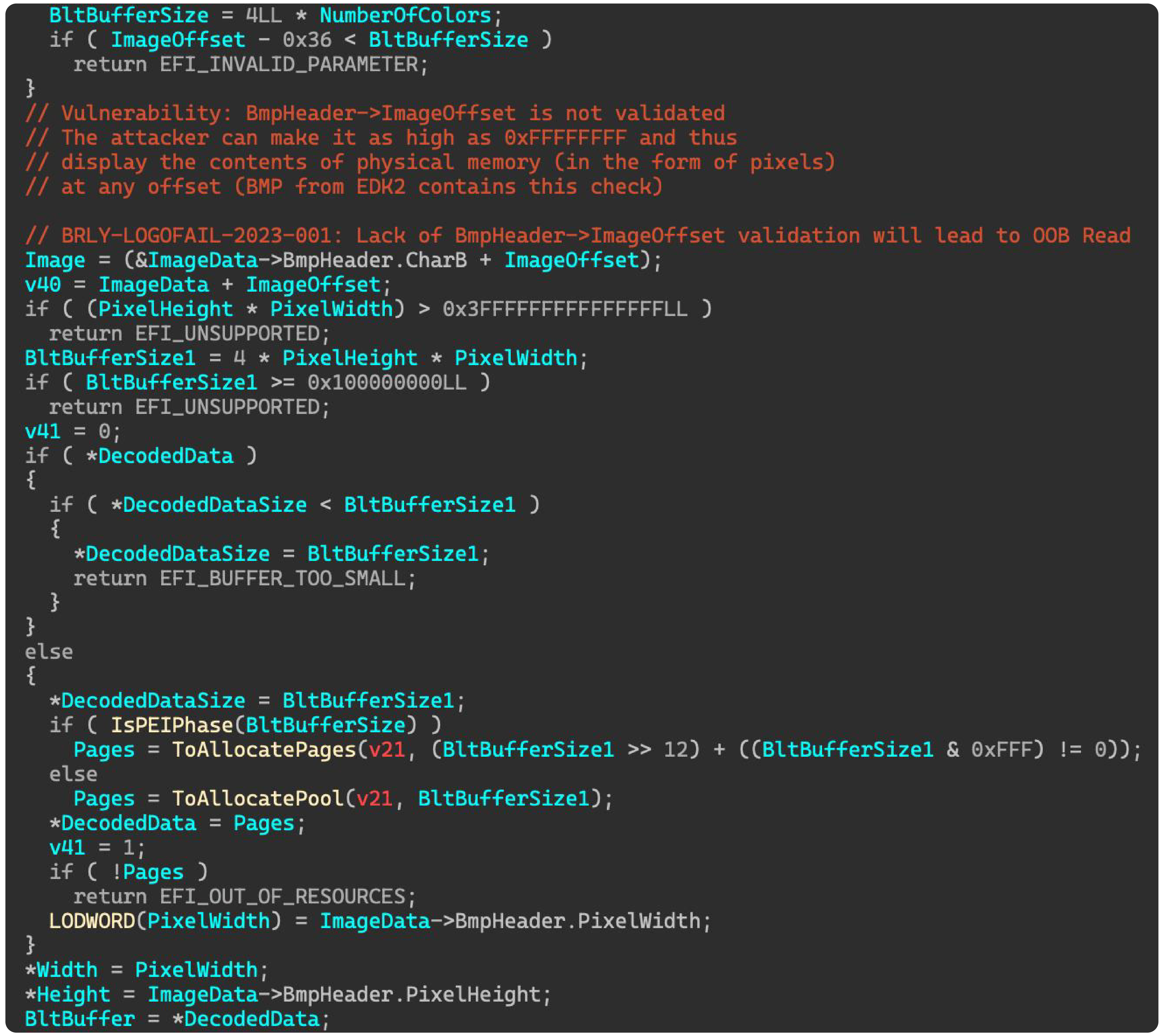
Insyde

Summary of the fixes

- We detected incomplete fixes for GIF and JPEG parsers
  - `BRLY-LOGOFAIL-2023-005 (GIF)`
  - `BRLY-LOGOFAIL-2023-006 (GIF)`
  - `BRLY-LOGOFAIL-2023-008 (JPEG)`
- All other parsers (BMP, PCX, TGA) were correctly fixed

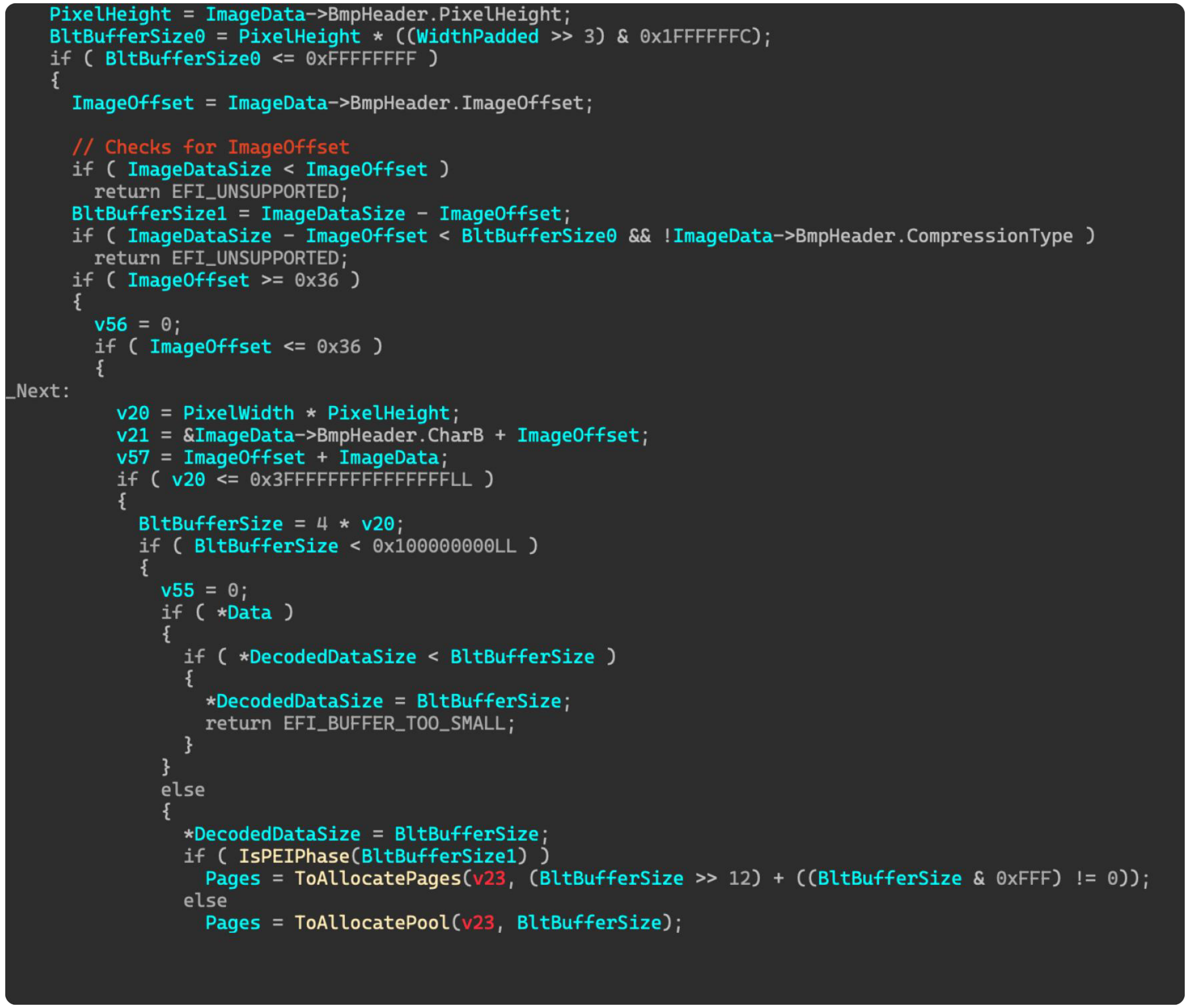
BmpDecoderDxe

BRLY-LOGOFAIL-2023-001 - fixed ✅



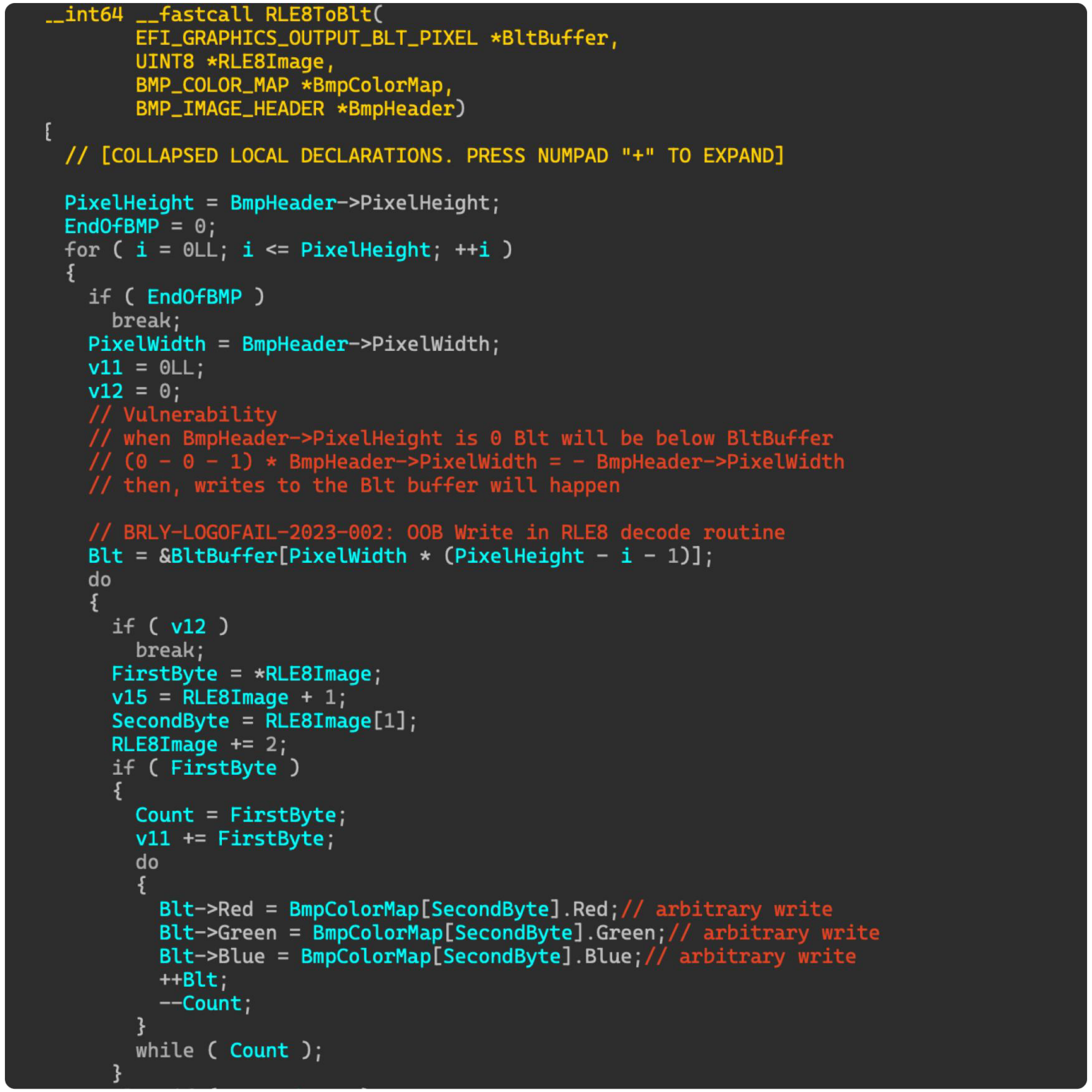
```
  BltBufferSize = 4LL * NumberOfColors;
  if ( ImageOffset - 0x36 < BltBufferSize )
    return EFI_INVALID_PARAMETER;
}
// Vulnerability: BmpHeader->ImageOffset is not validated
// The attacker can make it as high as 0xFFFFFFFF and thus
// display the contents of physical memory (in the form of pixels)
// at any offset (BMP from EDK2 contains this check)

// BRLY-LOGOFAIL-2023-001: Lack of BmpHeader->ImageOffset validation will lead to OOB Read
Image = (&ImageData->BmpHeader.CharB + ImageOffset);
v40 = ImageData + ImageOffset;
if ( (PixelHeight * PixelWidth) > 0x3FFFFFFFFFFFFFFFLL )
  return EFI_UNSUPPORTED;
BltBufferSize1 = 4 * PixelHeight * PixelWidth;
if ( BltBufferSize1 >= 0x100000000LL )
  return EFI_UNSUPPORTED;
v41 = 0;
if ( *DecodedData )
{
  if ( *DecodedDataSize < BltBufferSize1 )
  {
    *DecodedDataSize = BltBufferSize1;
    return EFI_BUFFER_TOO_SMALL;
  }
}
else
{
  *DecodedDataSize = BltBufferSize1;
  if ( IsPEIPhase(BltBufferSize) )
    Pages = ToAllocatePages(v21, (BltBufferSize1 >> 12) + ((BltBufferSize1 & 0xFFF) != 0));
  else
    Pages = ToAllocatePool(v21, BltBufferSize1);
  *DecodedData = Pages;
  v41 = 1;
  if ( !Pages )
    return EFI_OUT_OF_RESOURCES;
  LODWORD(PixelWidth) = ImageData->BmpHeader.PixelWidth;
}
*Width = PixelWidth;
*Height = ImageData->BmpHeader.PixelHeight;
BltBuffer = *DecodedData;
```

Fix: ImageOffset **checks added**



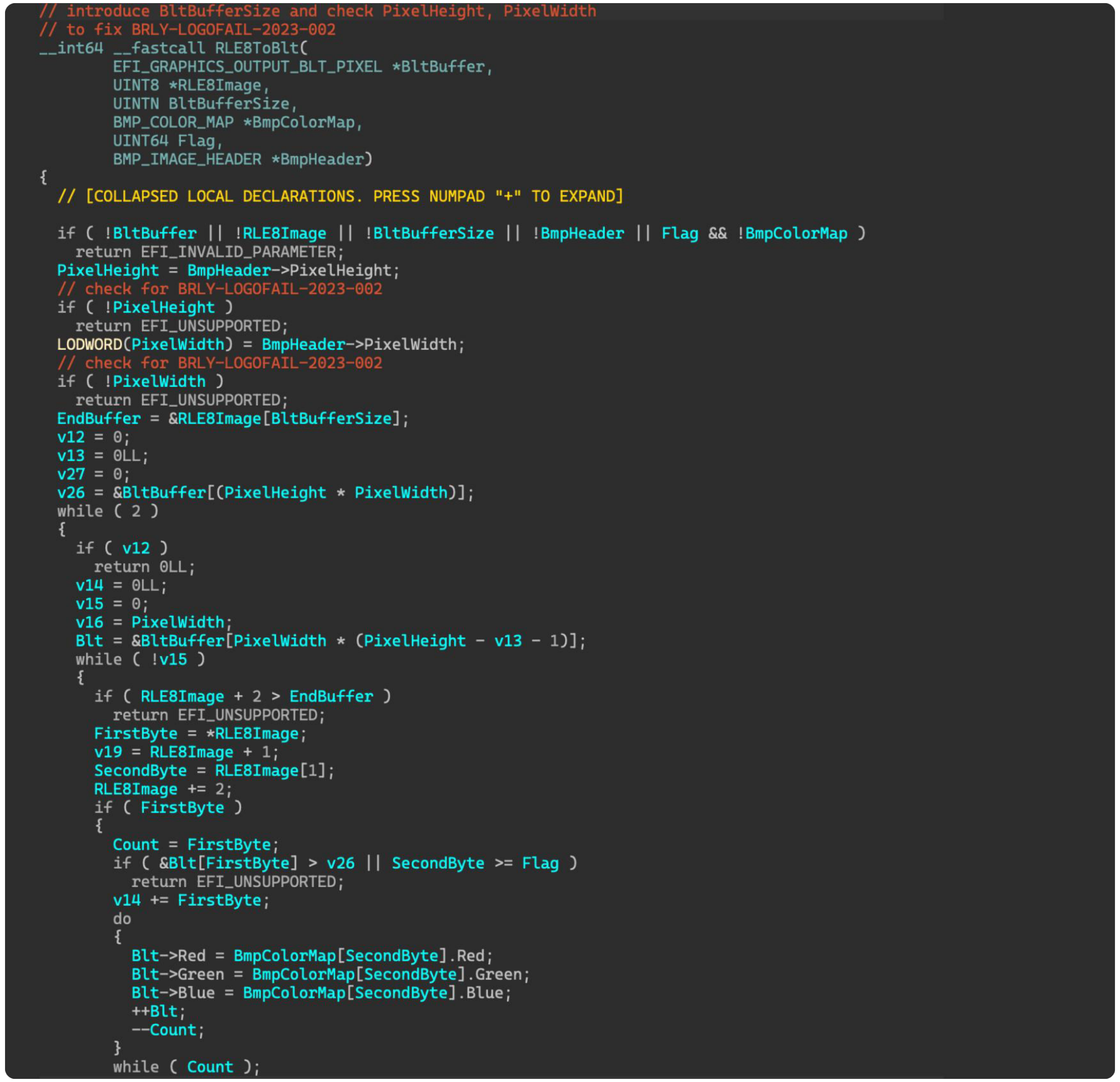
```
    PixelHeight = ImageData->BmpHeader.PixelHeight;
    BltBufferSize0 = PixelHeight * ((WidthPadded >> 3) & 0x1FFFFFFC);
    if ( BltBufferSize0 <= 0xFFFFFFFF )
    {
      ImageOffset = ImageData->BmpHeader.ImageOffset;

      // Checks for ImageOffset
      if ( ImageDataSize < ImageOffset )
        return EFI_UNSUPPORTED;
      BltBufferSize1 = ImageDataSize - ImageOffset;
      if ( ImageDataSize - ImageOffset < BltBufferSize0 && !ImageData->BmpHeader.CompressionType )
        return EFI_UNSUPPORTED;
      if ( ImageOffset >= 0x36 )
      {
        v56 = 0;
        if ( ImageOffset <= 0x36 )
        {
_Next:
          v20 = PixelWidth * PixelHeight;
          v21 = &ImageData->BmpHeader.CharB + ImageOffset;
          v57 = ImageOffset + ImageData;
          if ( v20 <= 0x3FFFFFFFFFFFFFFFLL )
          {
            BltBufferSize = 4 * v20;
            if ( BltBufferSize < 0x100000000LL )
            {
              v55 = 0;
              if ( *Data )
              {
                if ( *DecodedDataSize < BltBufferSize )
                {
                  *DecodedDataSize = BltBufferSize;
                  return EFI_BUFFER_TOO_SMALL;
                }
              }
              else
              {
                *DecodedDataSize = BltBufferSize;
                if ( IsPEIPhase(BltBufferSize1) )
                  Pages = ToAllocatePages(v23, (BltBufferSize >> 12) + ((BltBufferSize & 0xFFF) != 0));
                else
                  Pages = ToAllocatePool(v23, BltBufferSize);
```

BRLY-LOGOFAIL-2023-002 - fixed ✅



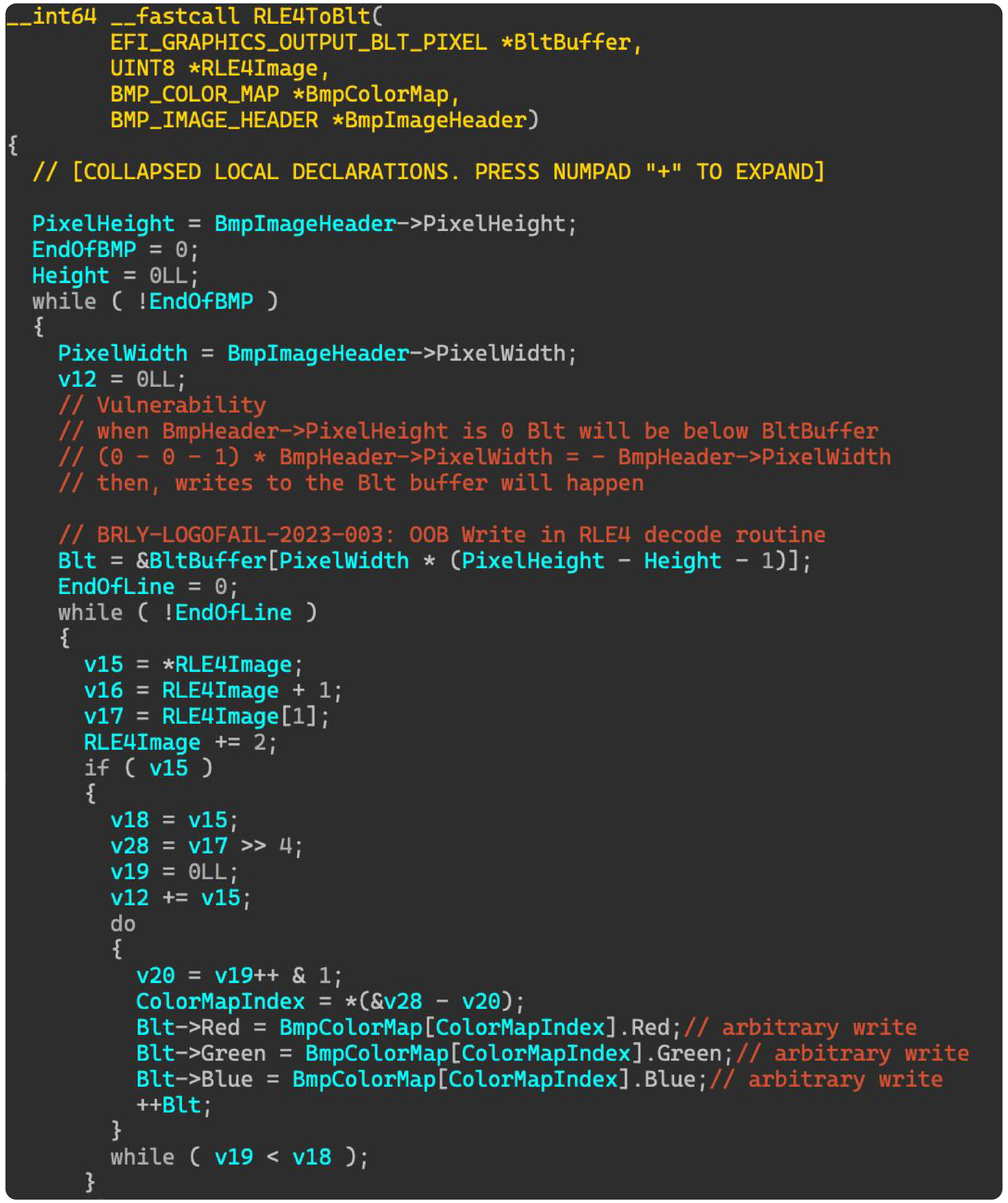
```
__int64 __fastcall RLE8ToBlt(
        EFI_GRAPHICS_OUTPUT_BLT_PIXEL *BltBuffer,
        UINT8 *RLE8Image,
        BMP_COLOR_MAP *BmpColorMap,
        BMP_IMAGE_HEADER *BmpHeader)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  PixelHeight = BmpHeader->PixelHeight;
  EndOfBMP = 0;
  for ( i = 0LL; i <= PixelHeight; ++i )
  {
    if ( EndOfBMP )
      break;
    PixelWidth = BmpHeader->PixelWidth;
    v11 = 0LL;
    v12 = 0;
    // Vulnerability
    // when BmpHeader->PixelHeight is 0 Blt will be below BltBuffer
    // (0 - 0 - 1) * BmpHeader->PixelWidth = - BmpHeader->PixelWidth
    // then, writes to the Blt buffer will happen

    // BRLY-LOGOFAIL-2023-002: OOB Write in RLE8 decode routine
    Blt = &BltBuffer[PixelWidth * (PixelHeight - i - 1)];
    do
    {
      if ( v12 )
        break;
      FirstByte = *RLE8Image;
      v15 = RLE8Image + 1;
      SecondByte = RLE8Image[1];
      RLE8Image += 2;
      if ( FirstByte )
      {
        Count = FirstByte;
        v11 += FirstByte;
        do
        {
          Blt->Red = BmpColorMap[SecondByte].Red;// arbitrary write
          Blt->Green = BmpColorMap[SecondByte].Green;// arbitrary write
          Blt->Blue = BmpColorMap[SecondByte].Blue;// arbitrary write
          ++Blt;
          --Count;
        }
        while ( Count );
      }
```

Fix: changed function prototype to take into account BltBufferSize and add checks for BmpHeader→PixelHeight, BmpHeader→PixelWidth



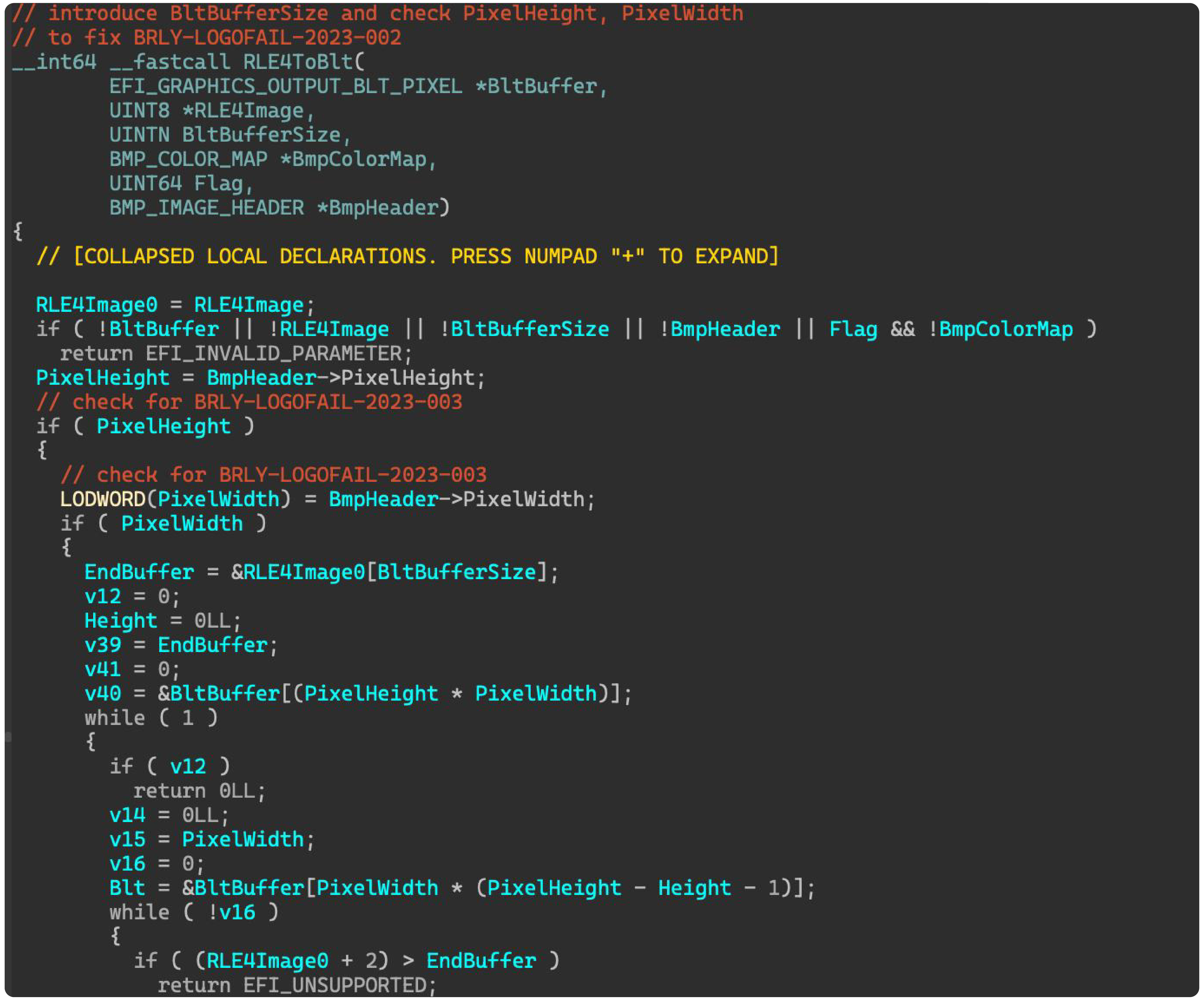
```
// introduce BltBufferSize and check PixelHeight, PixelWidth
// to fix BRLY-LOGOFAIL-2023-002
__int64 __fastcall RLE8ToBlt(
        EFI_GRAPHICS_OUTPUT_BLT_PIXEL *BltBuffer,
        UINT8 *RLE8Image,
        UINTN BltBufferSize,
        BMP_COLOR_MAP *BmpColorMap,
        UINT64 Flag,
        BMP_IMAGE_HEADER *BmpHeader)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  if ( !BltBuffer || !RLE8Image || !BltBufferSize || !BmpHeader || Flag && !BmpColorMap )
    return EFI_INVALID_PARAMETER;
  PixelHeight = BmpHeader->PixelHeight;
  // check for BRLY-LOGOFAIL-2023-002
  if ( !PixelHeight )
    return EFI_UNSUPPORTED;
  LODWORD(PixelWidth) = BmpHeader->PixelWidth;
  // check for BRLY-LOGOFAIL-2023-002
  if ( !PixelWidth )
    return EFI_UNSUPPORTED;
  EndBuffer = &RLE8Image[BltBufferSize];
  v12 = 0;
  v13 = 0LL;
  v27 = 0;
  v26 = &BltBuffer[(PixelHeight * PixelWidth)];
  while ( 2 )
  {
    if ( v12 )
      return 0LL;
    v14 = 0LL;
    v15 = 0;
    v16 = PixelWidth;
    Blt = &BltBuffer[PixelWidth * (PixelHeight - v13 - 1)];
    while ( !v15 )
    {
      if ( RLE8Image + 2 > EndBuffer )
        return EFI_UNSUPPORTED;
      FirstByte = *RLE8Image;
      v19 = RLE8Image + 1;
      SecondByte = RLE8Image[1];
      RLE8Image += 2;
      if ( FirstByte )
      {
        Count = FirstByte;
        if ( &Blt[FirstByte] > v26 || SecondByte >= Flag )
          return EFI_UNSUPPORTED;
        v14 += FirstByte;
        do
        {
          Blt->Red = BmpColorMap[SecondByte].Red;
          Blt->Green = BmpColorMap[SecondByte].Green;
          Blt->Blue = BmpColorMap[SecondByte].Blue;
          ++Blt;
          --Count;
        }
        while ( Count );
```

BRLY-LOGOFAIL-2023-003 - fixed ✅



```
__int64 __fastcall RLE4ToBlt(
        EFI_GRAPHICS_OUTPUT_BLT_PIXEL *BltBuffer,
        UINT8 *RLE4Image,
        BMP_COLOR_MAP *BmpColorMap,
        BMP_IMAGE_HEADER *BmpImageHeader)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  PixelHeight = BmpImageHeader->PixelHeight;
  EndOfBMP = 0;
  Height = 0LL;
  while ( !EndOfBMP )
  {
    PixelWidth = BmpImageHeader->PixelWidth;
    v12 = 0LL;
    // Vulnerability
    // when BmpHeader->PixelHeight is 0 Blt will be below BltBuffer
    // (0 - 0 - 1) * BmpHeader->PixelWidth = - BmpHeader->PixelWidth
    // then, writes to the Blt buffer will happen

    // BRLY-LOGOFAIL-2023-003: OOB Write in RLE4 decode routine
    Blt = &BltBuffer[PixelWidth * (PixelHeight - Height - 1)];
    EndOfLine = 0;
    while ( !EndOfLine )
    {
      v15 = *RLE4Image;
      v16 = RLE4Image + 1;
      v17 = RLE4Image[1];
      RLE4Image += 2;
      if ( v15 )
      {
        v18 = v15;
        v28 = v17 >> 4;
        v19 = 0LL;
        v12 += v15;
        do
        {
          v20 = v19++ & 1;
          ColorMapIndex = *(&v28 - v20);
          Blt->Red = BmpColorMap[ColorMapIndex].Red;// arbitrary write
          Blt->Green = BmpColorMap[ColorMapIndex].Green;// arbitrary write
          Blt->Blue = BmpColorMap[ColorMapIndex].Blue;// arbitrary write
          ++Blt;
        }
        while ( v19 < v18 );
      }
```

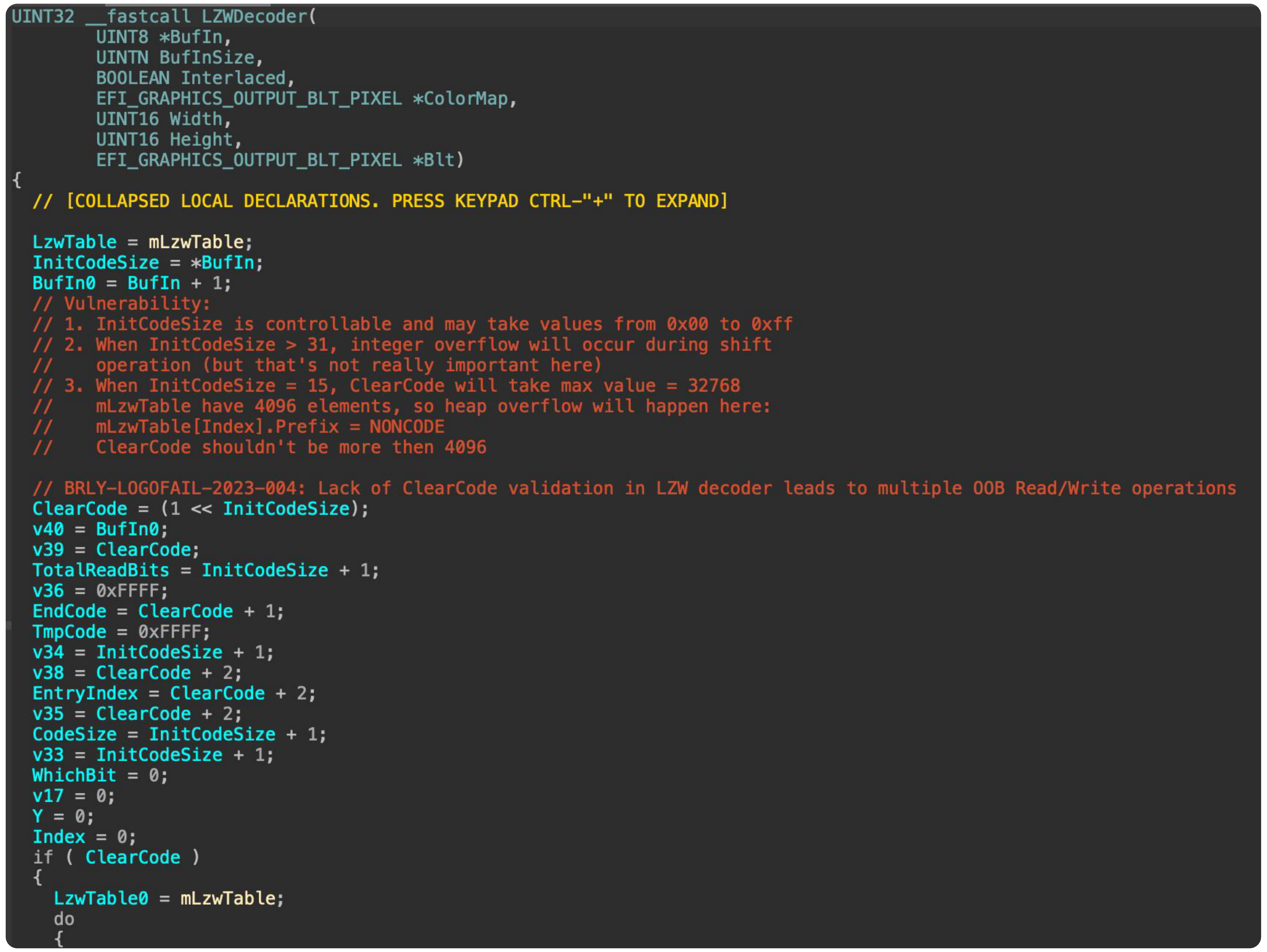
Fix: changed function prototype to take into account `BltBufferSize` **and add checks for** `BmpHeader→PixelHeight`, `BmpHeader→PixelWidth`



```
// introduce BltBufferSize and check PixelHeight, PixelWidth
// to fix BRLY-LOGOFAIL-2023-002
__int64 __fastcall RLE4ToBlt(
        EFI_GRAPHICS_OUTPUT_BLT_PIXEL *BltBuffer,
        UINT8 *RLE4Image,
        UINTN BltBufferSize,
        BMP_COLOR_MAP *BmpColorMap,
        UINT64 Flag,
        BMP_IMAGE_HEADER *BmpHeader)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  RLE4Image0 = RLE4Image;
  if ( !BltBuffer || !RLE4Image || !BltBufferSize || !BmpHeader || Flag && !BmpColorMap )
    return EFI_INVALID_PARAMETER;
  PixelHeight = BmpHeader->PixelHeight;
  // check for BRLY-LOGOFAIL-2023-003
  if ( PixelHeight )
  {
    // check for BRLY-LOGOFAIL-2023-003
    LODWORD(PixelWidth) = BmpHeader->PixelWidth;
    if ( PixelWidth )
    {
      EndBuffer = &RLE4Image0[BltBufferSize];
      v12 = 0;
      Height = 0LL;
      v39 = EndBuffer;
      v41 = 0;
      v40 = &BltBuffer[(PixelHeight * PixelWidth)];
      while ( 1 )
      {
        if ( v12 )
          return 0LL;
        v14 = 0LL;
        v15 = PixelWidth;
        v16 = 0;
        Blt = &BltBuffer[PixelWidth * (PixelHeight - Height - 1)];
        while ( !v16 )
        {
          if ( (RLE4Image0 + 2) > EndBuffer )
            return EFI_UNSUPPORTED;
```

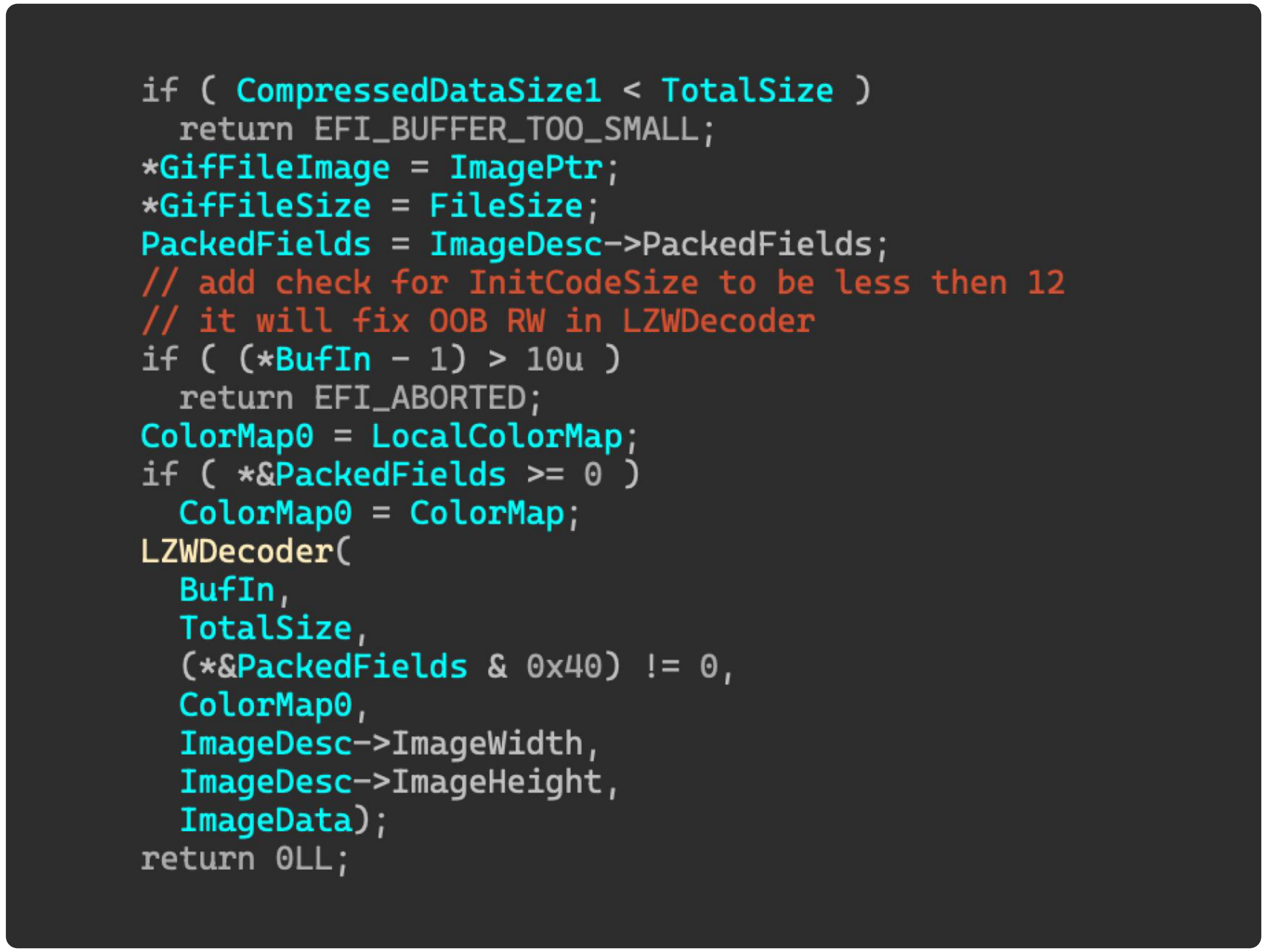
GifDecoderDxe

BRLY-LOGOFAIL-2023-004 – **fixed** ✅



```
UINT32 __fastcall LZWDecoder(
        UINT8 *BufIn,
        UINTN BufInSize,
        BOOLEAN Interlaced,
        EFI_GRAPHICS_OUTPUT_BLT_PIXEL *ColorMap,
        UINT16 Width,
        UINT16 Height,
        EFI_GRAPHICS_OUTPUT_BLT_PIXEL *Blt)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  LzwTable = mLzwTable;
  InitCodeSize = *BufIn;
  BufIn0 = BufIn + 1;
  // Vulnerability:
  // 1. InitCodeSize is controllable and may take values from 0x00 to 0xff
  // 2. When InitCodeSize > 31, integer overflow will occur during shift
  //    operation (but that's not really important here)
  // 3. When InitCodeSize = 15, ClearCode will take max value = 32768
  //    mLzwTable have 4096 elements, so heap overflow will happen here:
  //    mLzwTable[Index].Prefix = NONCODE
  //    ClearCode shouldn't be more then 4096

  // BRLY-LOGOFAIL-2023-004: Lack of ClearCode validation in LZW decoder leads to multiple OOB Read/Write operations
  ClearCode = (1 << InitCodeSize);
  v40 = BufIn0;
  v39 = ClearCode;
  TotalReadBits = InitCodeSize + 1;
  v36 = 0xFFFF;
  EndCode = ClearCode + 1;
  TmpCode = 0xFFFF;
  v34 = InitCodeSize + 1;
  v38 = ClearCode + 2;
  EntryIndex = ClearCode + 2;
  v35 = ClearCode + 2;
  CodeSize = InitCodeSize + 1;
  v33 = InitCodeSize + 1;
  WhichBit = 0;
  v17 = 0;
  Y = 0;
  Index = 0;
  if ( ClearCode )
  {
    LzwTable0 = mLzwTable;
    do
    {
```

Fix: **added check for** `InitCodeSize` **before** `LZWDecoder` **function call**



```
if ( CompressedDataSize1 < TotalSize )
  return EFI_BUFFER_TOO_SMALL;
*GifFileImage = ImagePtr;
*GifFileSize = FileSize;
PackedFields = ImageDesc->PackedFields;
// add check for InitCodeSize to be less then 12
// it will fix OOB RW in LZWDecoder
if ( (*BufIn - 1) > 10u )
  return EFI_ABORTED;
ColorMap0 = LocalColorMap;
if ( *&PackedFields >= 0 )
  ColorMap0 = ColorMap;
LZWDecoder(
  BufIn,
  TotalSize,
  (*&PackedFields & 0x40) != 0,
  ColorMap0,
  ImageDesc->ImageWidth,
  ImageDesc->ImageHeight,
  ImageData);
return 0LL;
```

BRLY-LOGOFAIL-2023-005 – **not fixed** ⛔

```
if ( ClearCode )
{
  LzwTable0 = mLzwTable;
  do
  {
    // Vulnerability
    // OOB Write here
    LzwTable0->Suffix = Index++;
    LzwTable0->Prefix = 0xFFFF;                    // NONCODE
    ++LzwTable0;
  }
  while ( Index < ClearCode );
  LzwTable = mLzwTable;
}
// Vulnerability:
// Incomplete check since in GetCode() function 4 bytes will be obtained
// from BufInSize + ReadBits / 8
// Address sanitizer will detect OOB Read even on valid binaries

// BRLY-LOGOFAIL-2023-005: Weak index checking leads to CompressedData OOB Read
if ( TotalReadBits >> 3 <= BufInSize )
{
  Height0 = Height;
  OutStack = mOutStack;
  do
  {
    GetCodeIndex = WhichBit >> 3;
    v24 = WhichBit & 7;
    WhichBit += CodeSize;
    // Vulnerability:
    // Code comes from the content of the GIF, but is not validated
    // It will lead to OOB Reads/Write since TmpCode/EntryIndex indexes will
    // depend from it
    Code = ((1 << CodeSize) - 1) & (*&BufIn0[GetCodeIndex] >> v24);
    if ( Code == EndCode )
      return WhichBit;
    if ( Code == ClearCode )
    {
      EntryIndex = v38;
      CodeSize = TotalReadBits;
      v35 = v38;
      TmpCode = ClearCode;
      v33 = TotalReadBits;
      v36 = ClearCode;
      goto LABEL_40;
    }
    if ( Y == Height0 )
      return WhichBit;
```

BRLY-LOGOFAIL-2023-006 – **not fixed** ⛔

```
WhichBit += CodeSize;
// Vulnerability:
// Code comes from the content of the GIF, but is not validated
// It will lead to OOB Reads/Write since TmpCode/EntryIndex indexes will
// depend from it

// BRLY-LOGOFAIL-2023-006: Lack of Code validation in LZW decoder leads to multiple OOB Read/Write operations
Code = ((1 << CodeSize) - 1) & (*&BufIn0[GetCodeIndex] >> v24);
if ( Code == EndCode )
  return WhichBit;
if ( Code == ClearCode )
{
  EntryIndex = v38;
  CodeSize = TotalReadBits;
  v35 = v38;
  TmpCode = ClearCode;
  v33 = TotalReadBits;
  v36 = ClearCode;
  goto LABEL_40;
}
if ( Y == Height0 )
  return WhichBit;
StackIndex = -1;
if ( Code >= EntryIndex )
{
  if ( TmpCode == ClearCode )
    goto LABEL_44;
  StackIndex = 0;
  Prefix = TmpCode;
  if ( TmpCode != 0xFFFF )
  {
    do
    {
      OutStack[++StackIndex] = LzwTable[Prefix].Suffix;
      Prefix = LzwTable[Prefix].Prefix;
    }
    while ( Prefix != 0xFFFF );
    CodeSize = v33;
    Height0 = Height;
  }
  *OutStack = OutStack[StackIndex];
}
else
{
  v27 = Code;
  if ( Code != 0xFFFF )
  {
    do
    {
      OutStack[++StackIndex] = LzwTable[v27].Suffix;
      v27 = LzwTable[v27].Prefix;
    }
    while ( v27 != 0xFFFF );
    Height0 = Height;
  }
  if ( TmpCode == ClearCode )
    goto LABEL_23;
}
// Vulnerability:
// OOB Write here: mLzwTable[EntryIndex].Prefix = PrevCode
// Due to the lack of Code validation
EntryIndex0 = EntryIndex++;
v35 = EntryIndex;
LzwTable[EntryIndex0].Prefix = TmpCode;
LzwTable[EntryIndex0].Suffix = OutStack[StackIndex];
```

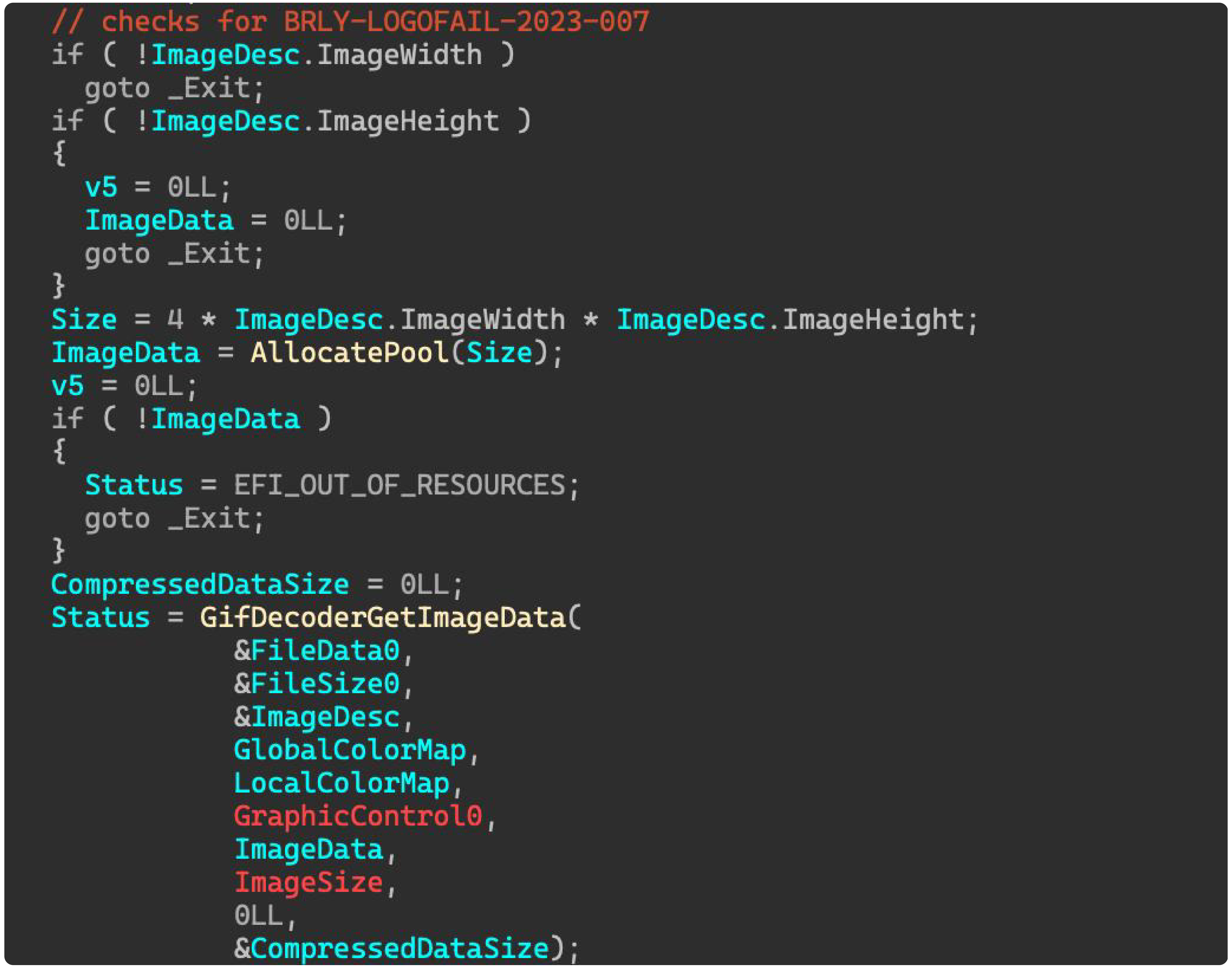
BRLY-LOGOFAIL-2023-007 – **fixed** ✅

```
// Vulnerability:
// there are no check for ImageSize
// wen ImageSize is 0, AllocatePool will return valid pointer to empty buffer
// This will then lead to an OOB Read
ImageData = AllocatePool(4 * ImageDesc.ImageWidth * ImageDesc.ImageHeight);
if ( !ImageData )
  break;
CompressedDataSize = 0i64;
Status = GifDecoderGetImageData(
           &FileData0,
           &FileSize0,
           &ImageDesc,
           GlobalColorMap,
           LocalColorMap,
           GraphicControl0,
           ImageData,
           ImageSize,
           0i64,
           &CompressedDataSize);
```

```
// Vulnerability:
// According to EDK2 description of AllocatePool function: If
// AllocationSize is 0, then a valid buffer of 0 size is returned

// Blt will be allocated in function H2OHiiCreateAnimationFromMem:
// ImageData = (EFI_GRAPHICS_OUTPUT_BLT_PIXEL *)AllocatePool(ImageSize);
// Where ImageSize = ImageDesc.ImageWidth * ImageDesc.ImageHeight *
// sizeof(EFI_GRAPHICS_OUTPUT_BLT_PIXEL)
// Thus, a write to a buffer of size 0 can occur here
// However, this is unlikely to result in a DoS at boot time

// crashes/id:000026,sig:06,src:000281+000255,time:1765104,execs:71361,op:splice,rep:4
// crashes/id:000027,sig:06,src:000281+000255,time:1765152,execs:71363,op:splice,rep:16

// BRLY-LOGOFAIL-2023-007: Unchecked ImageSize (which depends on ImageWidth and ImageHeight) results
// in allocation of a zero-sized buffer and subsequent writing to it
Blt[(X + Width * Y)] = ColorMap[ColorMapIndex];
if ( v17 == Width )
{
  if ( Interlaced )
  {
    if ( (Y & 7) != 0 )
    {
      if ( (Y & 3) != 0 )
      {
        if ( (Y & 1) != 0 )
        {
          Y += 2;
        }
        else
        {
          Y += 4;
          if ( Y >= Height0 )
            Y = 1;
        }
      }
      else
      {
        Y += 8;
        if ( Y >= Height0 )
          Y = 2;
      }
    }
    else
    {
      Y += 8;
      if ( Y >= Height0 )
        Y = 4;
    }
  }
```

`BRLY-LOGOFAIL-2023-007` – **fixed** ✅

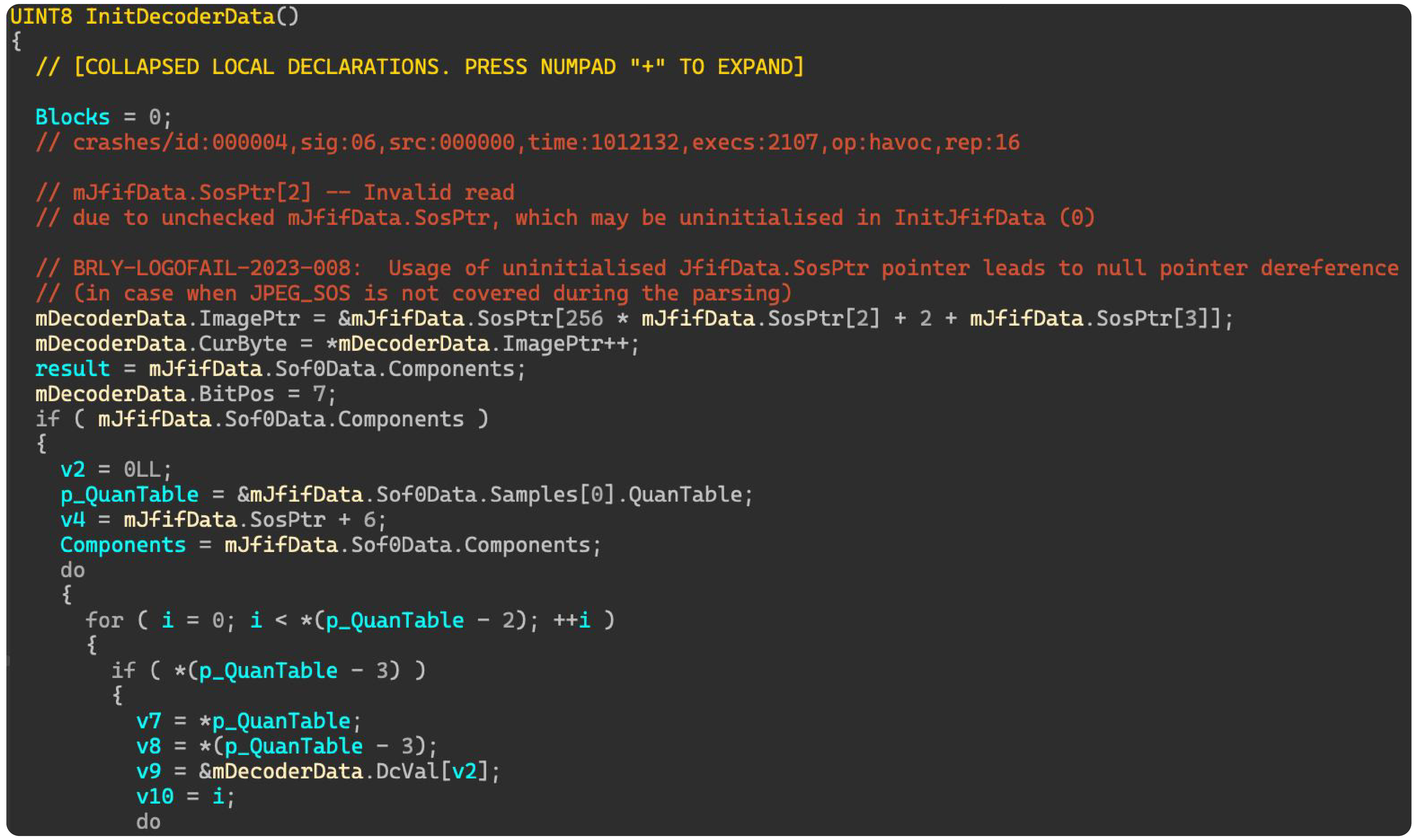
***Fix:* added check that `ImageDesc.ImageWidth` and `ImageDesc.ImageHeight` are not equal to zero, so allocation size cannot take zero value**



```
// checks for BRLY-LOGOFAIL-2023-007
if ( !ImageDesc.ImageWidth )
  goto _Exit;
if ( !ImageDesc.ImageHeight )
{
  v5 = 0LL;
  ImageData = 0LL;
  goto _Exit;
}
Size = 4 * ImageDesc.ImageWidth * ImageDesc.ImageHeight;
ImageData = AllocatePool(Size);
v5 = 0LL;
if ( !ImageData )
{
  Status = EFI_OUT_OF_RESOURCES;
  goto _Exit;
}
CompressedDataSize = 0LL;
Status = GifDecoderGetImageData(
           &FileData0,
           &FileSize0,
           &ImageDesc,
           GlobalColorMap,
           LocalColorMap,
           GraphicControl0,
           ImageData,
           ImageSize,
           0LL,
           &CompressedDataSize);
```

JpegDecoderDxe

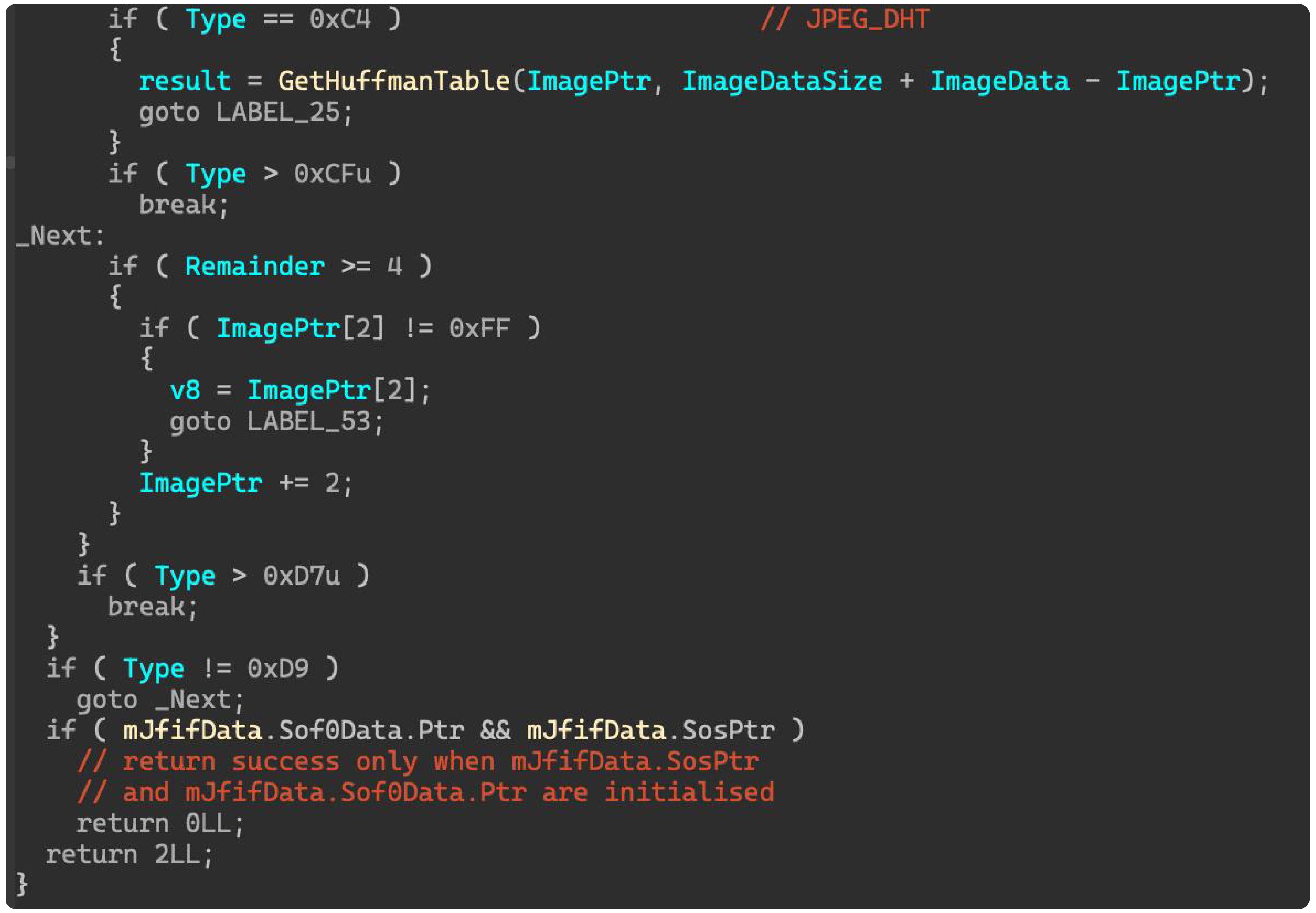
BRLY-LOGOFAIL-2023-008: incomplete fix ⛔



```
UINT8 InitDecoderData()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  Blocks = 0;
  // crashes/id:000004,sig:06,src:000000,time:1012132,execs:2107,op:havoc,rep:16

  // mJfifData.SosPtr[2] -- Invalid read
  // due to unchecked mJfifData.SosPtr, which may be uninitialised in InitJfifData (0)

  // BRLY-LOGOFAIL-2023-008:  Usage of uninitialised JfifData.SosPtr pointer leads to null pointer dereference
  // (in case when JPEG_SOS is not covered during the parsing)
  mDecoderData.ImagePtr = &mJfifData.SosPtr[256 * mJfifData.SosPtr[2] + 2 + mJfifData.SosPtr[3]];
  mDecoderData.CurByte = *mDecoderData.ImagePtr++;
  result = mJfifData.Sof0Data.Components;
  mDecoderData.BitPos = 7;
  if ( mJfifData.Sof0Data.Components )
  {
    v2 = 0LL;
    p_QuanTable = &mJfifData.Sof0Data.Samples[0].QuanTable;
    v4 = mJfifData.SosPtr + 6;
    Components = mJfifData.Sof0Data.Components;
    do
    {
      for ( i = 0; i < *(p_QuanTable - 2); ++i )
      {
        if ( *(p_QuanTable - 3) )
        {
          v7 = *p_QuanTable;
          v8 = *(p_QuanTable - 3);
          v9 = &mDecoderData.DcVal[v2];
          v10 = i;
          do
```

Fix: there is a check in `InitJfifData()` function for `SosPtr`, however, it's incomplete as this check is not performed before each successful return.

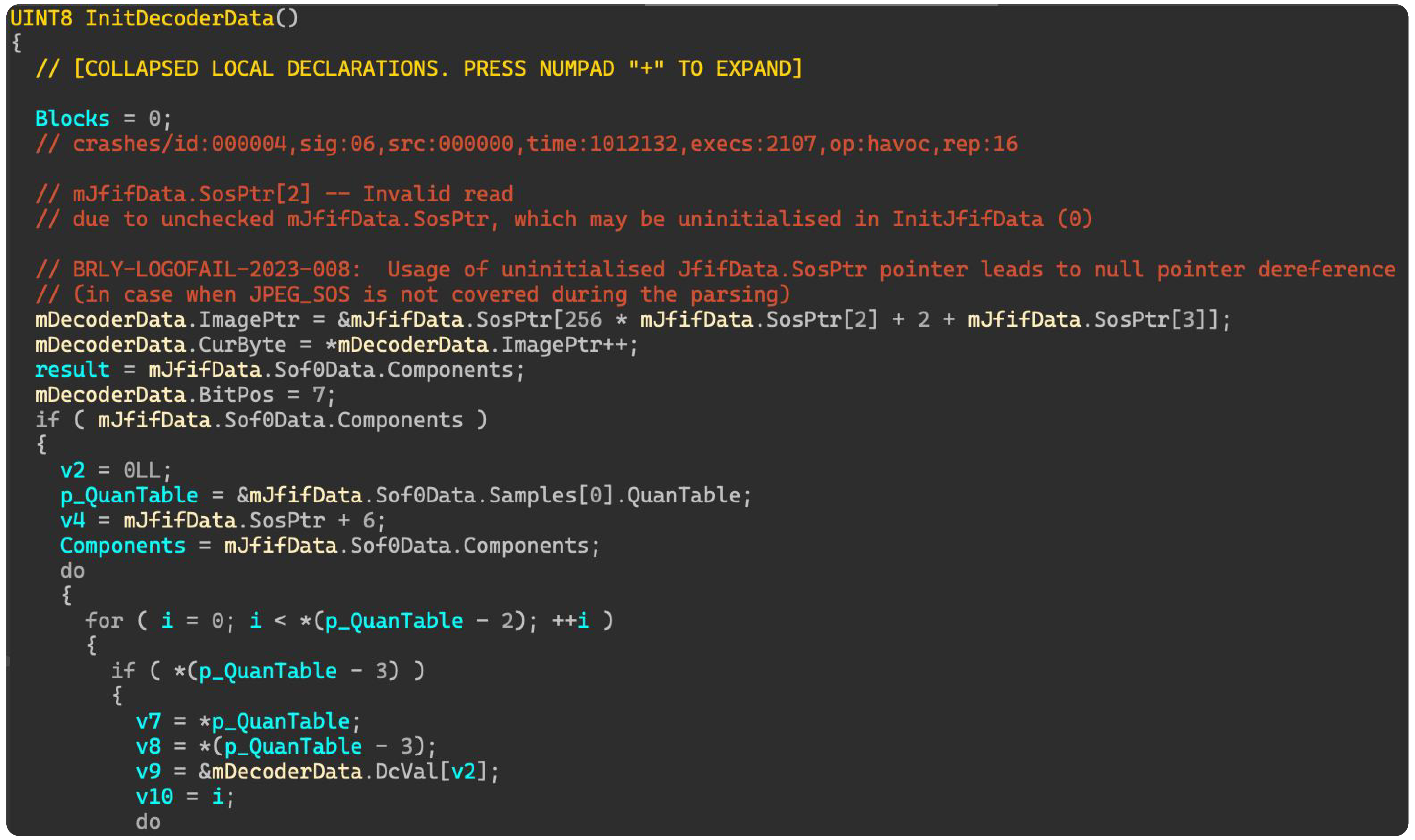
Added check:



```
      if ( Type == 0xC4 )                              // JPEG_DHT
      {
        result = GetHuffmanTable(ImagePtr, ImageDataSize + ImageData - ImagePtr);
        goto LABEL_25;
      }
      if ( Type > 0xCFu )
        break;
_Next:
      if ( Remainder >= 4 )
      {
        if ( ImagePtr[2] != 0xFF )
        {
          v8 = ImagePtr[2];
          goto LABEL_53;
        }
        ImagePtr += 2;
      }
    }
    if ( Type > 0xD7u )
      break;
  }
  if ( Type != 0xD9 )
    goto _Next;
  if ( mJfifData.Sof0Data.Ptr && mJfifData.SosPtr )
    // return success only when mJfifData.SosPtr
    // and mJfifData.Sof0Data.Ptr are initialised
    return 0LL;
  return 2LL;
}
```

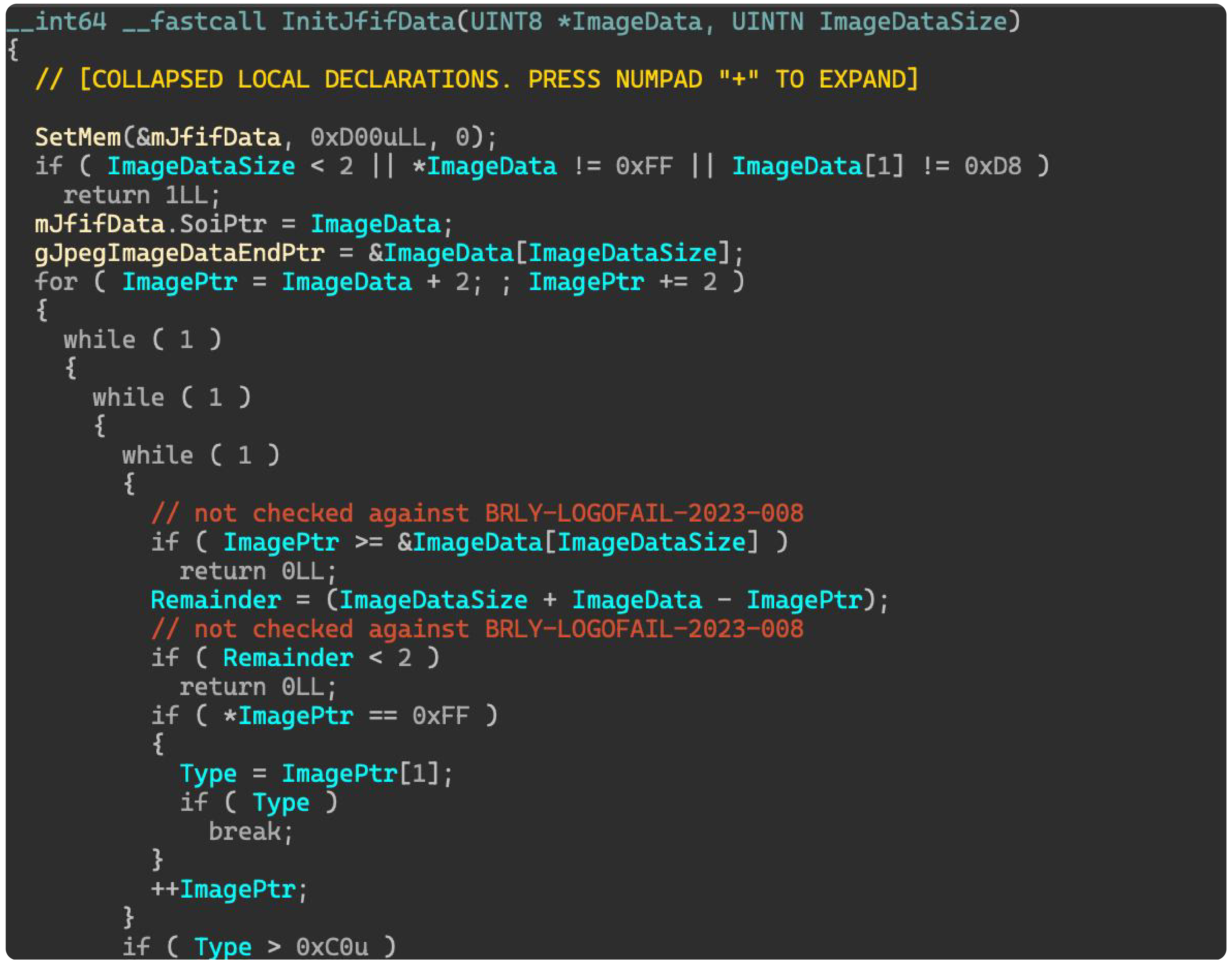
JpegDecoderDxe

BRLY-LOGOFAIL-2023-008: **incomplete fix** ⛔



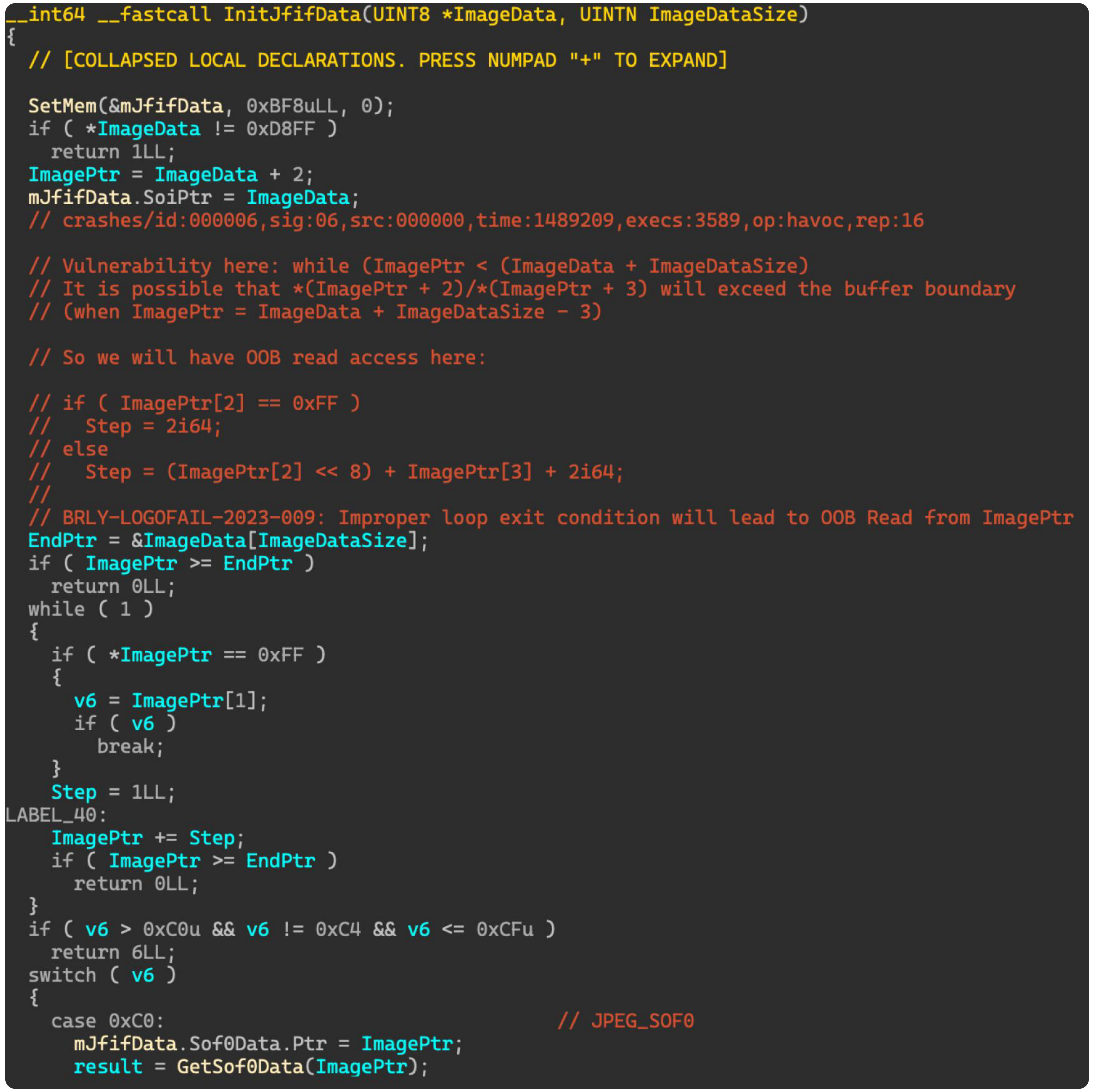
```
UINT8 InitDecoderData()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  Blocks = 0;
  // crashes/id:000004,sig:06,src:000000,time:1012132,execs:2107,op:havoc,rep:16

  // mJfifData.SosPtr[2] -- Invalid read
  // due to unchecked mJfifData.SosPtr, which may be uninitialised in InitJfifData (0)

  // BRLY-LOGOFAIL-2023-008:  Usage of uninitialised JfifData.SosPtr pointer leads to null pointer dereference
  // (in case when JPEG_SOS is not covered during the parsing)
  mDecoderData.ImagePtr = &mJfifData.SosPtr[256 * mJfifData.SosPtr[2] + 2 + mJfifData.SosPtr[3]];
  mDecoderData.CurByte = *mDecoderData.ImagePtr++;
  result = mJfifData.Sof0Data.Components;
  mDecoderData.BitPos = 7;
  if ( mJfifData.Sof0Data.Components )
  {
    v2 = 0LL;
    p_QuanTable = &mJfifData.Sof0Data.Samples[0].QuanTable;
    v4 = mJfifData.SosPtr + 6;
    Components = mJfifData.Sof0Data.Components;
    do
    {
      for ( i = 0; i < *(p_QuanTable - 2); ++i )
      {
        if ( *(p_QuanTable - 3) )
        {
          v7 = *p_QuanTable;
          v8 = *(p_QuanTable - 3);
          v9 = &mDecoderData.DcVal[v2];
          v10 = i;
          do
```

Locations of missing checks (if any of returns below will be triggered, null pointer dereference in InitDecoderData function will occur):



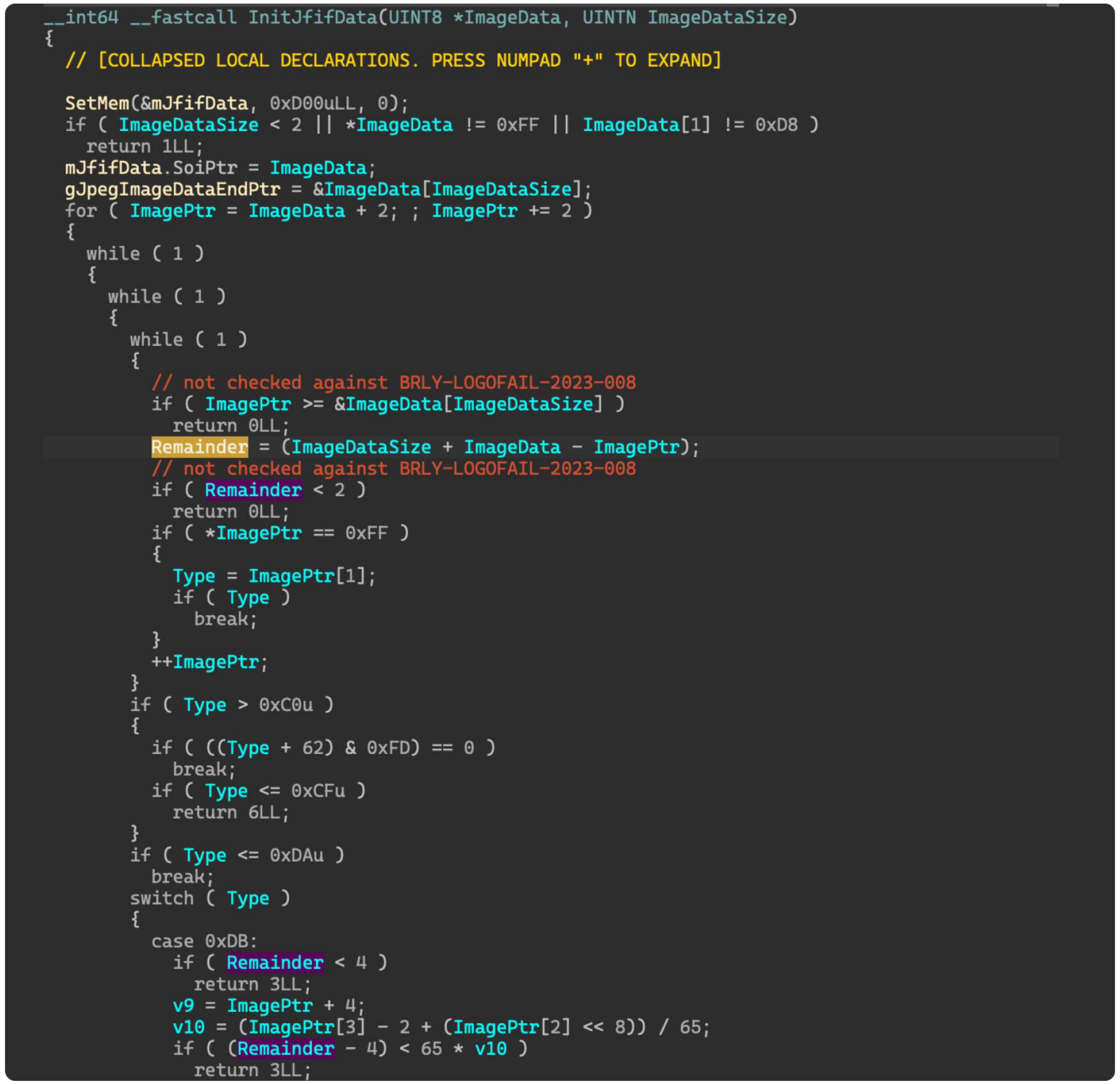
```
__int64 __fastcall InitJfifData(UINT8 *ImageData, UINTN ImageDataSize)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  SetMem(&mJfifData, 0xD00uLL, 0);
  if ( ImageDataSize < 2 || *ImageData != 0xFF || ImageData[1] != 0xD8 )
    return 1LL;
  mJfifData.SoiPtr = ImageData;
  gJpegImageDataEndPtr = &ImageData[ImageDataSize];
  for ( ImagePtr = ImageData + 2; ; ImagePtr += 2 )
  {
    while ( 1 )
    {
      while ( 1 )
      {
        while ( 1 )
        {
          // not checked against BRLY-LOGOFAIL-2023-008
          if ( ImagePtr >= &ImageData[ImageDataSize] )
            return 0LL;
          Remainder = (ImageDataSize + ImageData - ImagePtr);
          // not checked against BRLY-LOGOFAIL-2023-008
          if ( Remainder < 2 )
            return 0LL;
          if ( *ImagePtr == 0xFF )
          {
            Type = ImagePtr[1];
            if ( Type )
              break;
          }
          ++ImagePtr;
        }
        if ( Type > 0xC0u )
```

BRLY-LOGOFAIL-2023-009 – **fixed** ✅



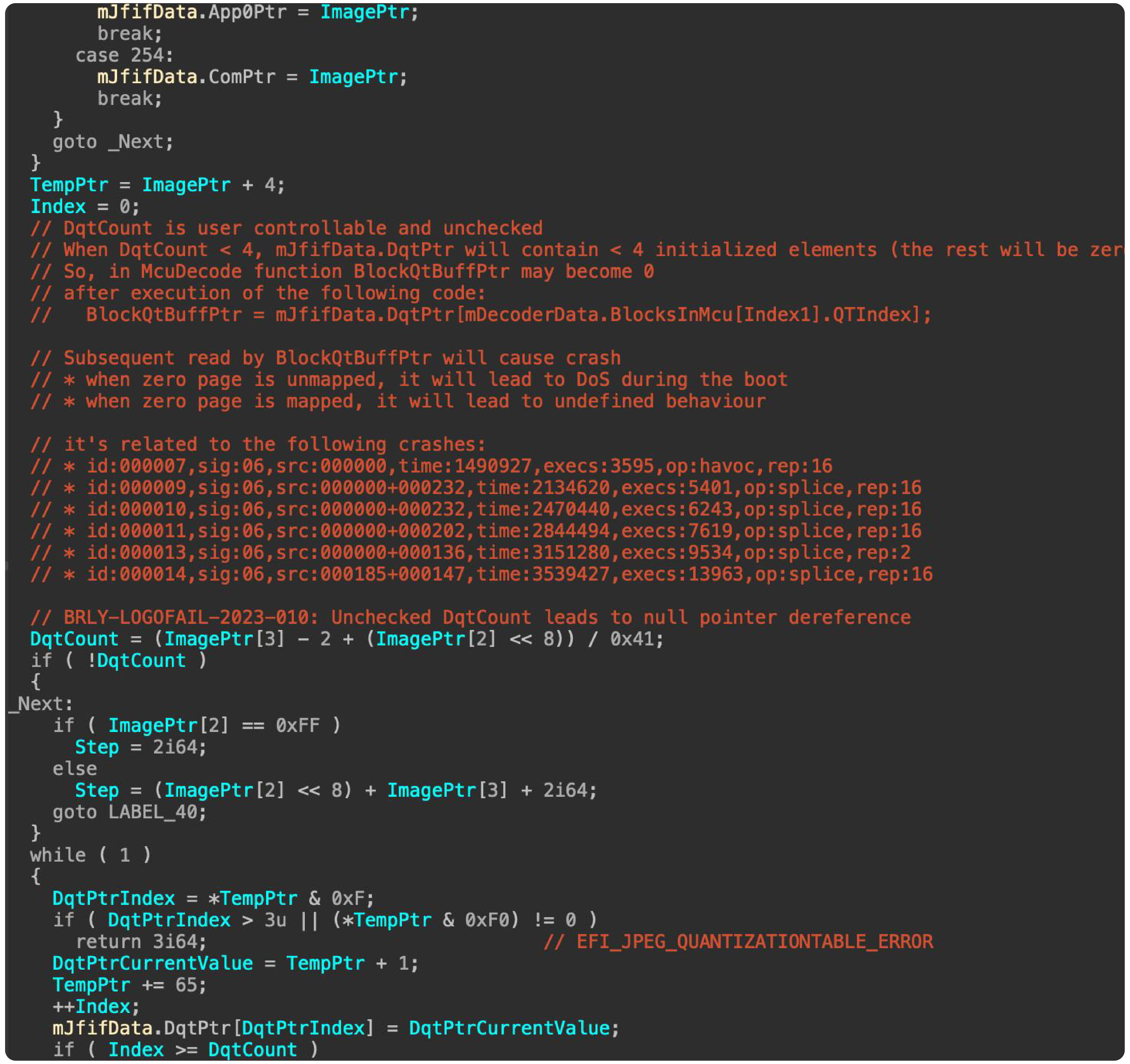
```
__int64 __fastcall InitJfifData(UINT8 *ImageData, UINTN ImageDataSize)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  SetMem(&mJfifData, 0xBF8uLL, 0);
  if ( *ImageData != 0xD8FF )
    return 1LL;
  ImagePtr = ImageData + 2;
  mJfifData.SoiPtr = ImageData;
  // crashes/id:000006,sig:06,src:000000,time:1489209,execs:3589,op:havoc,rep:16

  // Vulnerability here: while (ImagePtr < (ImageData + ImageDataSize)
  // It is possible that *(ImagePtr + 2)/*(ImagePtr + 3) will exceed the buffer boundary
  // (when ImagePtr = ImageData + ImageDataSize - 3)

  // So we will have OOB read access here:

  // if ( ImagePtr[2] == 0xFF )
  //   Step = 2i64;
  // else
  //   Step = (ImagePtr[2] << 8) + ImagePtr[3] + 2i64;
  //
  // BRLY-LOGOFAIL-2023-009: Improper loop exit condition will lead to OOB Read from ImagePtr
  EndPtr = &ImageData[ImageDataSize];
  if ( ImagePtr >= EndPtr )
    return 0LL;
  while ( 1 )
  {
    if ( *ImagePtr == 0xFF )
    {
      v6 = ImagePtr[1];
      if ( v6 )
        break;
    }
    Step = 1LL;
LABEL_40:
    ImagePtr += Step;
    if ( ImagePtr >= EndPtr )
      return 0LL;
  }
  if ( v6 > 0xC0u && v6 != 0xC4 && v6 <= 0xCFu )
    return 6LL;
  switch ( v6 )
  {
    case 0xC0:                              // JPEG_SOF0
      mJfifData.Sof0Data.Ptr = ImagePtr;
      result = GetSof0Data(ImagePtr);
```

Fix: fixed by introducing the `Remainder` variable that will be checked before read operations

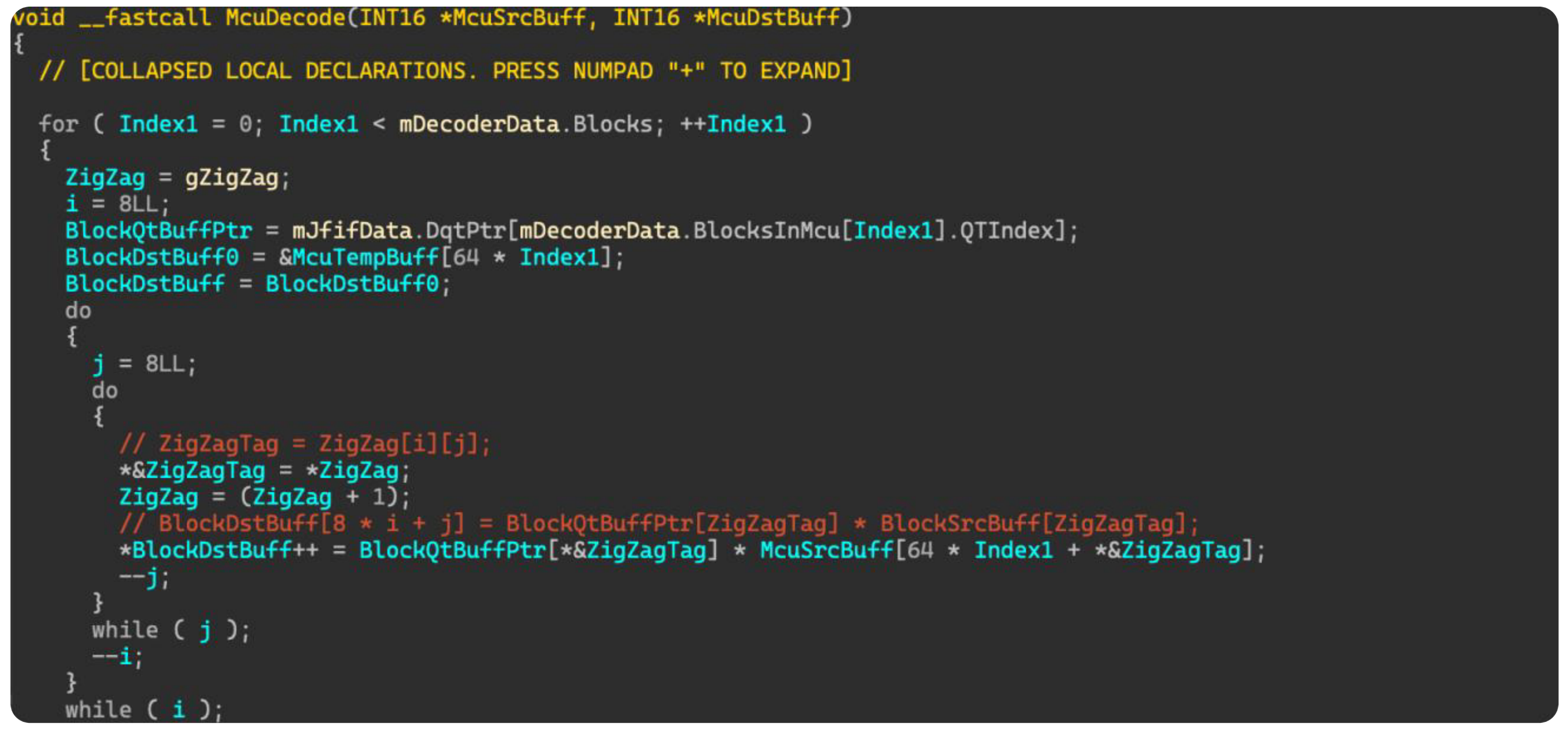


```
__int64 __fastcall InitJfifData(UINT8 *ImageData, UINTN ImageDataSize)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  SetMem(&mJfifData, 0xD00uLL, 0);
  if ( ImageDataSize < 2 || *ImageData != 0xFF || ImageData[1] != 0xD8 )
    return 1LL;
  mJfifData.SoiPtr = ImageData;
  gJpegImageDataEndPtr = &ImageData[ImageDataSize];
  for ( ImagePtr = ImageData + 2; ; ImagePtr += 2 )
  {
    while ( 1 )
    {
      while ( 1 )
      {
        while ( 1 )
        {
          // not checked against BRLY-LOGOFAIL-2023-008
          if ( ImagePtr >= &ImageData[ImageDataSize] )
            return 0LL;
          Remainder = (ImageDataSize + ImageData - ImagePtr);
          // not checked against BRLY-LOGOFAIL-2023-008
          if ( Remainder < 2 )
            return 0LL;
          if ( *ImagePtr == 0xFF )
          {
            Type = ImagePtr[1];
            if ( Type )
              break;
          }
          ++ImagePtr;
        }
        if ( Type > 0xC0u )
        {
          if ( ((Type + 62) & 0xFD) == 0 )
            break;
          if ( Type <= 0xCFu )
            return 6LL;
        }
        if ( Type <= 0xDAu )
          break;
        switch ( Type )
        {
          case 0xDB:
            if ( Remainder < 4 )
              return 3LL;
            v9 = ImagePtr + 4;
            v10 = (ImagePtr[3] - 2 + (ImagePtr[2] << 8)) / 65;
            if ( (Remainder - 4) < 65 * v10 )
              return 3LL;
```

BRLY-LOGOFAIL-2023-010 – **fixed** ✅

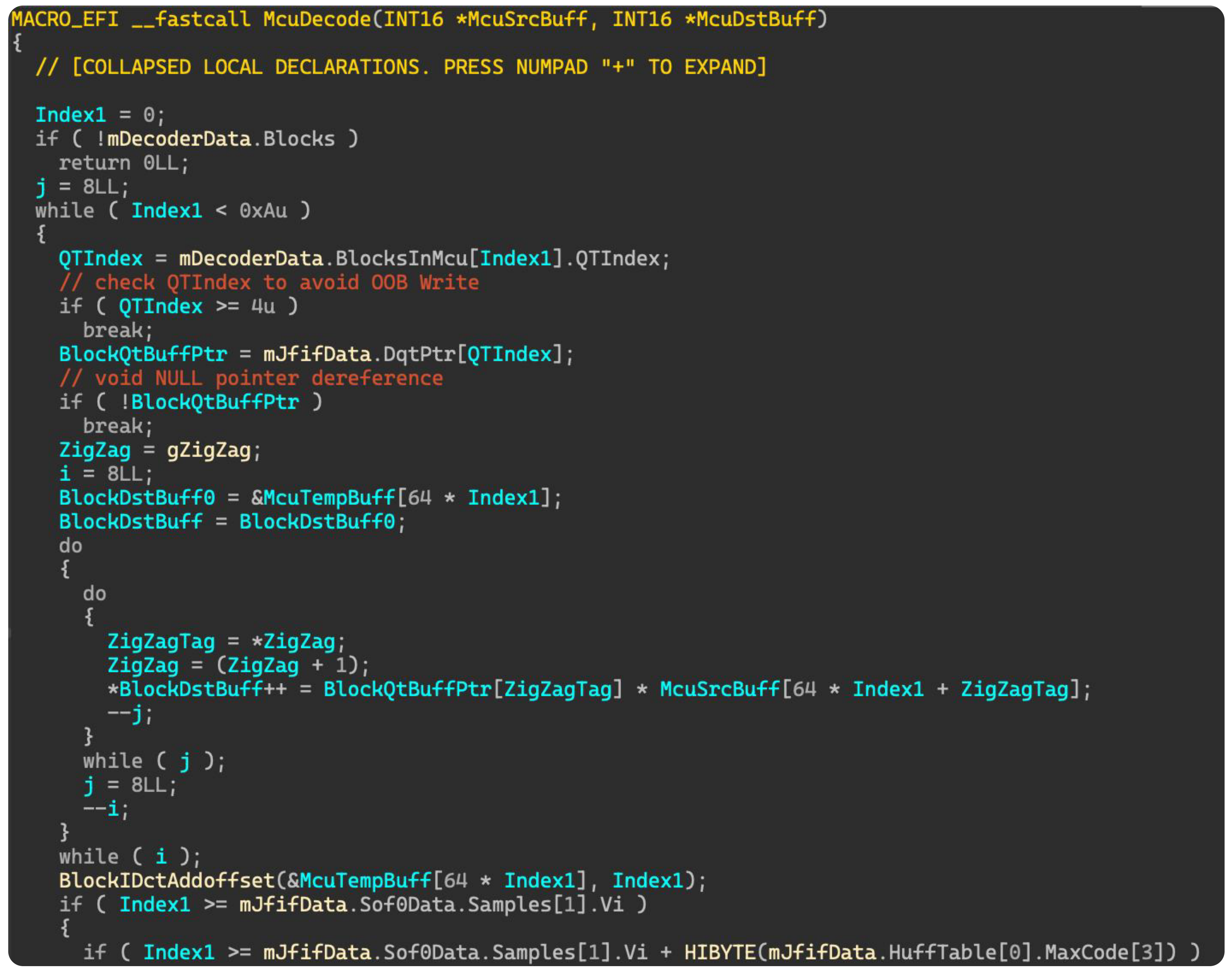


```
        mJfifData.App0Ptr = ImagePtr;
        break;
      case 254:
        mJfifData.ComPtr = ImagePtr;
        break;
    }
    goto _Next;
  }
  TempPtr = ImagePtr + 4;
  Index = 0;
  // DqtCount is user controllable and unchecked
  // When DqtCount < 4, mJfifData.DqtPtr will contain < 4 initialized elements (the rest will be zer
  // So, in McuDecode function BlockQtBuffPtr may become 0
  // after execution of the following code:
  //   BlockQtBuffPtr = mJfifData.DqtPtr[mDecoderData.BlocksInMcu[Index1].QTIndex];

  // Subsequent read by BlockQtBuffPtr will cause crash
  // * when zero page is unmapped, it will lead to DoS during the boot
  // * when zero page is mapped, it will lead to undefined behaviour

  // it's related to the following crashes:
  // * id:000007,sig:06,src:000000,time:1490927,execs:3595,op:havoc,rep:16
  // * id:000009,sig:06,src:000000+000232,time:2134620,execs:5401,op:splice,rep:16
  // * id:000010,sig:06,src:000000+000232,time:2470440,execs:6243,op:splice,rep:16
  // * id:000011,sig:06,src:000000+000202,time:2844494,execs:7619,op:splice,rep:16
  // * id:000013,sig:06,src:000000+000136,time:3151280,execs:9534,op:splice,rep:2
  // * id:000014,sig:06,src:000185+000147,time:3539427,execs:13963,op:splice,rep:16

  // BRLY-LOGOFAIL-2023-010: Unchecked DqtCount leads to null pointer dereference
  DqtCount = (ImagePtr[3] - 2 + (ImagePtr[2] << 8)) / 0x41;
  if ( !DqtCount )
  {
_Next:
    if ( ImagePtr[2] == 0xFF )
      Step = 2i64;
    else
      Step = (ImagePtr[2] << 8) + ImagePtr[3] + 2i64;
    goto LABEL_40;
  }
  while ( 1 )
  {
    DqtPtrIndex = *TempPtr & 0xF;
    if ( DqtPtrIndex > 3u || (*TempPtr & 0xF0) != 0 )
      return 3i64;                                  // EFI_JPEG_QUANTIZATIONTABLE_ERROR
    DqtPtrCurrentValue = TempPtr + 1;
    TempPtr += 65;
    ++Index;
    mJfifData.DqtPtr[DqtPtrIndex] = DqtPtrCurrentValue;
    if ( Index >= DqtCount )
```



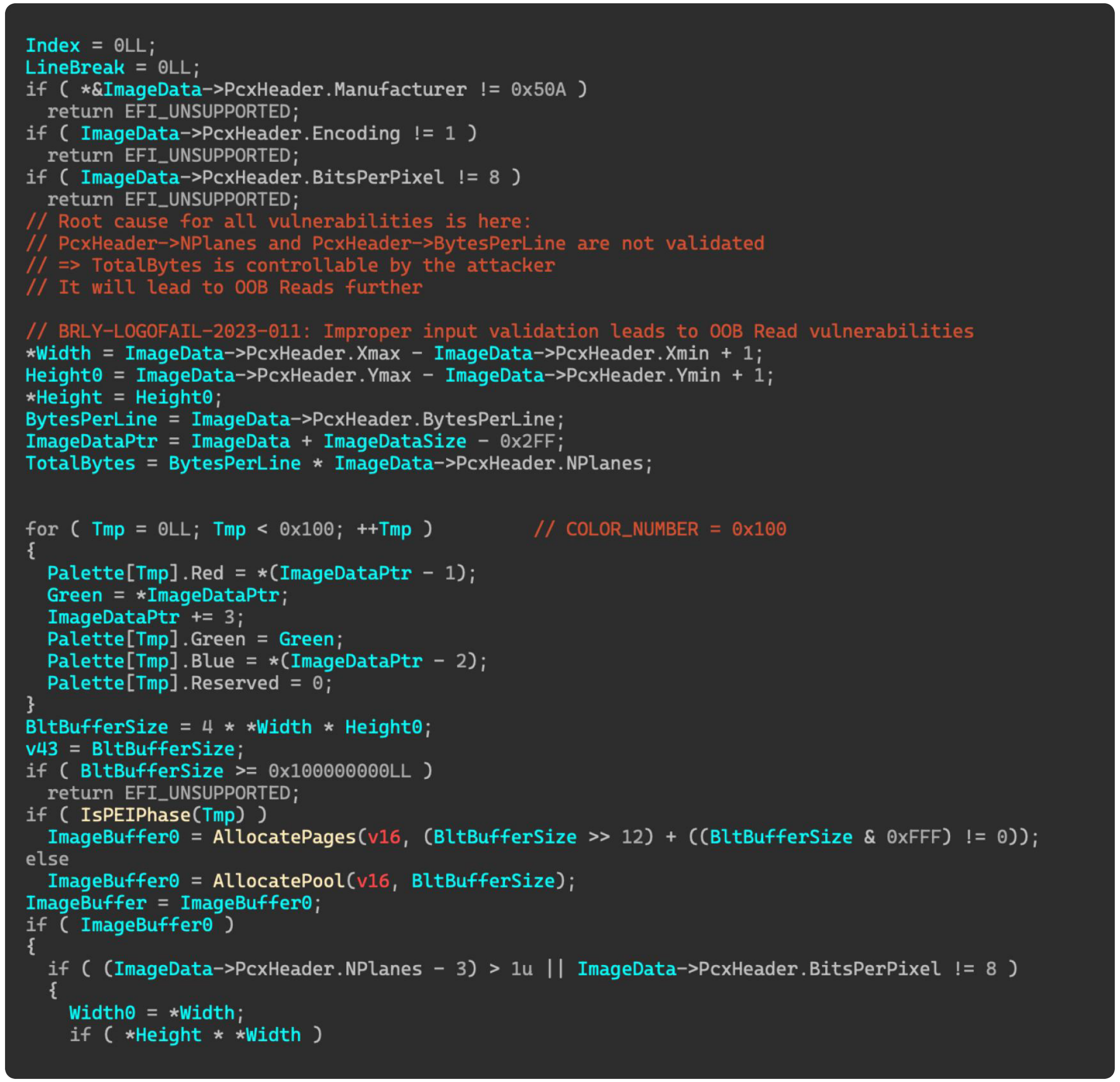
```
void __fastcall McuDecode(INT16 *McuSrcBuff, INT16 *McuDstBuff)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  for ( Index1 = 0; Index1 < mDecoderData.Blocks; ++Index1 )
  {
    ZigZag = gZigZag;
    i = 8LL;
    BlockQtBuffPtr = mJfifData.DqtPtr[mDecoderData.BlocksInMcu[Index1].QTIndex];
    BlockDstBuff0 = &McuTempBuff[64 * Index1];
    BlockDstBuff = BlockDstBuff0;
    do
    {
      j = 8LL;
      do
      {
        // ZigZagTag = ZigZag[i][j];
        *&ZigZagTag = *ZigZag;
        ZigZag = (ZigZag + 1);
        // BlockDstBuff[8 * i + j] = BlockQtBuffPtr[ZigZagTag] * BlockSrcBuff[ZigZagTag];
        *BlockDstBuff++ = BlockQtBuffPtr[*&ZigZagTag] * McuSrcBuff[64 * Index1 + *&ZigZagTag];
        --j;
      }
      while ( j );
      --i;
    }
    while ( i );
```

Fix: check the `BlockQtBuffPtr` **pointer in the** `McuDecode` **function before dereferencing**



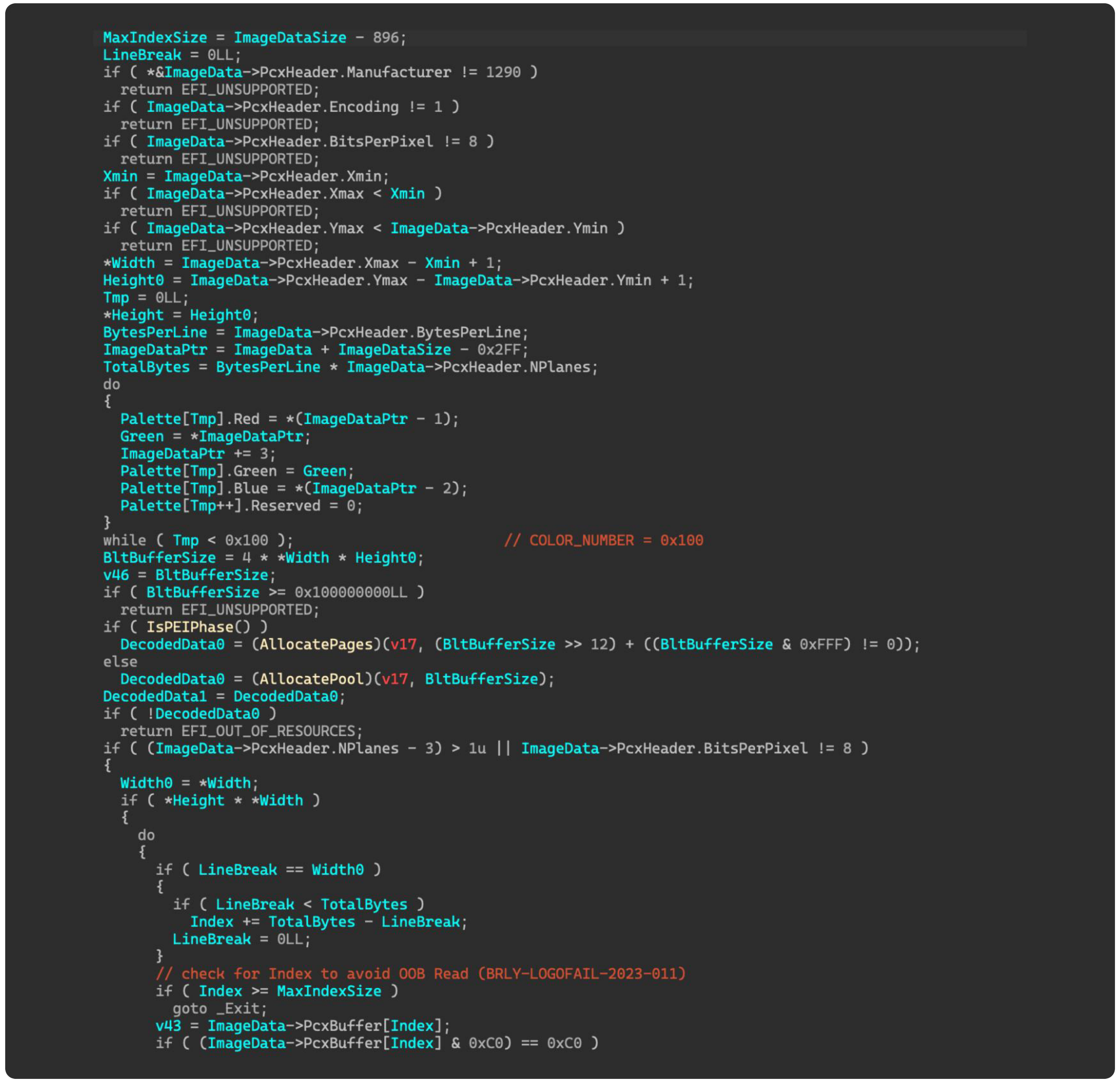
```
MACRO_EFI __fastcall McuDecode(INT16 *McuSrcBuff, INT16 *McuDstBuff)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  Index1 = 0;
  if ( !mDecoderData.Blocks )
    return 0LL;
  j = 8LL;
  while ( Index1 < 0xAu )
  {
    QTIndex = mDecoderData.BlocksInMcu[Index1].QTIndex;
    // check QTIndex to avoid OOB Write
    if ( QTIndex >= 4u )
      break;
    BlockQtBuffPtr = mJfifData.DqtPtr[QTIndex];
    // void NULL pointer dereference
    if ( !BlockQtBuffPtr )
      break;
    ZigZag = gZigZag;
    i = 8LL;
    BlockDstBuff0 = &McuTempBuff[64 * Index1];
    BlockDstBuff = BlockDstBuff0;
    do
    {
      do
      {
        ZigZagTag = *ZigZag;
        ZigZag = (ZigZag + 1);
        *BlockDstBuff++ = BlockQtBuffPtr[ZigZagTag] * McuSrcBuff[64 * Index1 + ZigZagTag];
        --j;
      }
      while ( j );
      j = 8LL;
      --i;
    }
    while ( i );
    BlockIDctAddoffset(&McuTempBuff[64 * Index1], Index1);
    if ( Index1 >= mJfifData.Sof0Data.Samples[1].Vi )
    {
      if ( Index1 >= mJfifData.Sof0Data.Samples[1].Vi + HIBYTE(mJfifData.HuffTable[0].MaxCode[3]) )
```

PcxDecoderDxe
BRLY-LOGOFAIL-2023-011 – fixed ✅



```
Index = 0LL;
LineBreak = 0LL;
if ( *&ImageData->PcxHeader.Manufacturer != 0x50A )
  return EFI_UNSUPPORTED;
if ( ImageData->PcxHeader.Encoding != 1 )
  return EFI_UNSUPPORTED;
if ( ImageData->PcxHeader.BitsPerPixel != 8 )
  return EFI_UNSUPPORTED;
// Root cause for all vulnerabilities is here:
// PcxHeader->NPlanes and PcxHeader->BytesPerLine are not validated
// => TotalBytes is controllable by the attacker
// It will lead to OOB Reads further

// BRLY-LOGOFAIL-2023-011: Improper input validation leads to OOB Read vulnerabilities
*Width = ImageData->PcxHeader.Xmax - ImageData->PcxHeader.Xmin + 1;
Height0 = ImageData->PcxHeader.Ymax - ImageData->PcxHeader.Ymin + 1;
*Height = Height0;
BytesPerLine = ImageData->PcxHeader.BytesPerLine;
ImageDataPtr = ImageData + ImageDataSize - 0x2FF;
TotalBytes = BytesPerLine * ImageData->PcxHeader.NPlanes;


for ( Tmp = 0LL; Tmp < 0x100; ++Tmp )          // COLOR_NUMBER = 0x100
{
  Palette[Tmp].Red = *(ImageDataPtr - 1);
  Green = *ImageDataPtr;
  ImageDataPtr += 3;
  Palette[Tmp].Green = Green;
  Palette[Tmp].Blue = *(ImageDataPtr - 2);
  Palette[Tmp].Reserved = 0;
}
BltBufferSize = 4 * *Width * Height0;
v43 = BltBufferSize;
if ( BltBufferSize >= 0x100000000LL )
  return EFI_UNSUPPORTED;
if ( IsPEIPhase(Tmp) )
  ImageBuffer0 = AllocatePages(v16, (BltBufferSize >> 12) + ((BltBufferSize & 0xFFF) != 0));
else
  ImageBuffer0 = AllocatePool(v16, BltBufferSize);
ImageBuffer = ImageBuffer0;
if ( ImageBuffer0 )
{
  if ( (ImageData->PcxHeader.NPlanes - 3) > 1u || ImageData->PcxHeader.BitsPerPixel != 8 )
  {
    Width0 = *Width;
    if ( *Height * *Width )
```

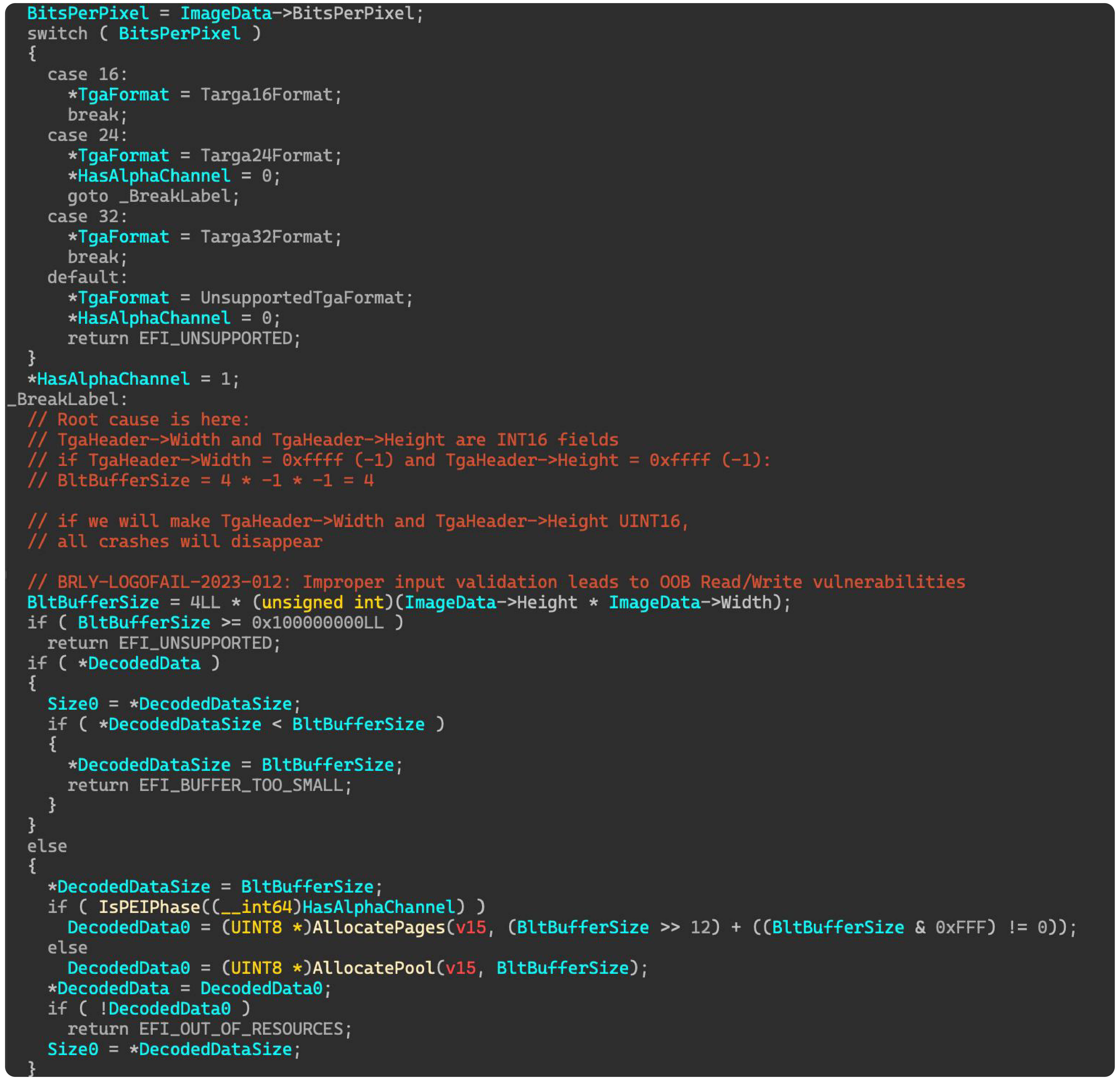
Fix: added check for index to avoid OOB Read



```
MaxIndexSize = ImageDataSize - 896;
LineBreak = 0LL;
if ( *&ImageData->PcxHeader.Manufacturer != 1290 )
  return EFI_UNSUPPORTED;
if ( ImageData->PcxHeader.Encoding != 1 )
  return EFI_UNSUPPORTED;
if ( ImageData->PcxHeader.BitsPerPixel != 8 )
  return EFI_UNSUPPORTED;
Xmin = ImageData->PcxHeader.Xmin;
if ( ImageData->PcxHeader.Xmax < Xmin )
  return EFI_UNSUPPORTED;
if ( ImageData->PcxHeader.Ymax < ImageData->PcxHeader.Ymin )
  return EFI_UNSUPPORTED;
*Width = ImageData->PcxHeader.Xmax - Xmin + 1;
Height0 = ImageData->PcxHeader.Ymax - ImageData->PcxHeader.Ymin + 1;
Tmp = 0LL;
*Height = Height0;
BytesPerLine = ImageData->PcxHeader.BytesPerLine;
ImageDataPtr = ImageData + ImageDataSize - 0x2FF;
TotalBytes = BytesPerLine * ImageData->PcxHeader.NPlanes;
do
{
  Palette[Tmp].Red = *(ImageDataPtr - 1);
  Green = *ImageDataPtr;
  ImageDataPtr += 3;
  Palette[Tmp].Green = Green;
  Palette[Tmp].Blue = *(ImageDataPtr - 2);
  Palette[Tmp++].Reserved = 0;
}
while ( Tmp < 0x100 );                          // COLOR_NUMBER = 0x100
BltBufferSize = 4 * *Width * Height0;
v46 = BltBufferSize;
if ( BltBufferSize >= 0x100000000LL )
  return EFI_UNSUPPORTED;
if ( IsPEIPhase() )
  DecodedData0 = (AllocatePages)(v17, (BltBufferSize >> 12) + ((BltBufferSize & 0xFFF) != 0));
else
  DecodedData0 = (AllocatePool)(v17, BltBufferSize);
DecodedData1 = DecodedData0;
if ( !DecodedData0 )
  return EFI_OUT_OF_RESOURCES;
if ( (ImageData->PcxHeader.NPlanes - 3) > 1u || ImageData->PcxHeader.BitsPerPixel != 8 )
{
  Width0 = *Width;
  if ( *Height * *Width )
  {
    do
    {
      if ( LineBreak == Width0 )
      {
        if ( LineBreak < TotalBytes )
          Index += TotalBytes - LineBreak;
        LineBreak = 0LL;
      }
      // check for Index to avoid OOB Read (BRLY-LOGOFAIL-2023-011)
      if ( Index >= MaxIndexSize )
        goto _Exit;
      v43 = ImageData->PcxBuffer[Index];
      if ( (ImageData->PcxBuffer[Index] & 0xC0) == 0xC0 )
```

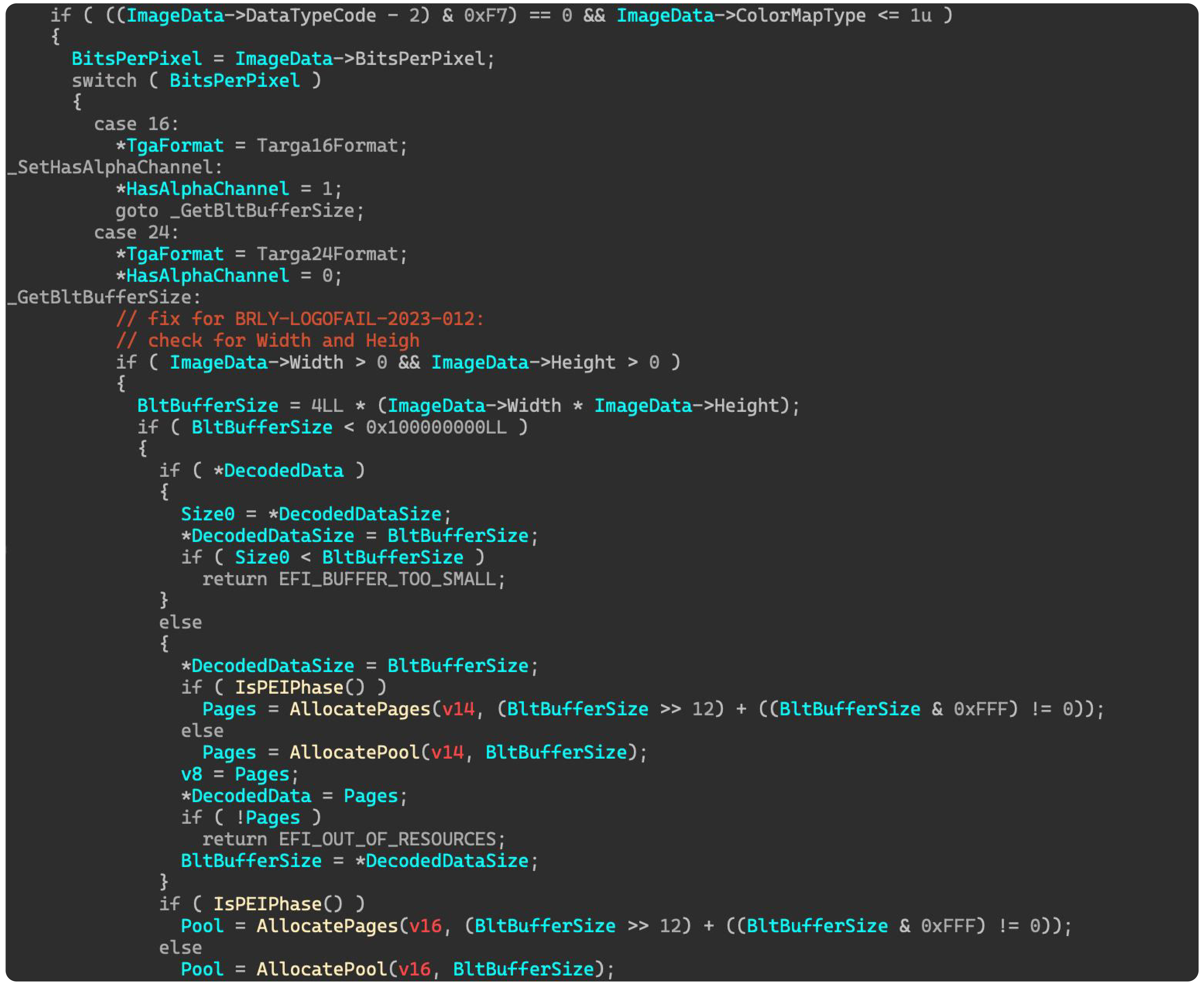
TgaDecoderDxe

BRLY-LOGOFAIL-2023-012 – **fixed** ✅



```
  BitsPerPixel = ImageData->BitsPerPixel;
  switch ( BitsPerPixel )
  {
    case 16:
      *TgaFormat = Targa16Format;
      break;
    case 24:
      *TgaFormat = Targa24Format;
      *HasAlphaChannel = 0;
      goto _BreakLabel;
    case 32:
      *TgaFormat = Targa32Format;
      break;
    default:
      *TgaFormat = UnsupportedTgaFormat;
      *HasAlphaChannel = 0;
      return EFI_UNSUPPORTED;
  }
  *HasAlphaChannel = 1;
_BreakLabel:
  // Root cause is here:
  // TgaHeader->Width and TgaHeader->Height are INT16 fields
  // if TgaHeader->Width = 0xffff (-1) and TgaHeader->Height = 0xffff (-1):
  // BltBufferSize = 4 * -1 * -1 = 4

  // if we will make TgaHeader->Width and TgaHeader->Height UINT16,
  // all crashes will disappear

  // BRLY-LOGOFAIL-2023-012: Improper input validation leads to OOB Read/Write vulnerabilities
  BltBufferSize = 4LL * (unsigned int)(ImageData->Height * ImageData->Width);
  if ( BltBufferSize >= 0x100000000LL )
    return EFI_UNSUPPORTED;
  if ( *DecodedData )
  {
    Size0 = *DecodedDataSize;
    if ( *DecodedDataSize < BltBufferSize )
    {
      *DecodedDataSize = BltBufferSize;
      return EFI_BUFFER_TOO_SMALL;
    }
  }
  else
  {
    *DecodedDataSize = BltBufferSize;
    if ( IsPEIPhase((__int64)HasAlphaChannel) )
      DecodedData0 = (UINT8 *)AllocatePages(v15, (BltBufferSize >> 12) + ((BltBufferSize & 0xFFF) != 0));
    else
      DecodedData0 = (UINT8 *)AllocatePool(v15, BltBufferSize);
    *DecodedData = DecodedData0;
    if ( !DecodedData0 )
      return EFI_OUT_OF_RESOURCES;
    Size0 = *DecodedDataSize;
  }
```

Fix: added check for TgaHeader→Width **and** TgaHeader→Height



```
    if ( ((ImageData->DataTypeCode - 2) & 0xF7) == 0 && ImageData->ColorMapType <= 1u )
    {
      BitsPerPixel = ImageData->BitsPerPixel;
      switch ( BitsPerPixel )
      {
        case 16:
          *TgaFormat = Targa16Format;
_SetHasAlphaChannel:
          *HasAlphaChannel = 1;
          goto _GetBltBufferSize;
        case 24:
          *TgaFormat = Targa24Format;
          *HasAlphaChannel = 0;
_GetBltBufferSize:
          // fix for BRLY-LOGOFAIL-2023-012:
          // check for Width and Heigh
          if ( ImageData->Width > 0 && ImageData->Height > 0 )
          {
            BltBufferSize = 4LL * (ImageData->Width * ImageData->Height);
            if ( BltBufferSize < 0x100000000LL )
            {
              if ( *DecodedData )
              {
                Size0 = *DecodedDataSize;
                *DecodedDataSize = BltBufferSize;
                if ( Size0 < BltBufferSize )
                  return EFI_BUFFER_TOO_SMALL;
              }
              else
              {
                *DecodedDataSize = BltBufferSize;
                if ( IsPEIPhase() )
                  Pages = AllocatePages(v14, (BltBufferSize >> 12) + ((BltBufferSize & 0xFFF) != 0));
                else
                  Pages = AllocatePool(v14, BltBufferSize);
                v8 = Pages;
                *DecodedData = Pages;
                if ( !Pages )
                  return EFI_OUT_OF_RESOURCES;
                BltBufferSize = *DecodedDataSize;
              }
              if ( IsPEIPhase() )
                Pool = AllocatePages(v16, (BltBufferSize >> 12) + ((BltBufferSize & 0xFFF) != 0));
              else
                Pool = AllocatePool(v16, BltBufferSize);
```

Patch Breakdown and Incomplete Fixes

AMI

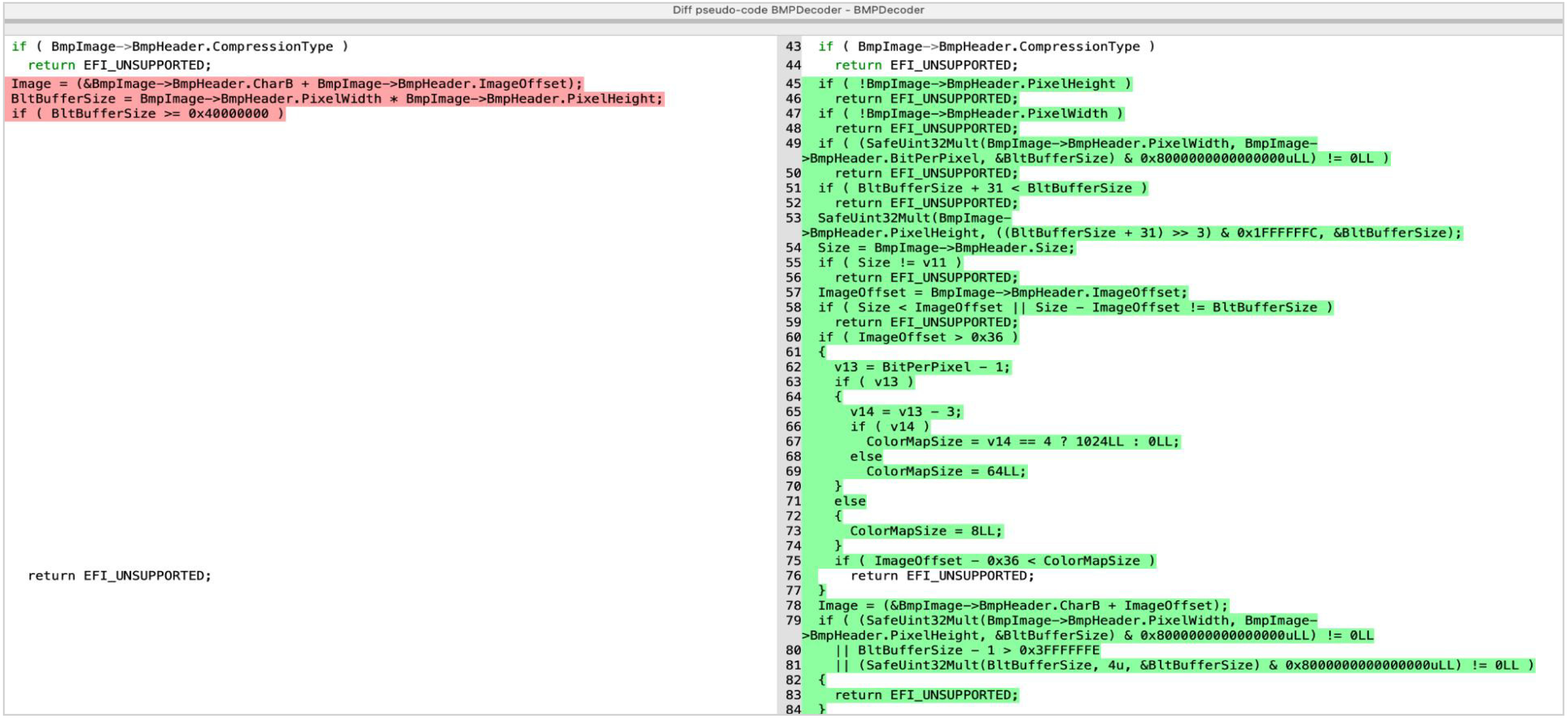
Summary of the fixes

- BMP parser is fixed by switching to BMP parser from EDK2
- All other parsers (GIF, JPEG, PNG) were correctly fixed

BMP decoder

BRLY-LOGOFAIL-2023-013 – **fixed** ✅

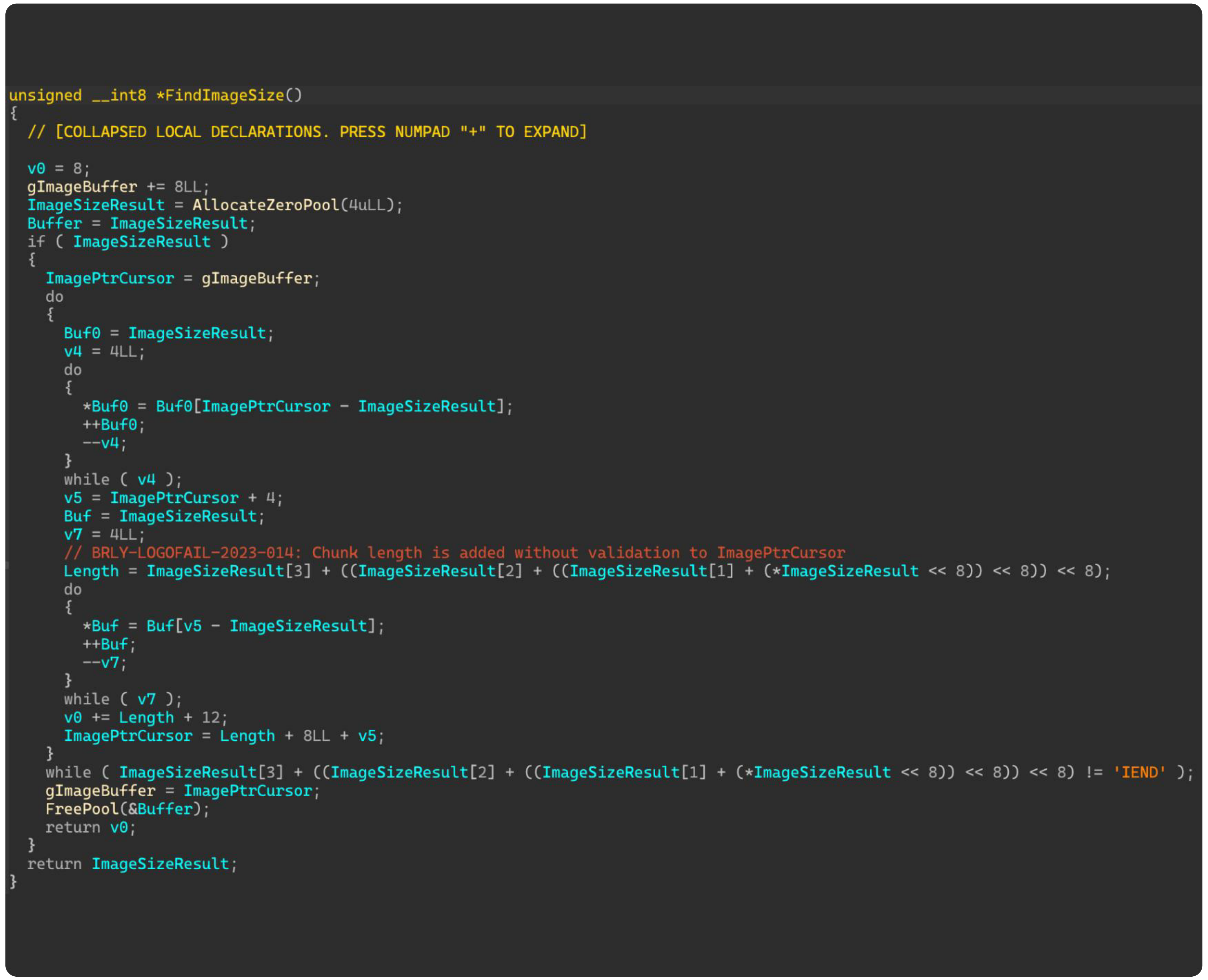
BMP parser is fixed by switching to BMP **parser from** EDK2



```
if ( BmpImage->BmpHeader.CompressionType )
  return EFI_UNSUPPORTED;
Image = (&BmpImage->BmpHeader.CharB + BmpImage->BmpHeader.ImageOffset);
BltBufferSize = BmpImage->BmpHeader.PixelWidth * BmpImage->BmpHeader.PixelHeight;
if ( BltBufferSize >= 0x40000000 )
  return EFI_UNSUPPORTED;
```

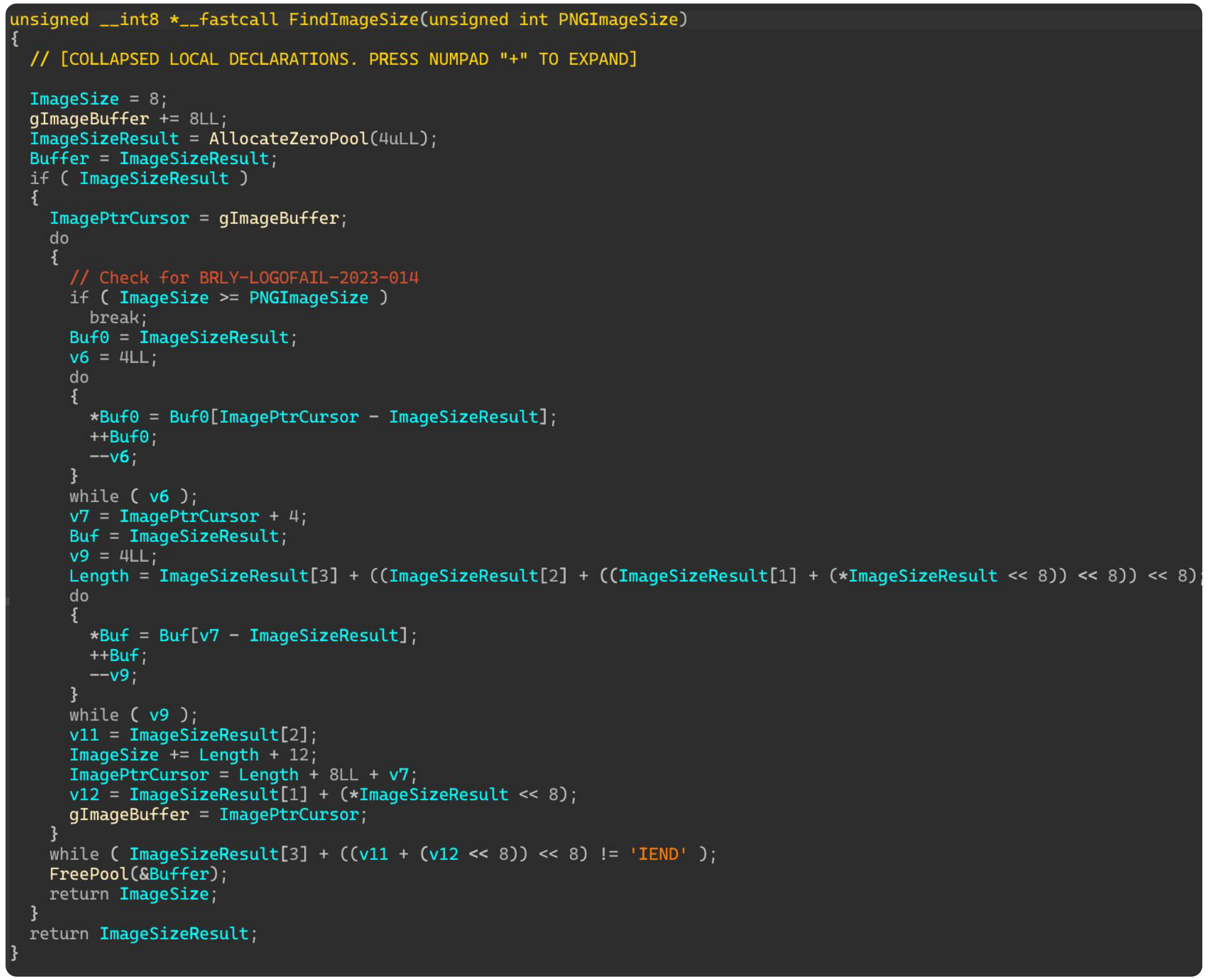
```
if ( BmpImage->BmpHeader.CompressionType )
  return EFI_UNSUPPORTED;
if ( !BmpImage->BmpHeader.PixelHeight )
  return EFI_UNSUPPORTED;
if ( !BmpImage->BmpHeader.PixelWidth )
  return EFI_UNSUPPORTED;
if ( (SafeUint32Mult(BmpImage->BmpHeader.PixelWidth, BmpImage-
>BmpHeader.BitPerPixel, &BltBufferSize) & 0x8000000000000000uLL) != 0LL )
  return EFI_UNSUPPORTED;
if ( BltBufferSize + 31 < BltBufferSize )
  return EFI_UNSUPPORTED;
SafeUint32Mult(BmpImage-
>BmpHeader.PixelHeight, ((BltBufferSize + 31) >> 3) & 0x1FFFFFFC, &BltBufferSize);
Size = BmpImage->BmpHeader.Size;
if ( Size != v11 )
  return EFI_UNSUPPORTED;
ImageOffset = BmpImage->BmpHeader.ImageOffset;
if ( Size < ImageOffset || Size - ImageOffset != BltBufferSize )
  return EFI_UNSUPPORTED;
if ( ImageOffset > 0x36 )
{
  v13 = BitPerPixel - 1;
  if ( v13 )
  {
    v14 = v13 - 3;
    if ( v14 )
      ColorMapSize = v14 == 4 ? 1024LL : 0LL;
    else
      ColorMapSize = 64LL;
  }
  else
  {
    ColorMapSize = 8LL;
  }
  if ( ImageOffset - 0x36 < ColorMapSize )
    return EFI_UNSUPPORTED;
}
Image = (&BmpImage->BmpHeader.CharB + ImageOffset);
if ( (SafeUint32Mult(BmpImage->BmpHeader.PixelWidth, BmpImage-
>BmpHeader.PixelHeight, &BltBufferSize) & 0x8000000000000000uLL) != 0LL
  || BltBufferSize - 1 > 0x3FFFFFFE
  || (SafeUint32Mult(BltBufferSize, 4u, &BltBufferSize) & 0x8000000000000000uLL) != 0LL )
{
  return EFI_UNSUPPORTED;
}
```

PNG Decoder
BRLY-LOGOFAIL-2023-014 – **fixed** ✅



```
unsigned __int8 *FindImageSize()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  v0 = 8;
  gImageBuffer += 8LL;
  ImageSizeResult = AllocateZeroPool(4uLL);
  Buffer = ImageSizeResult;
  if ( ImageSizeResult )
  {
    ImagePtrCursor = gImageBuffer;
    do
    {
      Buf0 = ImageSizeResult;
      v4 = 4LL;
      do
      {
        *Buf0 = Buf0[ImagePtrCursor - ImageSizeResult];
        ++Buf0;
        --v4;
      }
      while ( v4 );
      v5 = ImagePtrCursor + 4;
      Buf = ImageSizeResult;
      v7 = 4LL;
      // BRLY-LOGOFAIL-2023-014: Chunk length is added without validation to ImagePtrCursor
      Length = ImageSizeResult[3] + ((ImageSizeResult[2] + ((ImageSizeResult[1] + (*ImageSizeResult << 8)) << 8)) << 8);
      do
      {
        *Buf = Buf[v5 - ImageSizeResult];
        ++Buf;
        --v7;
      }
      while ( v7 );
      v0 += Length + 12;
      ImagePtrCursor = Length + 8LL + v5;
    }
    while ( ImageSizeResult[3] + ((ImageSizeResult[2] + ((ImageSizeResult[1] + (*ImageSizeResult << 8)) << 8)) << 8) != 'IEND' );
    gImageBuffer = ImagePtrCursor;
    FreePool(&Buffer);
    return v0;
  }
  return ImageSizeResult;
}
```

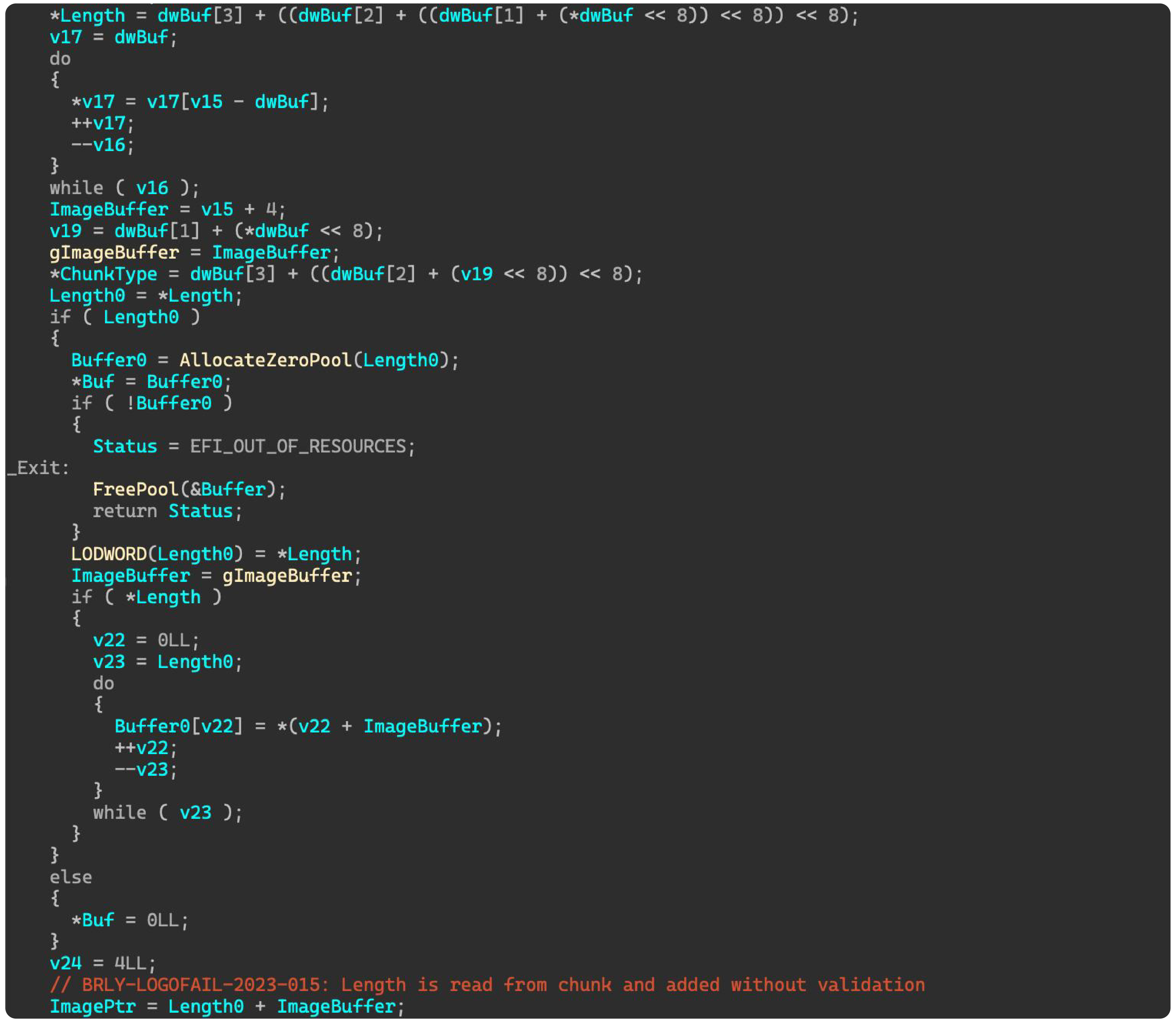
Fix: add validation for `ImageSize`



```
unsigned __int8 *__fastcall FindImageSize(unsigned int PNGImageSize)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  ImageSize = 8;
  gImageBuffer += 8LL;
  ImageSizeResult = AllocateZeroPool(4uLL);
  Buffer = ImageSizeResult;
  if ( ImageSizeResult )
  {
    ImagePtrCursor = gImageBuffer;
    do
    {
      // Check for BRLY-LOGOFAIL-2023-014
      if ( ImageSize >= PNGImageSize )
        break;
      Buf0 = ImageSizeResult;
      v6 = 4LL;
      do
      {
        *Buf0 = Buf0[ImagePtrCursor - ImageSizeResult];
        ++Buf0;
        --v6;
      }
      while ( v6 );
      v7 = ImagePtrCursor + 4;
      Buf = ImageSizeResult;
      v9 = 4LL;
      Length = ImageSizeResult[3] + ((ImageSizeResult[2] + ((ImageSizeResult[1] + (*ImageSizeResult << 8)) << 8)) << 8)
      do
      {
        *Buf = Buf[v7 - ImageSizeResult];
        ++Buf;
        --v9;
      }
      while ( v9 );
      v11 = ImageSizeResult[2];
      ImageSize += Length + 12;
      ImagePtrCursor = Length + 8LL + v7;
      v12 = ImageSizeResult[1] + (*ImageSizeResult << 8);
      gImageBuffer = ImagePtrCursor;
    }
    while ( ImageSizeResult[3] + ((v11 + (v12 << 8)) << 8) != 'IEND' );
    FreePool(&Buffer);
    return ImageSize;
  }
  return ImageSizeResult;
}
```

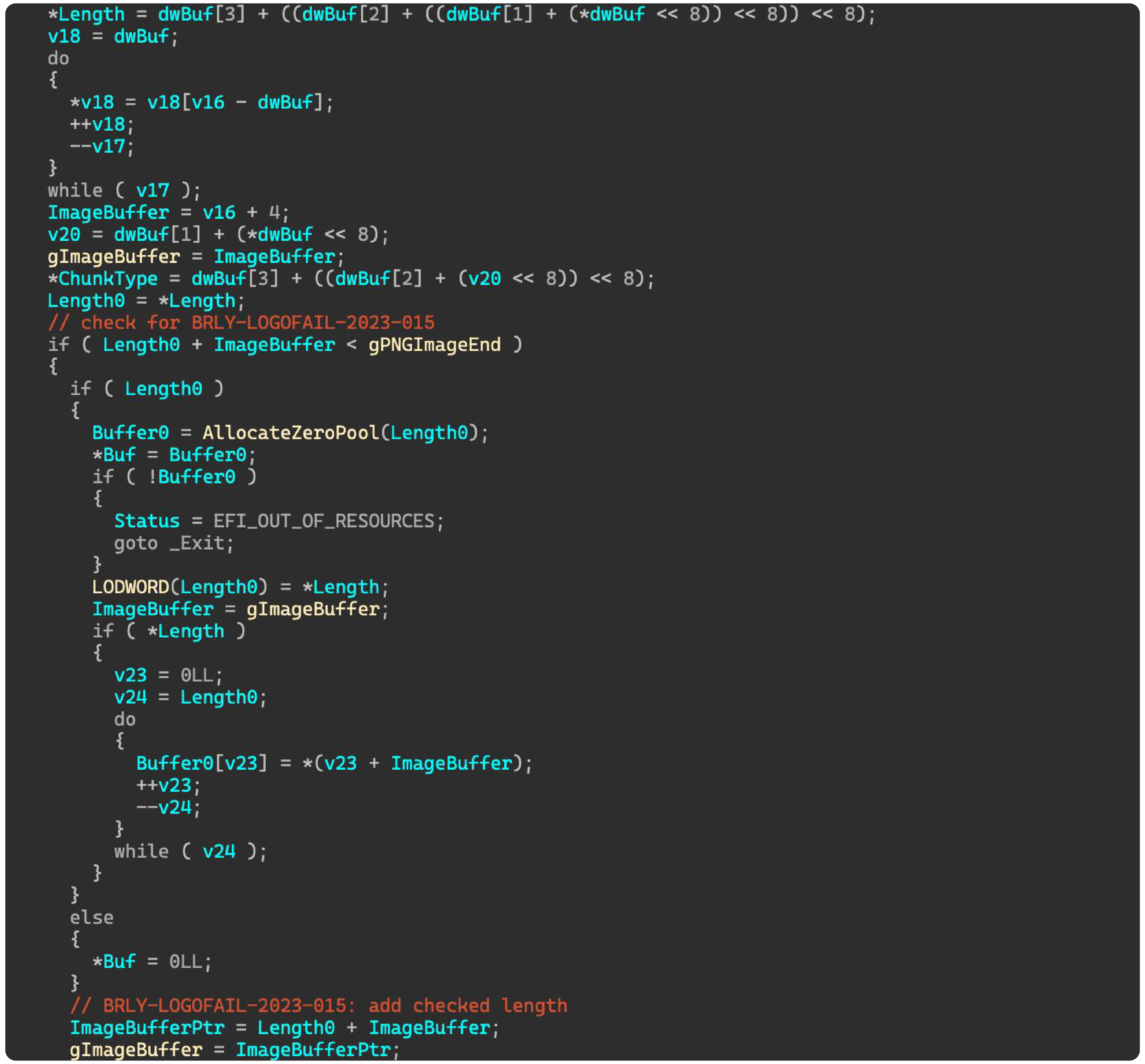
BRLY-LOGOFAIL-2023-015 – **fixed** ✅

In ReadChunk() **function:**



```
    *Length = dwBuf[3] + ((dwBuf[2] + ((dwBuf[1] + (*dwBuf << 8)) << 8)) << 8);
    v17 = dwBuf;
    do
    {
      *v17 = v17[v15 - dwBuf];
      ++v17;
      --v16;
    }
    while ( v16 );
    ImageBuffer = v15 + 4;
    v19 = dwBuf[1] + (*dwBuf << 8);
    gImageBuffer = ImageBuffer;
    *ChunkType = dwBuf[3] + ((dwBuf[2] + (v19 << 8)) << 8);
    Length0 = *Length;
    if ( Length0 )
    {
      Buffer0 = AllocateZeroPool(Length0);
      *Buf = Buffer0;
      if ( !Buffer0 )
      {
        Status = EFI_OUT_OF_RESOURCES;
_Exit:
        FreePool(&Buffer);
        return Status;
      }
      LODWORD(Length0) = *Length;
      ImageBuffer = gImageBuffer;
      if ( *Length )
      {
        v22 = 0LL;
        v23 = Length0;
        do
        {
          Buffer0[v22] = *(v22 + ImageBuffer);
          ++v22;
          --v23;
        }
        while ( v23 );
      }
    }
    else
    {
      *Buf = 0LL;
    }
    v24 = 4LL;
    // BRLY-LOGOFAIL-2023-015: Length is read from chunk and added without validation
    ImagePtr = Length0 + ImageBuffer;
```

Fix: **add check for** Length



```
    *Length = dwBuf[3] + ((dwBuf[2] + ((dwBuf[1] + (*dwBuf << 8)) << 8)) << 8);
    v18 = dwBuf;
    do
    {
      *v18 = v18[v16 - dwBuf];
      ++v18;
      --v17;
    }
    while ( v17 );
    ImageBuffer = v16 + 4;
    v20 = dwBuf[1] + (*dwBuf << 8);
    gImageBuffer = ImageBuffer;
    *ChunkType = dwBuf[3] + ((dwBuf[2] + (v20 << 8)) << 8);
    Length0 = *Length;
    // check for BRLY-LOGOFAIL-2023-015
    if ( Length0 + ImageBuffer < gPNGImageEnd )
    {
      if ( Length0 )
      {
        Buffer0 = AllocateZeroPool(Length0);
        *Buf = Buffer0;
        if ( !Buffer0 )
        {
          Status = EFI_OUT_OF_RESOURCES;
          goto _Exit;
        }
        LODWORD(Length0) = *Length;
        ImageBuffer = gImageBuffer;
        if ( *Length )
        {
          v23 = 0LL;
          v24 = Length0;
          do
          {
            Buffer0[v23] = *(v23 + ImageBuffer);
            ++v23;
            --v24;
          }
          while ( v24 );
        }
      }
      else
      {
        *Buf = 0LL;
      }
      // BRLY-LOGOFAIL-2023-015: add checked length
      ImageBufferPtr = Length0 + ImageBuffer;
      gImageBuffer = ImageBufferPtr;
```

BRLY-LOGOFAIL-2023-016 – **fixed** ✅

In the PrepareOutput() **function:**

```
      if ( gGlobalInfo.hdr.bitDepth != 2 )
      {
        if ( gGlobalInfo.hdr.bitDepth != 4 )
        {
          if ( gGlobalInfo.hdr.bitDepth != 8 )
            goto LABEL_48;
          goto LABEL_34;
        }
        goto LABEL_45;
      }
      goto LABEL_46;
    }
    goto LABEL_47;
  }
  if ( gGlobalInfo.hdr.bitDepth == 8 )
  {
    PngWidth = 3 * gGlobalInfo.hdr.width;
  }
  else if ( gGlobalInfo.hdr.bitDepth == 16 )
  {
    PngWidth = 6 * gGlobalInfo.hdr.width;
  }
LABEL_48:
  // BRLY-LOGOFAIL-2023-016: Integer overflow on the argument of EfiLibAllocateZeroPool
  OutputBuffer = AllocateZeroPool(2 * PngWidth);
  gGlobalInfo.OutputBuffer = OutputBuffer;
  if ( !OutputBuffer )
  {
    FreePool(&gGlobalInfo.rgba);
    gGlobalInfo.rgba = 0i64;
    return EFI_UNSUPPORTED;
  }
  gGlobalInfo.FirstHalf = OutputBuffer;
  gGlobalInfo.SecondHalf = &OutputBuffer[PngWidth];
  return 0i64;
```

Fix: **Indirectly patched by the fix for** BRLY-LOGOFAIL-2023-018**, since maximum value for** PngWidth **will be** 8 * gGlobalInfo.hdr.width **and** gGlobalInfo.hdr.width **is checked**

```
height = gGlobalInfo.hdr.height;
width = gGlobalInfo.hdr.width;
*&gGlobalInfo.wid = *&gGlobalInfo.hdr.width;
if ( gGlobalInfo.autoDeleteRgbaBuffer == 1 && gGlobalInfo.rgba )
{
  FreePool(&gGlobalInfo.rgba);
  gGlobalInfo.rgba = 0LL;
  height = gGlobalInfo.hei;
  width = gGlobalInfo.wid;
}
if ( !width )
  return EFI_UNSUPPORTED;
if ( !height )
  return EFI_UNSUPPORTED;
NumOfPixels = width * height;
// check for BRLY-LOGOFAIL-2023-018
if ( NumOfPixels >= 0x40000000 )
  return EFI_UNSUPPORTED;
gGlobalInfo.rgba = AllocateZeroPool(4 * NumOfPixels);
if ( !gGlobalInfo.rgba )
  return EFI_UNSUPPORTED;
gGlobalInfo.idx = -1;
gGlobalInfo.y = 0;
gGlobalInfo.filter = 0;
gGlobalInfo.inLineCount = 0;
gGlobalInfo.inPixelCount = 0;
gGlobalInfo.index = 0;
```

BRLY-LOGOFAIL-2023-017 – **fixed** ✅

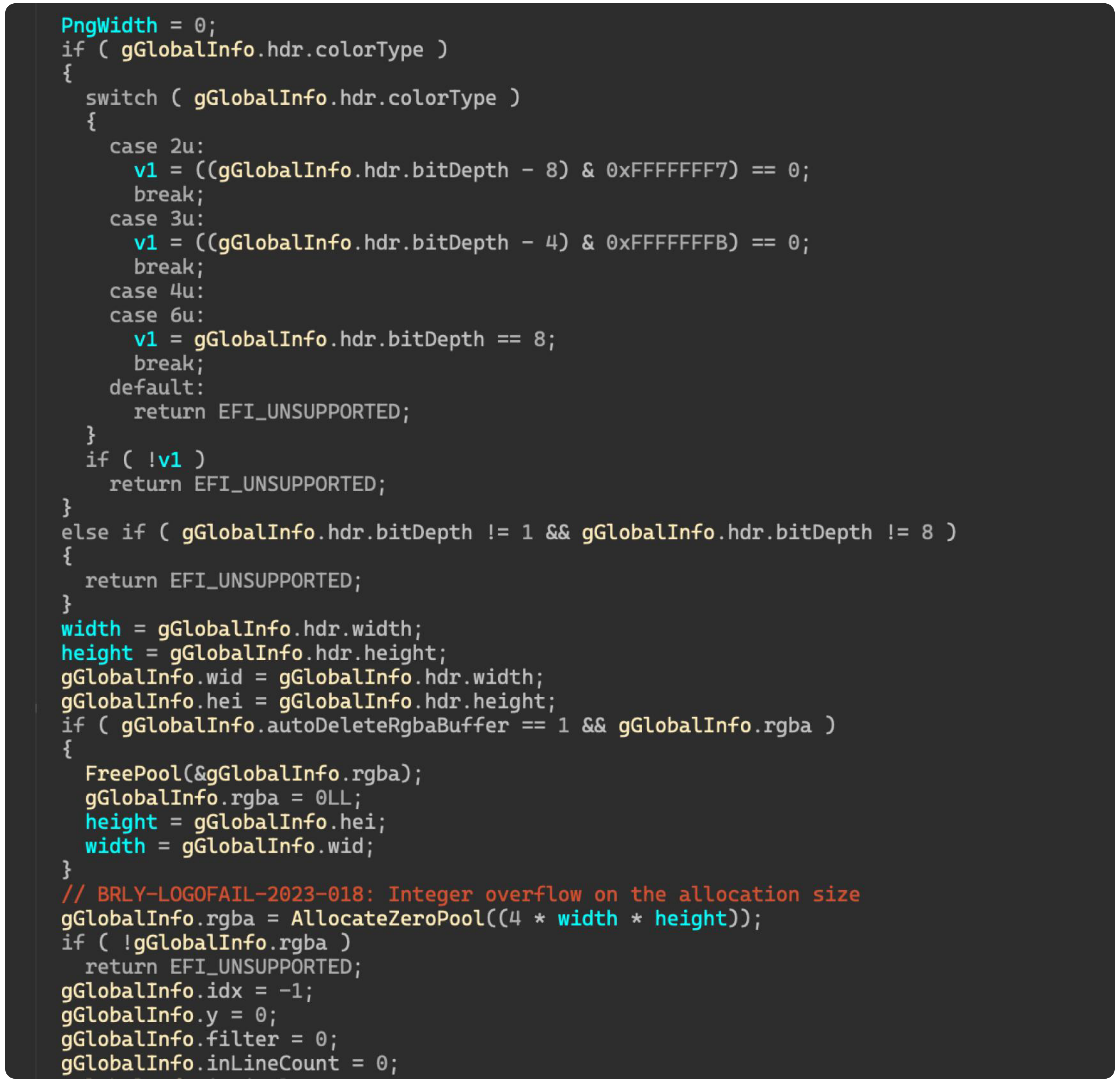
```
switch ( Value )
{
  case 0x10u:
    for ( i = GetNextMultiBit(Dat, byte_ptr, bit_ptr, 2u) + 3; i; --i )
    {
      v39 = v15;
      v40 = hLengthBuf[v15++ - 1];
      hLengthBuf[v39] = v40;
    }
    break;
  case 0x11u:
    for ( j = GetNextMultiBit(Dat, byte_ptr, bit_ptr, 3u) + 3; j; --j )
    {
      v42 = v15++;
      hLengthBuf[v42] = 0;
    }
    break;
  case 0x12u:
    for ( k = GetNextMultiBit(Dat, byte_ptr, bit_ptr, 7u) + 11; k; --k )
    {
      v44 = v15++;
      // BRLY-LOGOFAIL-2023-017: v15 could grow bigger than 322, thus writing OOB on the heap
      hLengthBuf[v44] = 0;
    }
    break;
}
```

Fix: **add check for array index**

```
switch ( Value )
{
  case 16u:
    for ( i = GetNextMultiBit(Dat, byte_ptr, bit_ptr, 2u) + 3; i; --i )
    {
      // Check for BRLY-LOGOFAIL-2023-017
      if ( n >= 322 )
        break;
      Index0 = n;
      v43 = hLengthBuf[n++ - 1];
      hLengthBuf[Index0] = v43;
    }
    break;
  case 17u:
    for ( j = GetNextMultiBit(Dat, byte_ptr, bit_ptr, 3u) + 3; j; --j )
    {
      // Check for BRLY-LOGOFAIL-2023-017
      if ( n >= 322 )
        break;
      Index1 = n++;
      hLengthBuf[Index1] = 0;
    }
    break;
  case 18u:
    for ( k = GetNextMultiBit(Dat, byte_ptr, bit_ptr, 7u) + 11; k; --k )
    {
      // Check for BRLY-LOGOFAIL-2023-017
      if ( n >= 322 )
        break;
      Index2 = n++;
      hLengthBuf[Index2] = 0;
    }
    break;
}
```

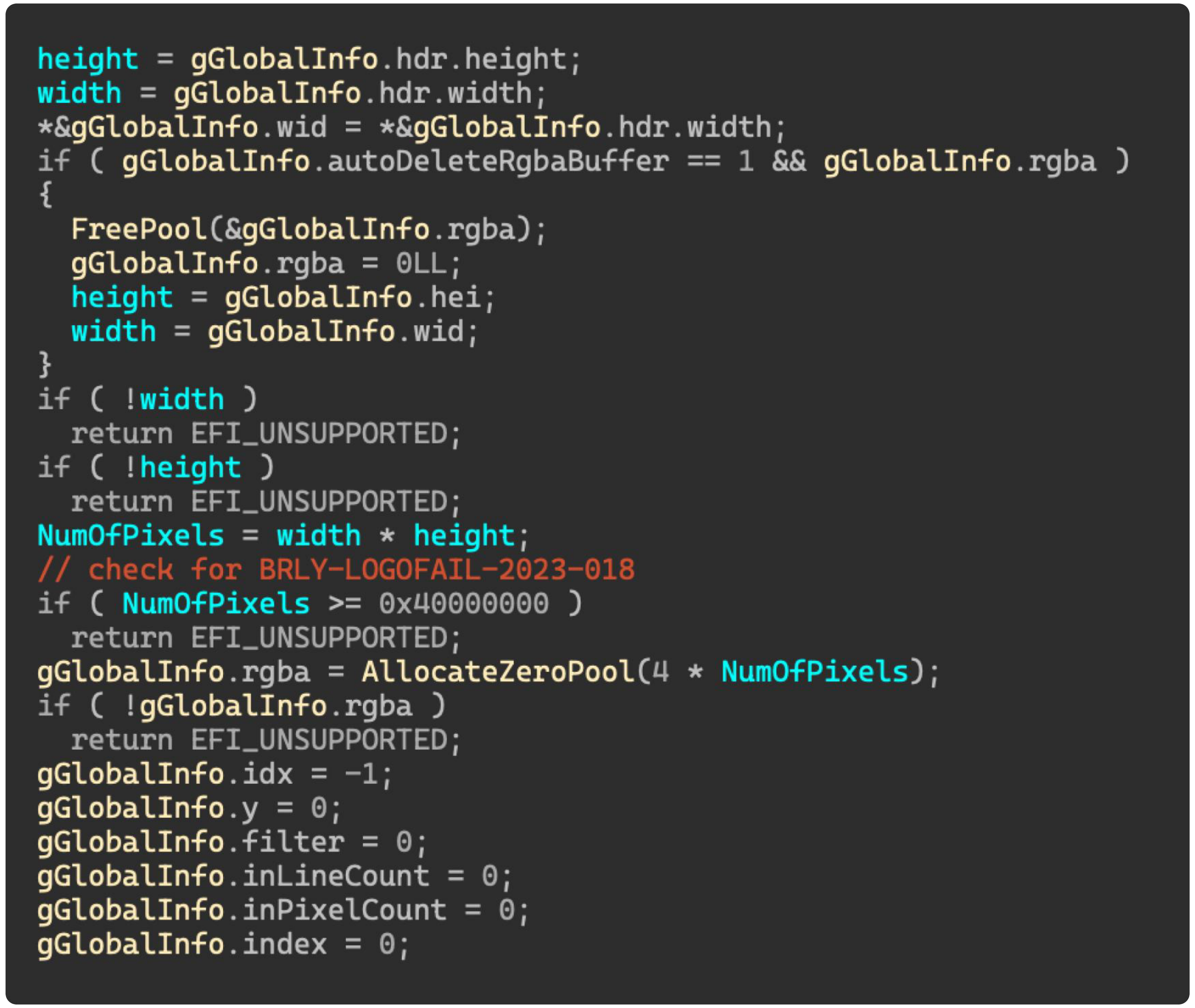
BRLY-LOGOFAIL-2023-018 – **fixed** ✅

In the `PrepareOutput()` **function:**



```
PngWidth = 0;
if ( gGlobalInfo.hdr.colorType )
{
  switch ( gGlobalInfo.hdr.colorType )
  {
    case 2u:
      v1 = ((gGlobalInfo.hdr.bitDepth - 8) & 0xFFFFFFF7) == 0;
      break;
    case 3u:
      v1 = ((gGlobalInfo.hdr.bitDepth - 4) & 0xFFFFFFFB) == 0;
      break;
    case 4u:
    case 6u:
      v1 = gGlobalInfo.hdr.bitDepth == 8;
      break;
    default:
      return EFI_UNSUPPORTED;
  }
  if ( !v1 )
    return EFI_UNSUPPORTED;
}
else if ( gGlobalInfo.hdr.bitDepth != 1 && gGlobalInfo.hdr.bitDepth != 8 )
{
  return EFI_UNSUPPORTED;
}
width = gGlobalInfo.hdr.width;
height = gGlobalInfo.hdr.height;
gGlobalInfo.wid = gGlobalInfo.hdr.width;
gGlobalInfo.hei = gGlobalInfo.hdr.height;
if ( gGlobalInfo.autoDeleteRgbaBuffer == 1 && gGlobalInfo.rgba )
{
  FreePool(&gGlobalInfo.rgba);
  gGlobalInfo.rgba = 0LL;
  height = gGlobalInfo.hei;
  width = gGlobalInfo.wid;
}
// BRLY-LOGOFAIL-2023-018: Integer overflow on the allocation size
gGlobalInfo.rgba = AllocateZeroPool((4 * width * height));
if ( !gGlobalInfo.rgba )
  return EFI_UNSUPPORTED;
gGlobalInfo.idx = -1;
gGlobalInfo.y = 0;
gGlobalInfo.filter = 0;
gGlobalInfo.inLineCount = 0;
```

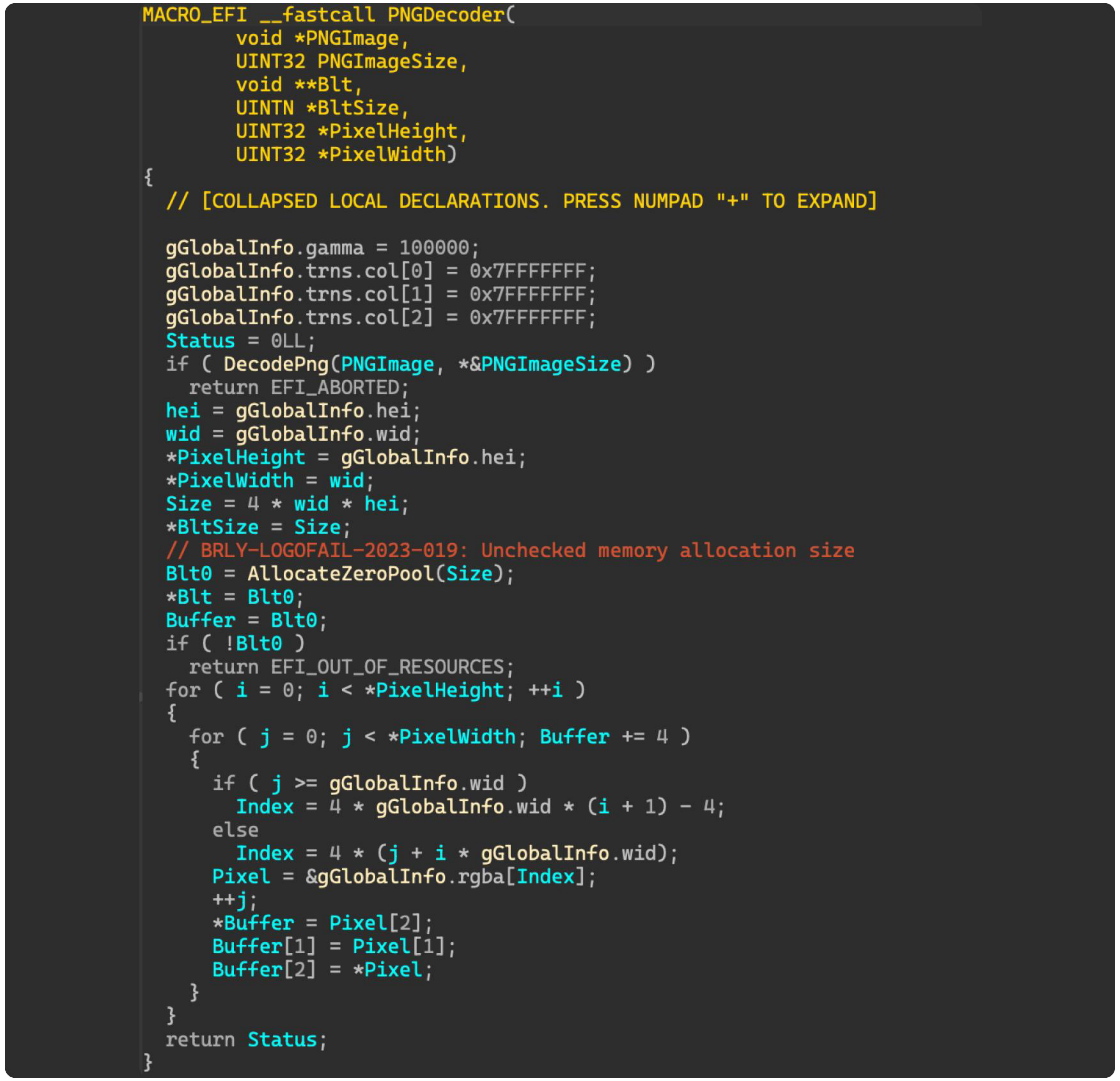
***Fix:* add check for** `width * height`



```
height = gGlobalInfo.hdr.height;
width = gGlobalInfo.hdr.width;
*&gGlobalInfo.wid = *&gGlobalInfo.hdr.width;
if ( gGlobalInfo.autoDeleteRgbaBuffer == 1 && gGlobalInfo.rgba )
{
  FreePool(&gGlobalInfo.rgba);
  gGlobalInfo.rgba = 0LL;
  height = gGlobalInfo.hei;
  width = gGlobalInfo.wid;
}
if ( !width )
  return EFI_UNSUPPORTED;
if ( !height )
  return EFI_UNSUPPORTED;
NumOfPixels = width * height;
// check for BRLY-LOGOFAIL-2023-018
if ( NumOfPixels >= 0x40000000 )
  return EFI_UNSUPPORTED;
gGlobalInfo.rgba = AllocateZeroPool(4 * NumOfPixels);
if ( !gGlobalInfo.rgba )
  return EFI_UNSUPPORTED;
gGlobalInfo.idx = -1;
gGlobalInfo.y = 0;
gGlobalInfo.filter = 0;
gGlobalInfo.inLineCount = 0;
gGlobalInfo.inPixelCount = 0;
gGlobalInfo.index = 0;
```

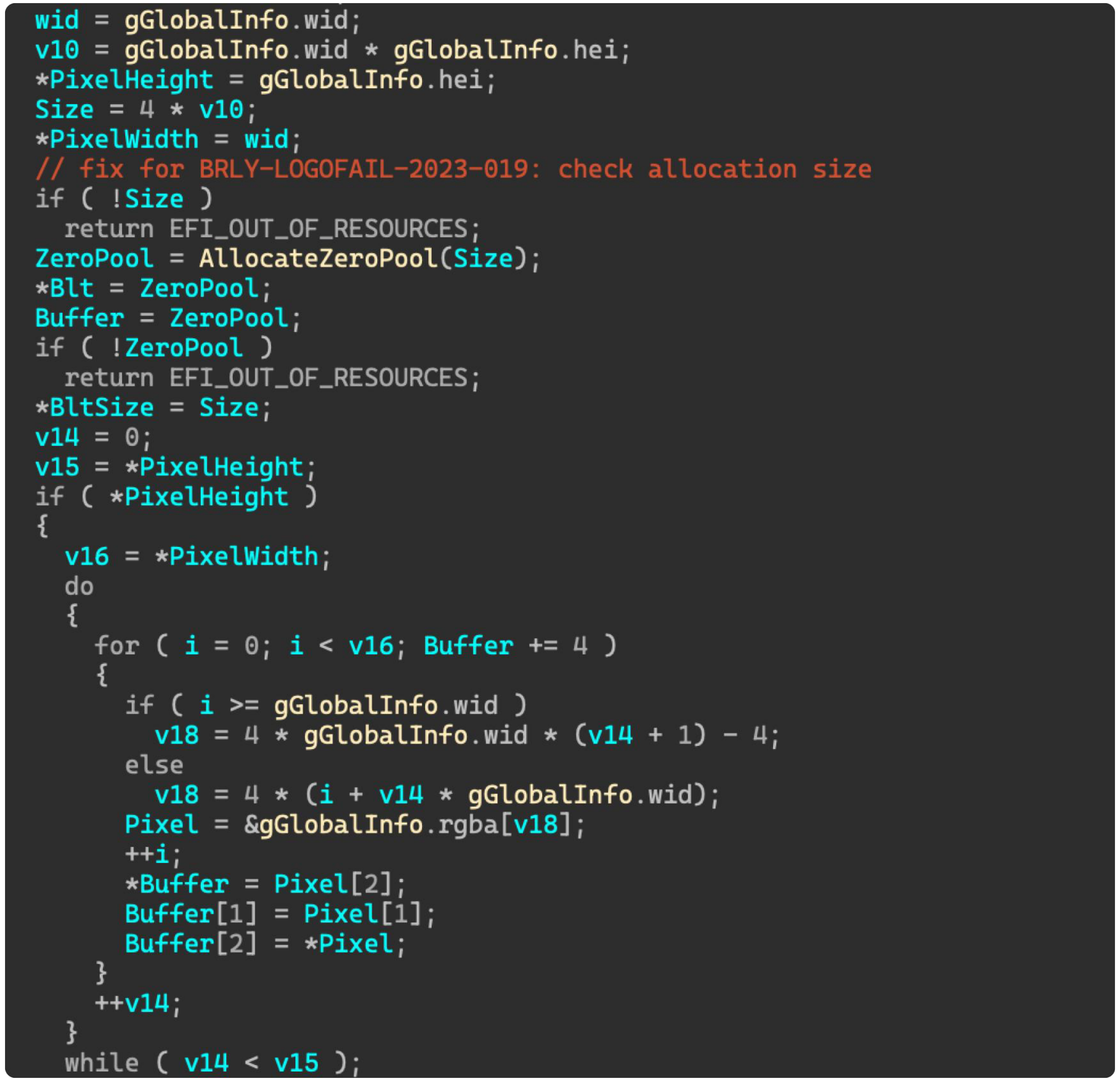
BRLY-LOGOFAIL-2023-019 – **fixed** ✅

In the decoder entry point:



```
MACRO_EFI __fastcall PNGDecoder(
        void *PNGImage,
        UINT32 PNGImageSize,
        void **Blt,
        UINTN *BltSize,
        UINT32 *PixelHeight,
        UINT32 *PixelWidth)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  gGlobalInfo.gamma = 100000;
  gGlobalInfo.trns.col[0] = 0x7FFFFFFF;
  gGlobalInfo.trns.col[1] = 0x7FFFFFFF;
  gGlobalInfo.trns.col[2] = 0x7FFFFFFF;
  Status = 0LL;
  if ( DecodePng(PNGImage, *&PNGImageSize) )
    return EFI_ABORTED;
  hei = gGlobalInfo.hei;
  wid = gGlobalInfo.wid;
  *PixelHeight = gGlobalInfo.hei;
  *PixelWidth = wid;
  Size = 4 * wid * hei;
  *BltSize = Size;
  // BRLY-LOGOFAIL-2023-019: Unchecked memory allocation size
  Blt0 = AllocateZeroPool(Size);
  *Blt = Blt0;
  Buffer = Blt0;
  if ( !Blt0 )
    return EFI_OUT_OF_RESOURCES;
  for ( i = 0; i < *PixelHeight; ++i )
  {
    for ( j = 0; j < *PixelWidth; Buffer += 4 )
    {
      if ( j >= gGlobalInfo.wid )
        Index = 4 * gGlobalInfo.wid * (i + 1) - 4;
      else
        Index = 4 * (j + i * gGlobalInfo.wid);
      Pixel = &gGlobalInfo.rgba[Index];
      ++j;
      *Buffer = Pixel[2];
      Buffer[1] = Pixel[1];
      Buffer[2] = *Pixel;
    }
  }
  return Status;
}
```

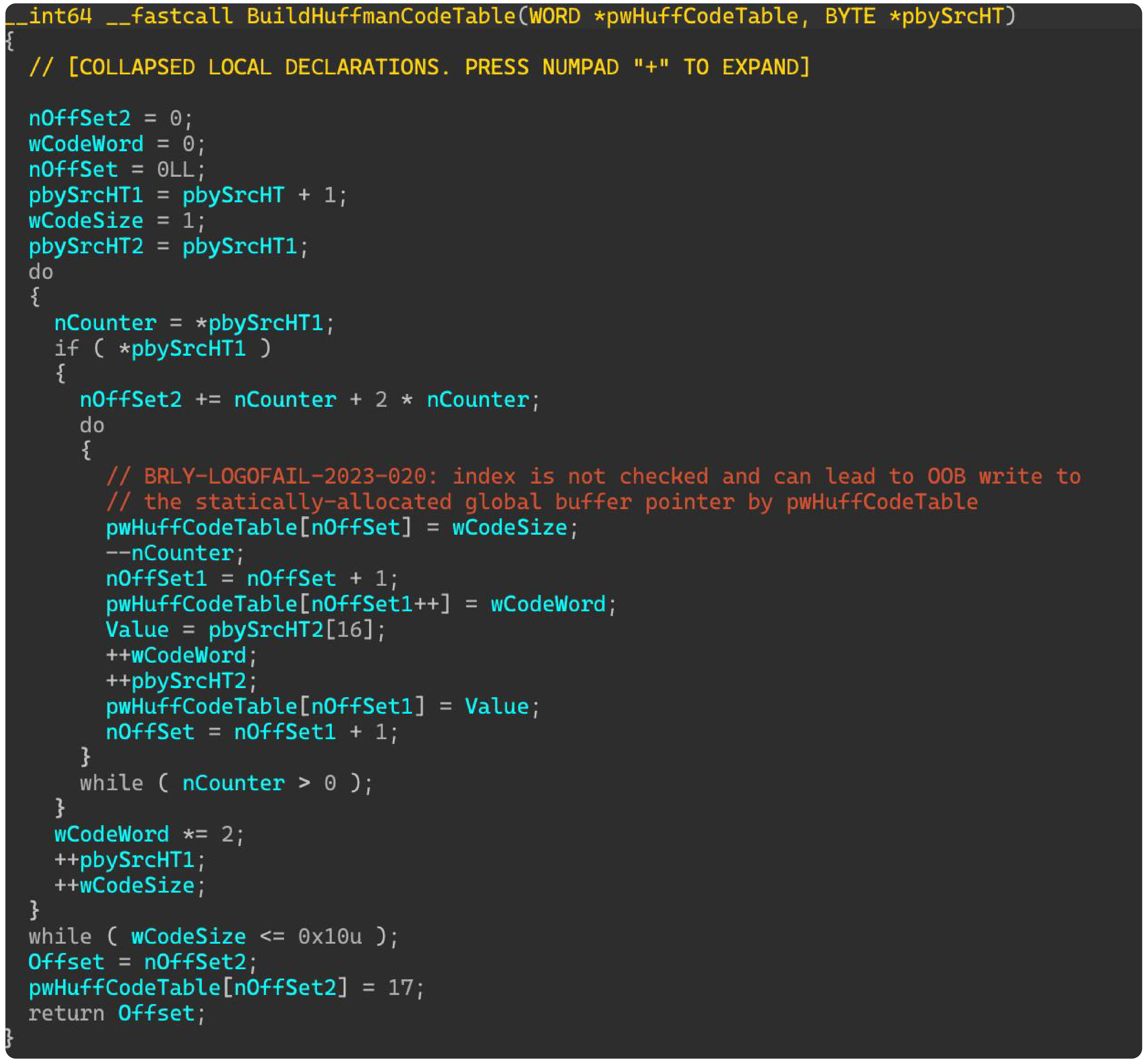
***Fix:* check allocation size**



```
wid = gGlobalInfo.wid;
v10 = gGlobalInfo.wid * gGlobalInfo.hei;
*PixelHeight = gGlobalInfo.hei;
Size = 4 * v10;
*PixelWidth = wid;
// fix for BRLY-LOGOFAIL-2023-019: check allocation size
if ( !Size )
  return EFI_OUT_OF_RESOURCES;
ZeroPool = AllocateZeroPool(Size);
*Blt = ZeroPool;
Buffer = ZeroPool;
if ( !ZeroPool )
  return EFI_OUT_OF_RESOURCES;
*BltSize = Size;
v14 = 0;
v15 = *PixelHeight;
if ( *PixelHeight )
{
  v16 = *PixelWidth;
  do
  {
    for ( i = 0; i < v16; Buffer += 4 )
    {
      if ( i >= gGlobalInfo.wid )
        v18 = 4 * gGlobalInfo.wid * (v14 + 1) - 4;
      else
        v18 = 4 * (i + v14 * gGlobalInfo.wid);
      Pixel = &gGlobalInfo.rgba[v18];
      ++i;
      *Buffer = Pixel[2];
      Buffer[1] = Pixel[1];
      Buffer[2] = *Pixel;
    }
    ++v14;
  }
  while ( v14 < v15 );
```

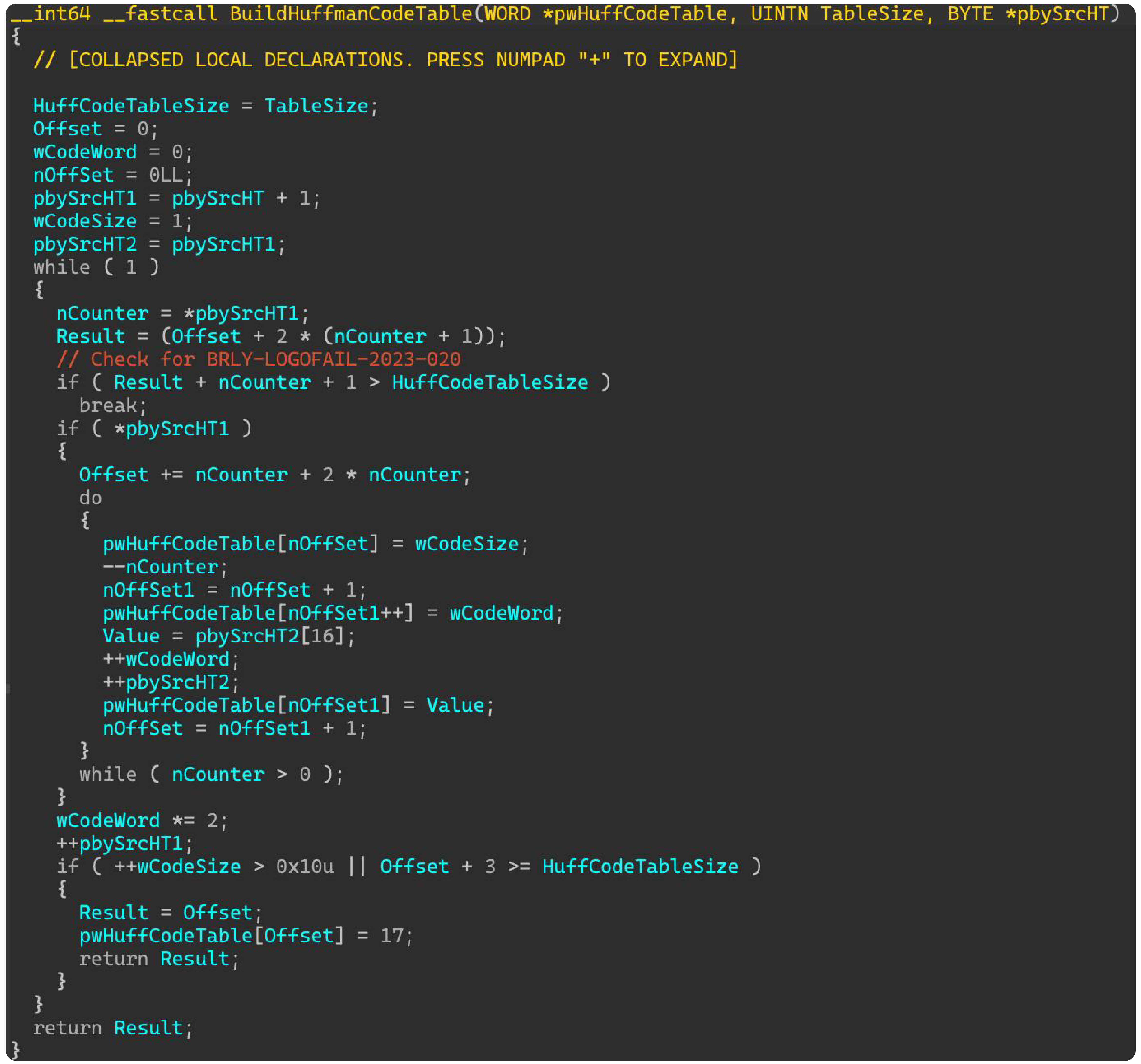
JPEG Decoder

BRLY-LOGOFAIL-2023-020 – **fixed** ✅



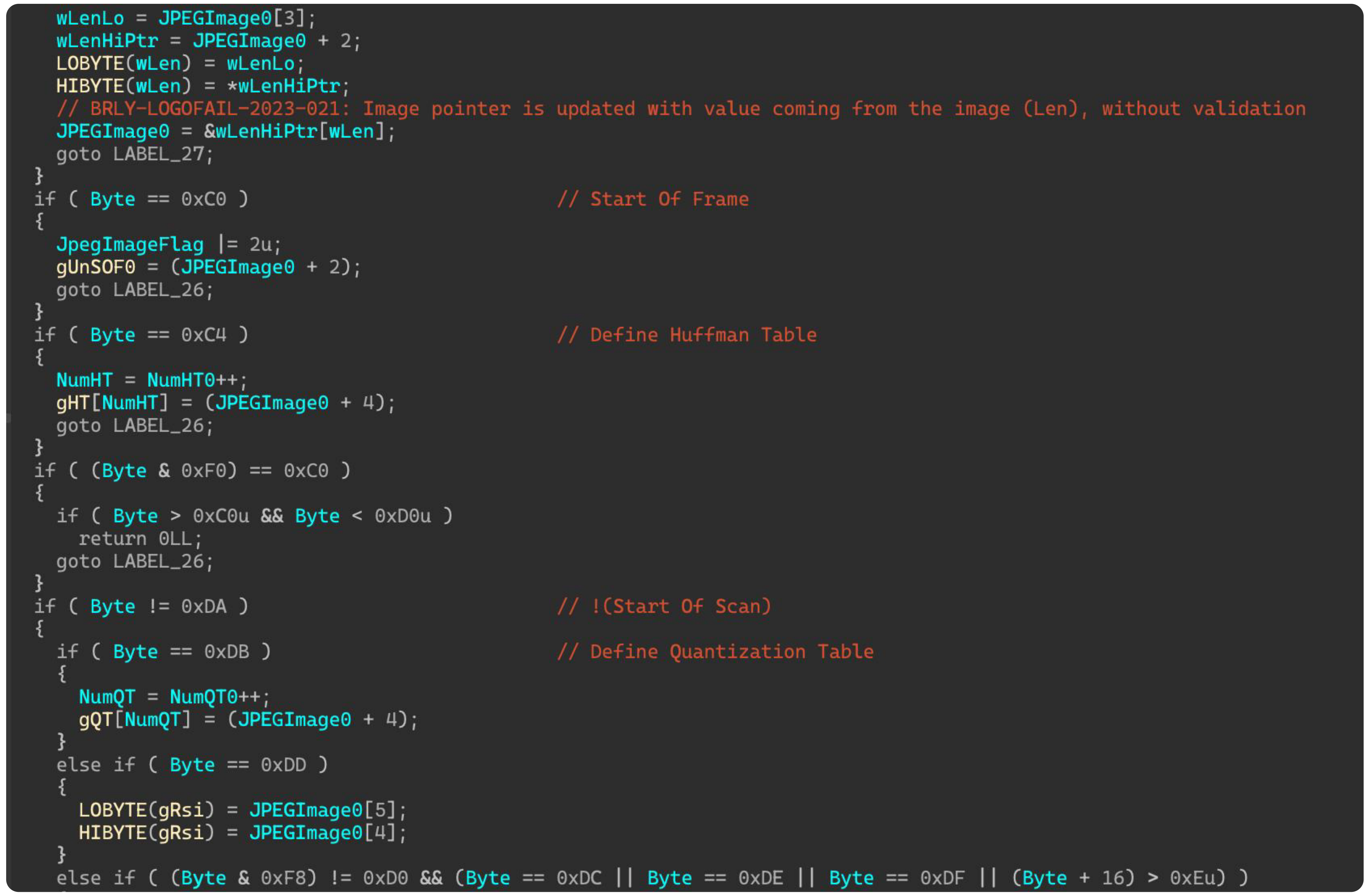
```
__int64 __fastcall BuildHuffmanCodeTable(WORD *pwHuffCodeTable, BYTE *pbySrcHT)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  nOffSet2 = 0;
  wCodeWord = 0;
  nOffSet = 0LL;
  pbySrcHT1 = pbySrcHT + 1;
  wCodeSize = 1;
  pbySrcHT2 = pbySrcHT1;
  do
  {
    nCounter = *pbySrcHT1;
    if ( *pbySrcHT1 )
    {
      nOffSet2 += nCounter + 2 * nCounter;
      do
      {
        // BRLY-LOGOFAIL-2023-020: index is not checked and can lead to OOB write to
        // the statically-allocated global buffer pointer by pwHuffCodeTable
        pwHuffCodeTable[nOffSet] = wCodeSize;
        --nCounter;
        nOffSet1 = nOffSet + 1;
        pwHuffCodeTable[nOffSet1++] = wCodeWord;
        Value = pbySrcHT2[16];
        ++wCodeWord;
        ++pbySrcHT2;
        pwHuffCodeTable[nOffSet1] = Value;
        nOffSet = nOffSet1 + 1;
      }
      while ( nCounter > 0 );
    }
    wCodeWord *= 2;
    ++pbySrcHT1;
    ++wCodeSize;
  }
  while ( wCodeSize <= 0x10u );
  Offset = nOffSet2;
  pwHuffCodeTable[nOffSet2] = 17;
  return Offset;
}
```

Fix: **added a check to prevent writing outside the table**



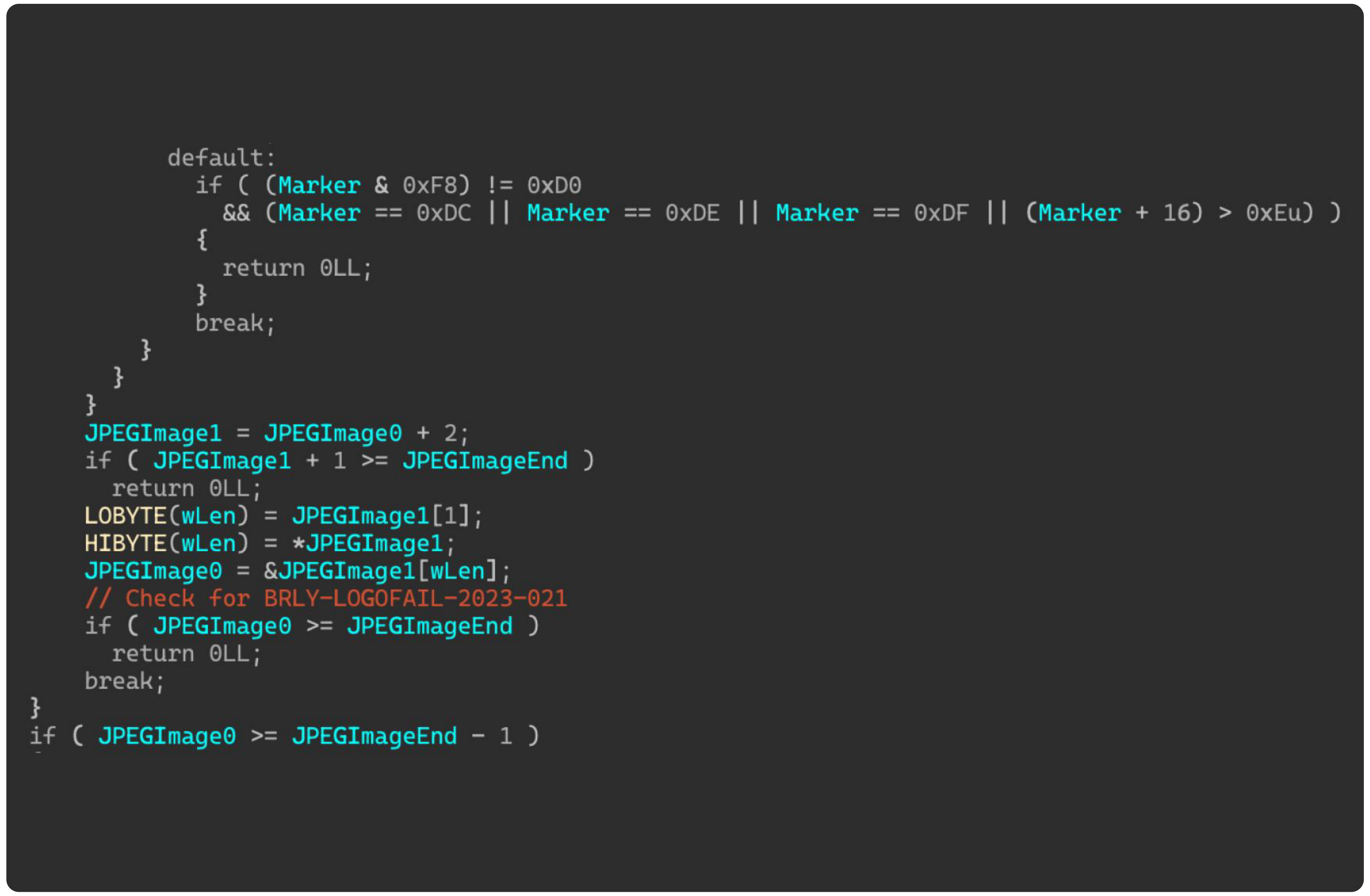
```
__int64 __fastcall BuildHuffmanCodeTable(WORD *pwHuffCodeTable, UINTN TableSize, BYTE *pbySrcHT)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  HuffCodeTableSize = TableSize;
  Offset = 0;
  wCodeWord = 0;
  nOffSet = 0LL;
  pbySrcHT1 = pbySrcHT + 1;
  wCodeSize = 1;
  pbySrcHT2 = pbySrcHT1;
  while ( 1 )
  {
    nCounter = *pbySrcHT1;
    Result = (Offset + 2 * (nCounter + 1));
    // Check for BRLY-LOGOFAIL-2023-020
    if ( Result + nCounter + 1 > HuffCodeTableSize )
      break;
    if ( *pbySrcHT1 )
    {
      Offset += nCounter + 2 * nCounter;
      do
      {
        pwHuffCodeTable[nOffSet] = wCodeSize;
        --nCounter;
        nOffSet1 = nOffSet + 1;
        pwHuffCodeTable[nOffSet1++] = wCodeWord;
        Value = pbySrcHT2[16];
        ++wCodeWord;
        ++pbySrcHT2;
        pwHuffCodeTable[nOffSet1] = Value;
        nOffSet = nOffSet1 + 1;
      }
      while ( nCounter > 0 );
    }
    wCodeWord *= 2;
    ++pbySrcHT1;
    if ( ++wCodeSize > 0x10u || Offset + 3 >= HuffCodeTableSize )
    {
      Result = Offset;
      pwHuffCodeTable[Offset] = 17;
      return Result;
    }
  }
  return Result;
}
```

BRLY-LOGOFAIL-2023-021 – **fixed** ✅



```
    wLenLo = JPEGImage0[3];
    wLenHiPtr = JPEGImage0 + 2;
    LOBYTE(wLen) = wLenLo;
    HIBYTE(wLen) = *wLenHiPtr;
    // BRLY-LOGOFAIL-2023-021: Image pointer is updated with value coming from the image (Len), without validation
    JPEGImage0 = &wLenHiPtr[wLen];
    goto LABEL_27;
  }
  if ( Byte == 0xC0 )                           // Start Of Frame
  {
    JpegImageFlag |= 2u;
    gUnSOF0 = (JPEGImage0 + 2);
    goto LABEL_26;
  }
  if ( Byte == 0xC4 )                           // Define Huffman Table
  {
    NumHT = NumHT0++;
    gHT[NumHT] = (JPEGImage0 + 4);
    goto LABEL_26;
  }
  if ( (Byte & 0xF0) == 0xC0 )
  {
    if ( Byte > 0xC0u && Byte < 0xD0u )
      return 0LL;
    goto LABEL_26;
  }
  if ( Byte != 0xDA )                           // !(Start Of Scan)
  {
    if ( Byte == 0xDB )                         // Define Quantization Table
    {
      NumQT = NumQT0++;
      gQT[NumQT] = (JPEGImage0 + 4);
    }
    else if ( Byte == 0xDD )
    {
      LOBYTE(gRsi) = JPEGImage0[5];
      HIBYTE(gRsi) = JPEGImage0[4];
    }
    else if ( (Byte & 0xF8) != 0xD0 && (Byte == 0xDC || Byte == 0xDE || Byte == 0xDF || (Byte + 16) > 0xEu) )
```

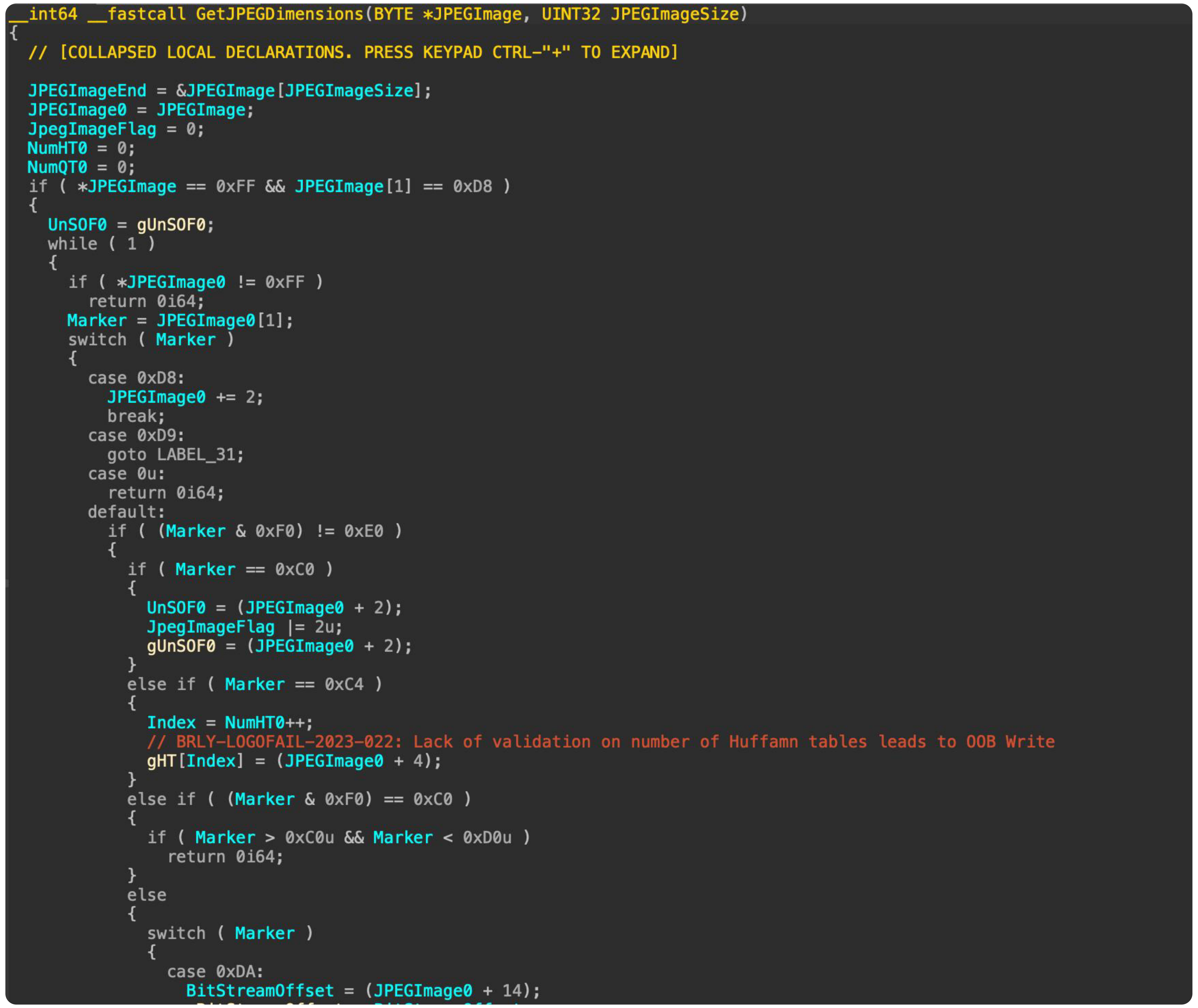
Fix: **added check for** `wLen`



```
      default:
        if ( (Marker & 0xF8) != 0xD0
          && (Marker == 0xDC || Marker == 0xDE || Marker == 0xDF || (Marker + 16) > 0xEu) )
        {
          return 0LL;
        }
        break;
    }
  }
  }
  JPEGImage1 = JPEGImage0 + 2;
  if ( JPEGImage1 + 1 >= JPEGImageEnd )
    return 0LL;
  LOBYTE(wLen) = JPEGImage1[1];
  HIBYTE(wLen) = *JPEGImage1;
  JPEGImage0 = &JPEGImage1[wLen];
  // Check for BRLY-LOGOFAIL-2023-021
  if ( JPEGImage0 >= JPEGImageEnd )
    return 0LL;
  break;
}
if ( JPEGImage0 >= JPEGImageEnd - 1 )
```

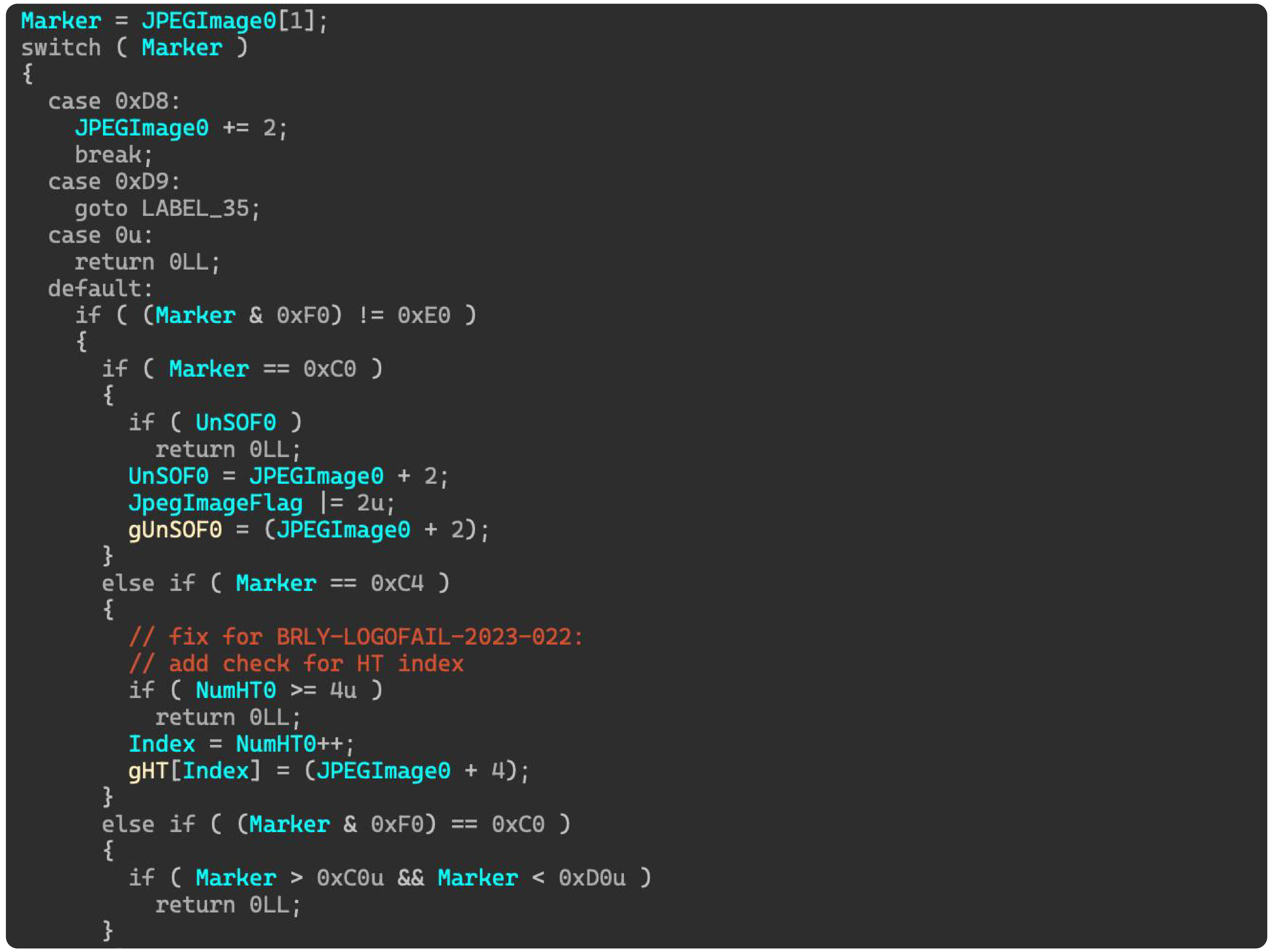
BRLY-LOGOFAIL-2023-022 – **fixed** ✅

In `GetJpegDimensions()` **function the index used to access the** `HuffamTables` **array is not checked.**



```
__int64 __fastcall GetJPEGDimensions(BYTE *JPEGImage, UINT32 JPEGImageSize)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  JPEGImageEnd = &JPEGImage[JPEGImageSize];
  JPEGImage0 = JPEGImage;
  JpegImageFlag = 0;
  NumHT0 = 0;
  NumQT0 = 0;
  if ( *JPEGImage == 0xFF && JPEGImage[1] == 0xD8 )
  {
    UnSOF0 = gUnSOF0;
    while ( 1 )
    {
      if ( *JPEGImage0 != 0xFF )
        return 0i64;
      Marker = JPEGImage0[1];
      switch ( Marker )
      {
        case 0xD8:
          JPEGImage0 += 2;
          break;
        case 0xD9:
          goto LABEL_31;
        case 0u:
          return 0i64;
        default:
          if ( (Marker & 0xF0) != 0xE0 )
          {
            if ( Marker == 0xC0 )
            {
              UnSOF0 = (JPEGImage0 + 2);
              JpegImageFlag |= 2u;
              gUnSOF0 = (JPEGImage0 + 2);
            }
            else if ( Marker == 0xC4 )
            {
              Index = NumHT0++;
              // BRLY-LOGOFAIL-2023-022: Lack of validation on number of Huffamn tables leads to OOB Write
              gHT[Index] = (JPEGImage0 + 4);
            }
            else if ( (Marker & 0xF0) == 0xC0 )
            {
              if ( Marker > 0xC0u && Marker < 0xD0u )
                return 0i64;
            }
            else
            {
              switch ( Marker )
              {
                case 0xDA:
                  BitStreamOffset = (JPEGImage0 + 14);
```

Fix: **add check for** `HT index`

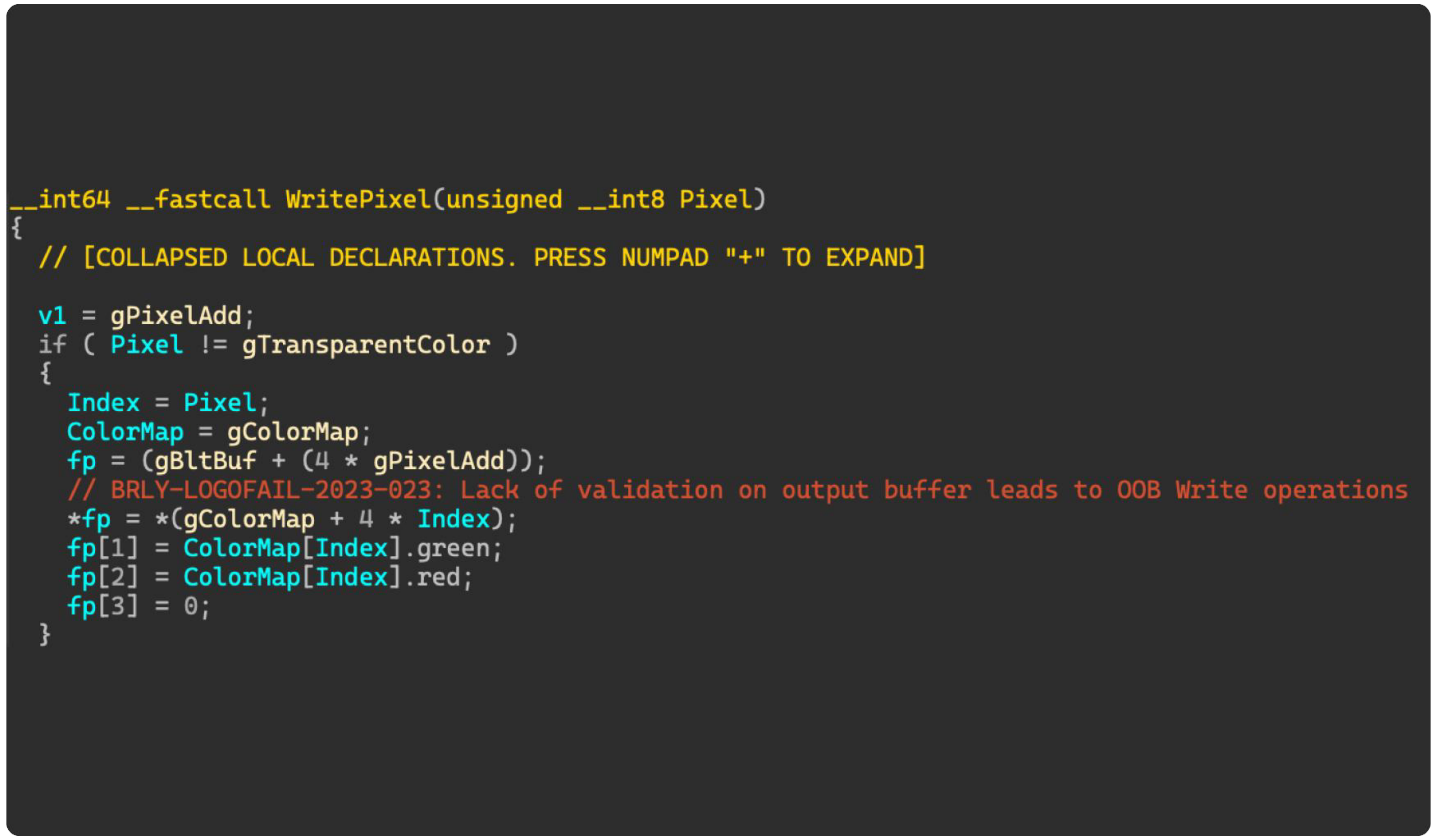


```
Marker = JPEGImage0[1];
switch ( Marker )
{
  case 0xD8:
    JPEGImage0 += 2;
    break;
  case 0xD9:
    goto LABEL_35;
  case 0u:
    return 0LL;
  default:
    if ( (Marker & 0xF0) != 0xE0 )
    {
      if ( Marker == 0xC0 )
      {
        if ( UnSOF0 )
          return 0LL;
        UnSOF0 = JPEGImage0 + 2;
        JpegImageFlag |= 2u;
        gUnSOF0 = (JPEGImage0 + 2);
      }
      else if ( Marker == 0xC4 )
      {
        // fix for BRLY-LOGOFAIL-2023-022:
        // add check for HT index
        if ( NumHT0 >= 4u )
          return 0LL;
        Index = NumHT0++;
        gHT[Index] = (JPEGImage0 + 4);
      }
      else if ( (Marker & 0xF0) == 0xC0 )
      {
        if ( Marker > 0xC0u && Marker < 0xD0u )
          return 0LL;
      }
```

GIF Decoder

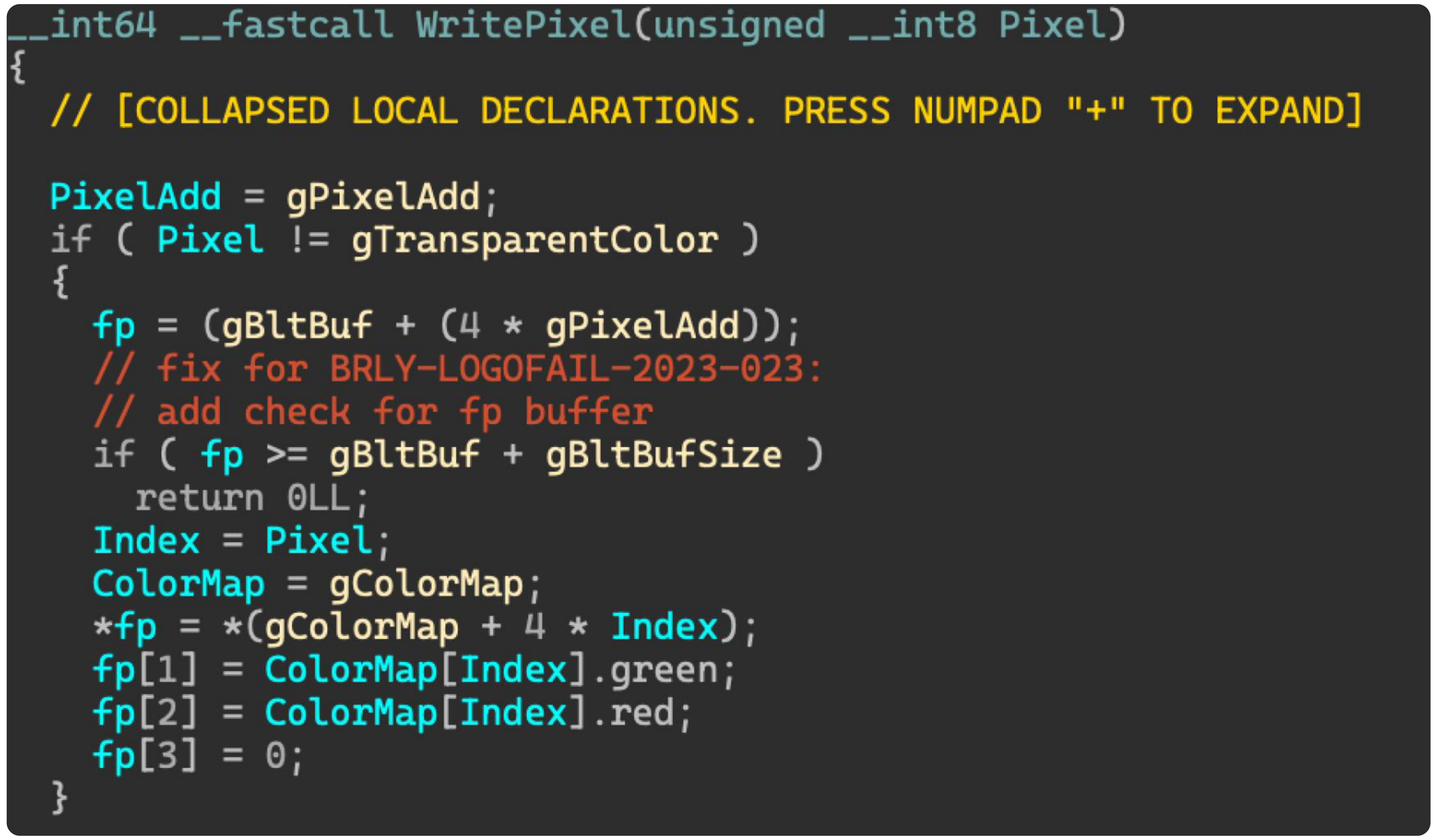
BRLY-LOGOFAIL-2023-023 – **fixed** ✅

In `WritePixel()` function



```
__int64 __fastcall WritePixel(unsigned __int8 Pixel)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  v1 = gPixelAdd;
  if ( Pixel != gTransparentColor )
  {
    Index = Pixel;
    ColorMap = gColorMap;
    fp = (gBltBuf + (4 * gPixelAdd));
    // BRLY-LOGOFAIL-2023-023: Lack of validation on output buffer leads to OOB Write operations
    *fp = *(gColorMap + 4 * Index);
    fp[1] = ColorMap[Index].green;
    fp[2] = ColorMap[Index].red;
    fp[3] = 0;
  }
```

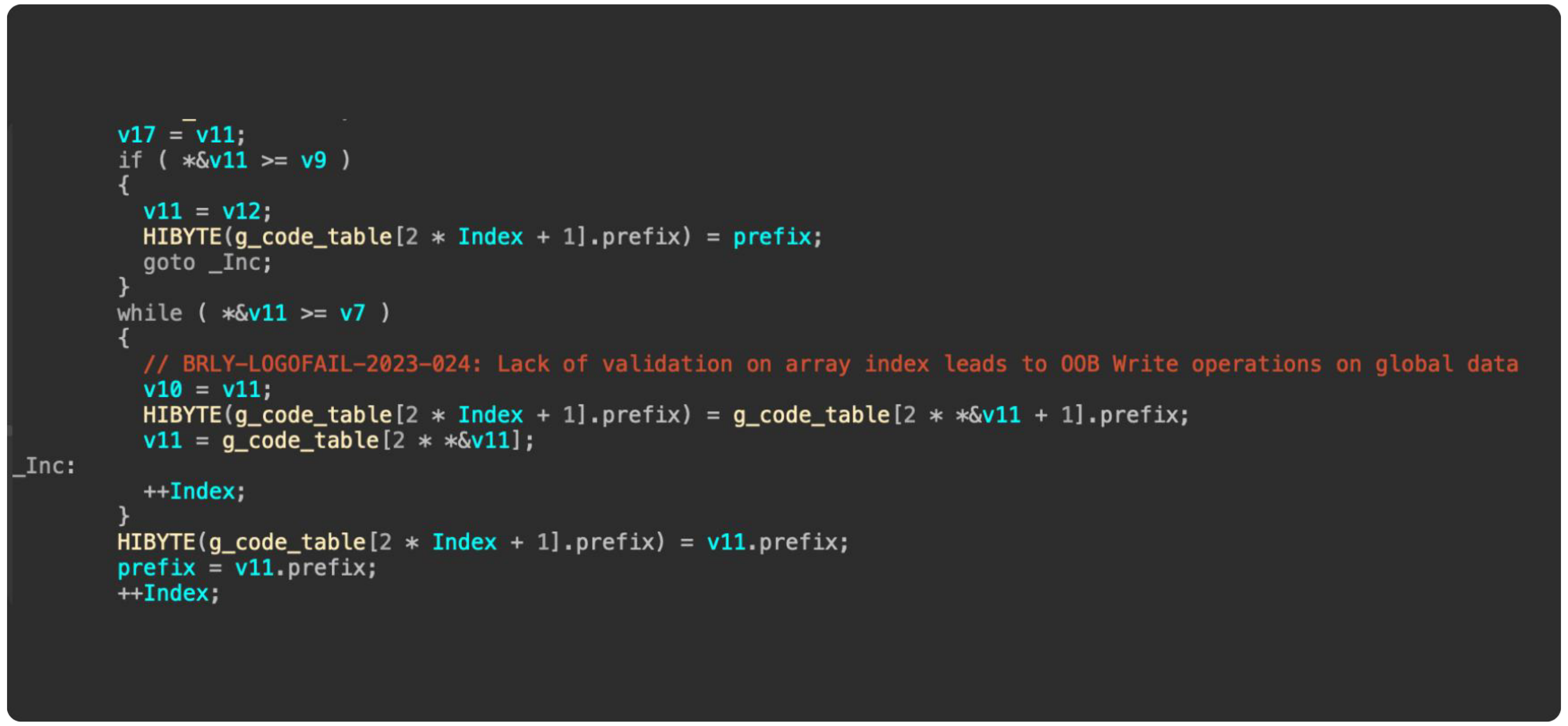
***Fix:* add check for `fp` buffer**



```
__int64 __fastcall WritePixel(unsigned __int8 Pixel)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  PixelAdd = gPixelAdd;
  if ( Pixel != gTransparentColor )
  {
    fp = (gBltBuf + (4 * gPixelAdd));
    // fix for BRLY-LOGOFAIL-2023-023:
    // add check for fp buffer
    if ( fp >= gBltBuf + gBltBufSize )
      return 0LL;
    Index = Pixel;
    ColorMap = gColorMap;
    *fp = *(gColorMap + 4 * Index);
    fp[1] = ColorMap[Index].green;
    fp[2] = ColorMap[Index].red;
    fp[3] = 0;
  }
```

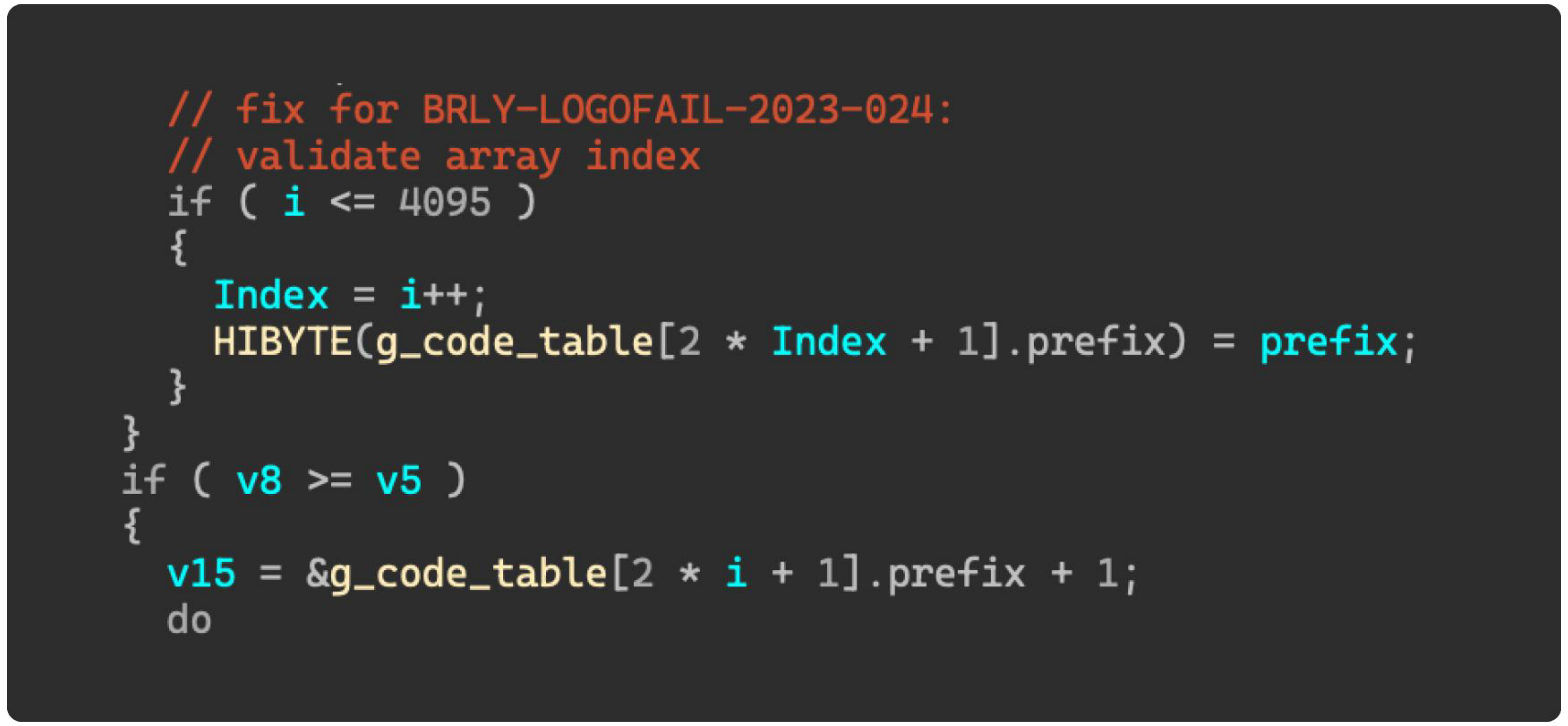
`BRLY-LOGOFAIL-2023-024` – **fixed** ✅

In `ExpandData()` **function:**



```
        v17 = v11;
        if ( *&v11 >= v9 )
        {
          v11 = v12;
          HIBYTE(g_code_table[2 * Index + 1].prefix) = prefix;
          goto _Inc;
        }
        while ( *&v11 >= v7 )
        {
          // BRLY-LOGOFAIL-2023-024: Lack of validation on array index leads to OOB Write operations on global data
          v10 = v11;
          HIBYTE(g_code_table[2 * Index + 1].prefix) = g_code_table[2 * *&v11 + 1].prefix;
          v11 = g_code_table[2 * *&v11];
_Inc:
          ++Index;
        }
        HIBYTE(g_code_table[2 * Index + 1].prefix) = v11.prefix;
        prefix = v11.prefix;
        ++Index;
```

***Fix:* added check for** `array index`



```
      // fix for BRLY-LOGOFAIL-2023-024:
      // validate array index
      if ( i <= 4095 )
      {
        Index = i++;
        HIBYTE(g_code_table[2 * Index + 1].prefix) = prefix;
      }
    }
    if ( v8 >= v5 )
    {
      v15 = &g_code_table[2 * i + 1].prefix + 1;
      do
```

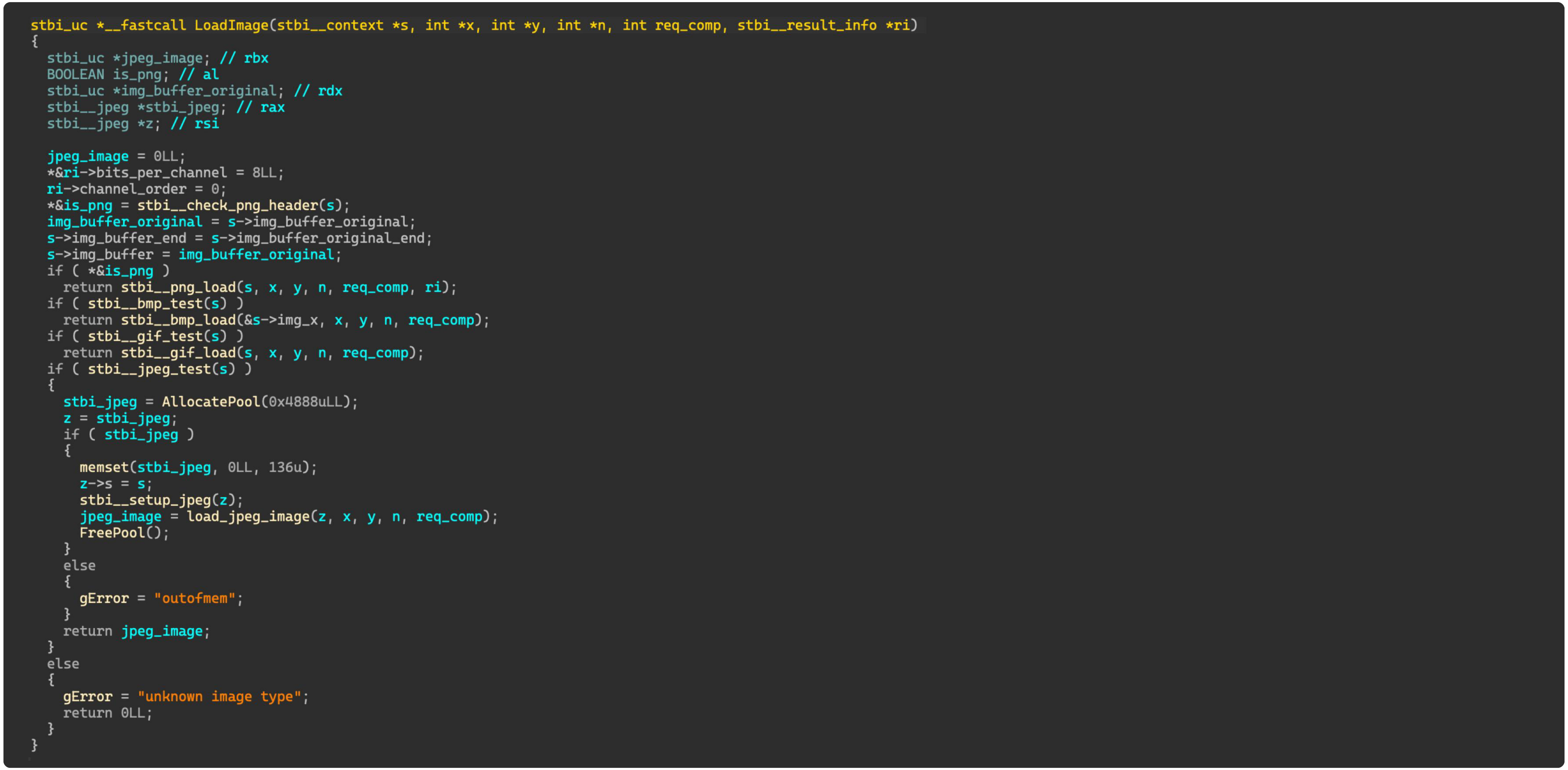
Patch Breakdown and Incomplete Fixes

Phoenix

Summary of the fixes:

Phoenix moved to another parsing library: stb_image.h. We did not identify any problems during the fuzzing of the new version. This library introduced an additional PNG parser besides the listed above.

Phoenix



```
stbi_uc *__fastcall LoadImage(stbi__context *s, int *x, int *y, int *n, int req_comp, stbi__result_info *ri)
{
  stbi_uc *jpeg_image; // rbx
  BOOLEAN is_png; // al
  stbi_uc *img_buffer_original; // rdx
  stbi__jpeg *stbi_jpeg; // rax
  stbi__jpeg *z; // rsi

  jpeg_image = 0LL;
  *&ri->bits_per_channel = 8LL;
  ri->channel_order = 0;
  *&is_png = stbi__check_png_header(s);
  img_buffer_original = s->img_buffer_original;
  s->img_buffer_end = s->img_buffer_original_end;
  s->img_buffer = img_buffer_original;
  if ( *&is_png )
    return stbi__png_load(s, x, y, n, req_comp, ri);
  if ( stbi__bmp_test(s) )
    return stbi__bmp_load(&s->img_x, x, y, n, req_comp);
  if ( stbi__gif_test(s) )
    return stbi__gif_load(s, x, y, n, req_comp);
  if ( stbi__jpeg_test(s) )
  {
    stbi_jpeg = AllocatePool(0x4888uLL);
    z = stbi_jpeg;
    if ( stbi_jpeg )
    {
      memset(stbi_jpeg, 0LL, 136u);
      z->s = s;
      stbi__setup_jpeg(z);
      jpeg_image = load_jpeg_image(z, x, y, n, req_comp);
      FreePool();
    }
    else
    {
      gError = "outofmem";
    }
    return jpeg_image;
  }
  else
  {
    gError = "unknown image type";
    return 0LL;
  }
}
```