VERSPRITE

OPTO 22 PAC CONTROL BASIC

ZERO-DAY DISCLOSURE

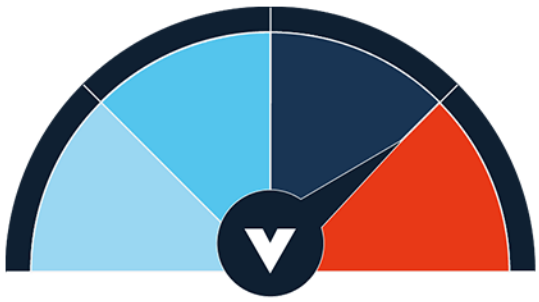
SECURITY RESEARCH

EXECUTIVE SUMMARY

VerSprite's Research and Development Team, VS-Labs, discovered a vulnerability in OPTO 22's Control Basic Software suite that affects the Industrial Control System (ICS) and Operational Technology (OT) industries.

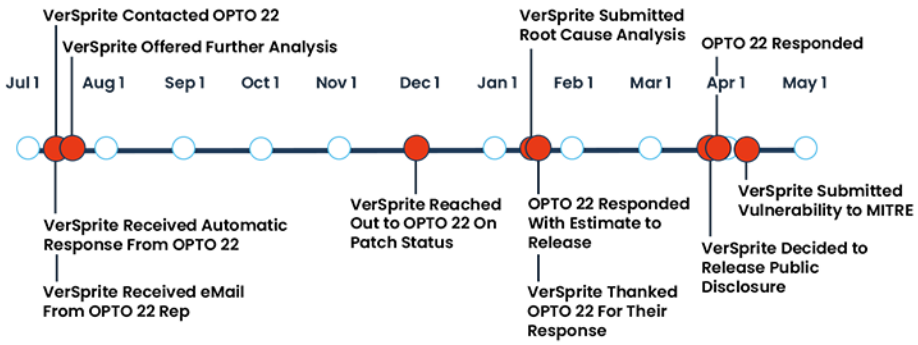
To date, this software remains unpatched and is a **high-critical zero-day** vulnerability that can leave ICS and OT organizations open to attack by malicious actors.

VerSprite's VS-Labs initially discovered the **Control.basic.exe** vulnerability in July of 2020. Following proper protocol, we reached out to OPTO 22 within days of discovering the vulnerability and gave them ample time to produce a fix. Due to their inaction, we are releasing the vulnerability synopsis to raise awareness around this security issue. Please refer to our Vendor Disclosure Timeline on page 2 to review the steps we took to uncover the OPTO 22 PAC Control vulnerability.



HIGH-CRITICAL
RISK LEVEL

VENDOR DISCLOSURE TIMELINE



Vendor Disclosure Timeline

07-06-2020 Contacted OPTO 22 and submitted initial security issues to vendor.

07-06-2020 Received Automatic response from OPTO 22 Product Support Group (PSG).

07-06-2020 Received email from OPTO 22 Representative explaining they were able to duplicate said issues

07-08-2020 We offered to provide some further analysis; however, due to time constraints with other high priority clients, we were unable to assist further at the time.

12-08-2020 We reached out to OPTO 22 again in December, to check on the status of the patching of security issues and they did not have any updates on when a fix would be produced.

01-20-2021 We submitted a report with root cause analysis and technical details of the issues to OPTO 22. We also inquired about when a patch would be available.

01-21-2021 OPTO 22 Responded stating that they plan to release the fixes within PAC 10.14; however, no date for that version release has been scheduled at that time.

01-22-2021 OPTO 22 Responded to our request of further details of dates by stating they had a rough estimate of mid-year and if they received any newer information, they would email it to us.

01-22-2021 We thanked OPTO 22 for their response and let them know we would reach out in June 2021 to check status.

03-26-2021 After not receiving any new information about release schedule and further analysis of the product suite and industry that OPTO 22 operates within, we decided to move forward with public disclosure, 8 months is ample time for patching and remediation, and we feel the public needs to be aware of risks associated with software that operates within critical industries. We let OPTO 22 know that they would be releasing the vulnerability information public within 30 days.

03-29-2021 OPTO 22 responded saying that a fix would be provided in the next PAC Project Version.

04-09-2021 VerSprite submitted vulnerability details to MITRE to receive CVE ID.

OPTO22 PAC CONTROL BASIC VULNERABILITY SYNOPSIS

In the land of software development, a common hurdle faced by developers is creating secure file parsers. File parsers are responsible for processing an incoming file where the file stream data must adhere to a strict format that the software can accept. Not only is the parser responsible for processing the file stream data accurately, but also ***securely*** as processing files are subject to attacker manipulation.

This seems like a simple task. However, in practice, creating secure file parsers is not that easy, which is demonstrated by the hundreds (possibly thousands) of file parsing CVE's released every year from MITRE. File parsing vulnerabilities have plagued many software development teams for years and will continue to cause issues if file parsers remain written in system-level languages like **C** or **C++**, where the memory management is left up to the developer to handle entirely.

This report demonstrates how developing in system-level languages results in file parsing errors. VerSprite security researchers perform an initial root cause analysis of a file format parsing vulnerability they discovered within the OPTO 22 PAC Control Basic software suite.

This report only covers the basics of file parsing vulnerabilities, so interested readers are encouraged to further explore function calls within the main binary **Control.basic.exe** imported from the **IOCDB.dll** dynamic

VULNERABILITY OVERVIEW

VerSprite VS-Labs researchers discovered a vulnerability within OPTO 22's PAC Control Basic software suite's main application, **Control.basic.exe**. This application's file parser is vulnerable due to an Out-of-Bounds Read (OOB[R]) during the parsing of a modified **idb** file within a projects strategy folder. The OOB[R] occurs due to misuse of the ***ATL/MFC CStringT class constructor***. The **CStringT** class constructor is responsible for performing string copy operations, and has the potential for causing memory exceptions as detailed in the **Remarks** section of the official Microsoft Documentation.

Because the constructors copy the input data into new allocated storage, memory exceptions may result. Some of these constructors act as conversion functions. This allows you to substitute, for example, a LPTSTR where a CStringT object is expected.

REVIEW OF INITIAL ACCESS VIOLATION

Through utilizing the Windows Debugger (WinDbg Preview) from the Microsoft Store, we were able to record the **Access Violation** during the parsing of the malicious **idb** file within the **Control.basic.exe** binary. The initial **Access Violation** recorded with WinDbg's Time Travel Debugger (TTD) can be seen in the table below.

Malicious-Trigger.idb

File Information:

- SHA256 - 5C61148D316A07849D2EA10008D47141B4DA38951C6942706712051885565E595
- Command:
  - PS C:\Users\User> Get-FileHash C:\Users\User\Desktop\PoCs\Opto22-New\Malicious\Case1\Malicious-Trigger.idb

```
(19ec.1fe0): Access violation - code c0000005 (first/second chance not available)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
Time Travel Position: E315A:0
Unable to load image C:\WINDOWS\SYSTEM32\mfc120.dll, Win32 error 0n2
eax=1da7f000 ebx=08d9af28 ecx=1da7f0de edx=00000c00 esi=0e6feffc edi=0e6feffc
eip=6c6e303a esp=0019dc18 ebp=0019dc24 iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00200202
MSVCR120!strlen+0xc:
6c6e303a 8a01            mov     al,byte ptr [ecx]          ds:002b:1da7f0de=??
```

An important note is that during testing, symbols may cause issues that were within a VM function and **strcpy_s()** is recorded. However, within a different VM the function, **strlen()** is recorded.

ROOT CAUSE ANALYSIS

The issue stems from an attacker-controlled boundary condition associated within either a while or for loop within the **sub_6900A0** function within the **Control.basic.exe** binary.

The attacker-controlled condition (stored within the pointer at **ECX+202h**), is first checked against an incremented counter (stored within **var_18**), before progressing further into the loop where a pointer to a string is passed as an argument to the function name **case_1_atl_string_creation()** (modified function name). The **case_1_atl_string_creation()** function appears to be a wrapper for the **ATL::CStringT** string class constructor itself. Verification of the initial boundary check is seen in the image below.

```
.text:006901BC                mov     [ebp+var_18], 0 ; Initialize Counter (var_18) to NULL
.text:006901C3                jmp     short loc_6901CE ; Initialize "ECX" to be the pointer to the file stream
.text:006901C5 ; ---------------------------------------------------------------------------
.text:006901C5
.text:006901C5 loc_6901C5:                            ; CODE XREF: sub_6900A0+198↓j
.text:006901C5                mov     eax, [ebp+var_18]
.text:006901C8                add     eax, 1
.text:006901CB                mov     [ebp+var_18], eax
.text:006901CE
.text:006901CE loc_6901CE:                            ; CODE XREF: sub_6900A0+123↑j
.text:006901CE                mov     ecx, [ebp+arg_0] ; Initialize "ECX" to be the pointer to the file stream
.text:006901D1                mov     edx, [ebp+var_18] ; Initialize "EDX" to be utilized during the boundary condition check
.text:006901D4                cmp     edx, [ecx+202h] ; Compare the counter (var_18) against the attacker controlled
.text:006901D4                                        ; value (ecx+202h)
.text:006901DA                jge     short loc_69023A
```

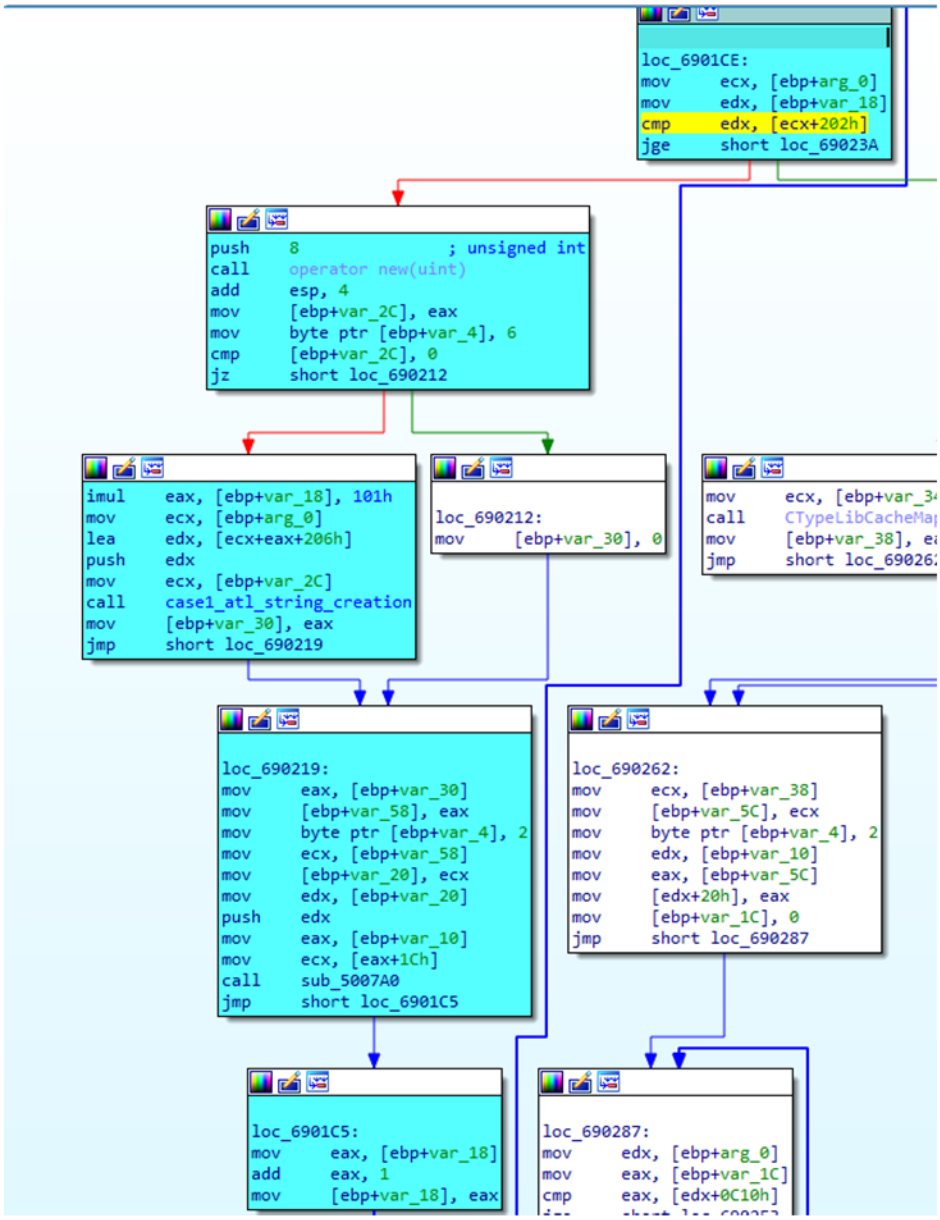
The counter **(var_18)**, is utilized before the **ATL::CStringT** wrapper function call during an **imul** instruction where the resulting value, stored within **EAX**, is used as an index into the **ECX** register. Which at this time holds a buffer of file data extracted from previous **CFile::[Open/Seek/Read]()** function calls within the IOCDB DLL. The ECX register is then further indexed by a static offset of **0x206** into the file stream. The resulting pointer is then assigned to the **EDX** register and passed to the **ATL::CStringT** wrapper function.

Verification of the file stream data within **ECX** and the usage of the counter during the **imul** instruction is seen in the image below.

```
.text:006901F3                imul    eax, [ebp+var_18], 101h ; Multiplication of the "var_18" counter
.text:006901FA                mov     ecx, [ebp+arg_0] ; Loading of file stream buffer into "ECX"
.text:006901FD                lea     edx, [ecx+eax+206h] ; Loading string located at "0x1da7d6c0+(var_18*0x101)+0x206"
.text:006901FD                                        ; into "EDX"
.text:006901FD                                        ;
.text:006901FD                                        ; The Argument to "ATL::CStringT" wrapper
.text:006901FD                                        ;
.text:006901FD                                        ; Verification of "ECX" file stream data during runtime:
.text:006901FD                                        ; 0:002> dc ecx
.text:006901FD                                        ; 1da7d6c0  41414141 00414141 00000000 00000000  AAAAAAA.........
.text:006901FD                                        ; 1da7d6d0  00000000 00000000 00000000 00000000  ................
.text:006901FD                                        ; 1da7d6e0  00000000 00000000 00000000 00000000
.text:006901FD                                        ; 0:002> dc ecx+eax+206h
.text:006901FD                                        ; 1da7d8c6  00000000 00000000 00000000 00000000  ................
.text:006901FD                                        ; 1da7d8d6  00000000 00000000 00000000 00000000  ................
.text:006901FD                                        ; 1da7d8e6  00000000 00000000 00000000 00000000  ................
.text:006901FD                                        ; 1da7d8f6  00000000 00000000 00000000 00000000  ................
.text:006901FD                                        ; 1da7d906  00000000 00000000 00000000 00000000  ................
.text:006901FD                                        ; 1da7d916  00000000 00000000 00000000 00000000  ................
.text:006901FD                                        ; 1da7d926  00000000 00000000 00000000 00000000  ................
.text:006901FD                                        ; 1da7d936  00000000 00000000 00000000 00000000  ................
.text:00690204                push    edx
.text:00690205                mov     ecx, [ebp+var_2C]
.text:00690208                call    case1_atl_string_creation
.text:0069020D                mov     [ebp+var_30], eax
.text:00690210                jmp     short loc_690219
```

With each iteration where the counter (**var_18**) is less than the attacker-controlled value at offset **0x202** within the file buffer (**ECX/arg_0**), the counter is multiplied by **0x101**. With improper bounds checks on the attacker-controlled value (**ECX+202h**), it is possible for an Out-of-Bounds Read condition to present itself. The OOB[R] occurs due to data outside of the intended file buffer being accessed (read) during the **ATL::CStringT** string class constructor because the [**while/for**] loop fails to validate that the pointer is within the proper bounds of the actual file buffer before the **ATL:CStringT** wrapper is called.

Verification of the file stream data within **ECX** and the usage of the counter during the **imul** instruction can be seen in the image below.



MITIGATION

VerSprite Researchers recommend that users of OPTO 22 PAC Control Basic do not engage with any files from non-verified sources, as they have the potential to be malicious in nature. Once a patch is released, update the software suite as soon as possible.

CONCLUSION

This report raises awareness on OPTO 22's software vulnerability within the PAC Control Basic software suite. As demonstrated, the file parsing vulnerability results from incorrect file format parsing prior to file stream manipulation within the **Control.Basic.exe** binary when parsing malicious **ibd** files.

Importantly, file parsing vulnerabilities can come in various iterations. For example, these security issues can stem from improper usage of APIs or even a product development team unintentionally introducing a vulnerability in their own code.

Because of the various ways vulnerabilities are introduced into software, developers need to know that continuous testing must be utilized where a mix between code review and automated code coverage guided fuzzing is performed. Using code review, developers can use static analysis to find potential security flaws while coverage guided fuzzing is performed 24/7. It is important to note that fuzzing is complimentary to code review and both are equally important when it comes to securing one's products.