Research Threat intelligence Microsoft Defender Vulnerability Management Vulnerabilities and exploits
13 min read

Uncursing the ncurses: Memory corruption vulnerabilities found in library

By Microsoft Threat Intelligence

September 14, 2023

Microsoft Defender

Microsoft Defender for Endpoint

Elevation of privilege

Linux

macOS

Microsoft has discovered a set of memory corruption vulnerabilities in a library called *ncurses*, which provides APIs that support text-based user interfaces (TUI). Released in 1993, the *ncurses* library is commonly used by various programs on Portable Operating System Interface (POSIX) operating systems, including Linux, macOS, and FreeBSD. Using environment variable poisoning, attackers could chain these vulnerabilities to elevate privileges and run code in the targeted program's context or perform other malicious actions.

One of the most common vulnerabilities found in modern software, memory corruption vulnerabilities, can allow attackers to gain unauthorized access to systems and data by modifying a program's memory. The impact of memory corruption vulnerabilities can range from leaking sensitive information and performing a simple denial-of-service (DoS) to elevating privileges and executing arbitrary code.

Microsoft has shared these vulnerabilities with the relevant maintainers through Coordinated Vulnerability Disclosure (CVD) via Microsoft Security Vulnerability Research (MSVR). Fixes for these vulnerabilities, now identified as CVE-2023-29491 with a CVSS score of 7.8, have been successfully deployed by the maintainers of the *ncurses* library, Thomas E. Dickey, in commit 20230408. We wish to thank Thomas for his professionalism and collaboration in resolving those issues. We also worked with Apple on addressing the macOS-specific issues related to these vulnerabilities, and we thank Apple for their response and partnership. Lastly, during our analysis, a researcher named Gergely Kalman engaged us privately over Twitter and contributed relevant use cases in addition to his own hand-coded fuzzer. We thank Gergely for his contributions in advancing this research and community engagement. Users of *ncurses* are encouraged to update their instances and systems.

In this blog post, we share information about *ncurses* and the discovered memory corruption vulnerabilities. We also share this research to emphasize the importance of collaboration among researchers, industry partners, and the larger security community in the effort to improve security for all.

Understanding terminal databases

Terminal databases are used by *ncurses* to be terminal-independent, meaning the capabilities of the terminal are not required to be known ahead-of-time. Terminal databases contain a set of capabilities that ultimately determine the control characters that are sent to the terminal (instructing the terminal to perform basic interactions) and describe various properties of the terminal. Terminal databases come in two major formats: the older and less commonly used termcap (terminal capability) format, and the improved terminfo format. Since terminals can differ on the types of control characters they expect and the operations they support, terminfo became necessary to address this discrepancy. In its textual syntax, capabilities are separated by commas, and come in three forms:

- Boolean capabilities: for example, the *am* capability specifies that the terminal supports automatic margins. In the terminfo textual syntax, Boolean capabilities appear by their name alone, without any additions.
- Numeric capabilities: for instance, the *cols* capability contains the number of columns in a line. In the terminfo textual syntax, numeric capabilities are recognized with a "#" symbol after their name, followed by the numeric value, such as "cols#80".
- String capabilities: for instance, the *clear* capability describes the control character that should be transmitted to the terminal to clear the screen. In the terminfo textual syntax, string capabilities are recognized with a "=" symbol after their name, followed by the string value, such as "clear=\E[H\E[2J".

POSIX systems usually pre-ship with tens of such databases. It's possible to parse the capabilities of the current database with the infocmp utility:

Figure 1. infocmp output reveals the current terminfo database along with its capabilities

Environment variable poisoning

Every modern operating system contains a set of environment variables that might affect the behavior of programs. A well-known technique for attackers is to manipulate those environment variables to cause programs to perform actions that would benefit their malicious purposes, hence "poisoning" them. There have been multiple cases of environment variable poisoning in the past, for instance:

- CVE-2023-22809: users were allowed to elevate their privileges by poisoning the EDITOR environment variable (and similar other environment variables) and running sudoedit, which ultimately allowed them to edit arbitrary files.
- CVE-2022-0563: the environment variable INPUTRC is indirectly used by the chsh and chfn set-UID Linux binaries. It was discovered that INPUTRC could be poisoned to dump the contents of sensitive files on the system.
- CVE-2020-9934: the HOME environment variable could be poisoned to bypass Transparency, Consent, and Control (TCC) on macOS, thus gaining access to otherwise inaccessible sensitive data. We have found a similar bypass and reported it in 2021.
- CVE–2023-32369: the PERL5OPT and BASH_ENV environment variables could be poisoned to bypass System Integrity Protection (SIP) in macOS, thus elevating privileges. We have reported the vulnerability in April 2023.
- The LD_PRELOAD environment variable is commonly used in Linux for code injection purposes.
- The WINDIR and SYSTEMROOT environment variables have been used in the past on Windows for bypassing User Account Control (UAC).

We have discovered that during initialization, the *ncurses* library searches for several environment variables, including an environment variable similarly named TERMINFO. When using terminfo databases, the program consults a fixed directory path unless a TERMINFO environment variable is present, which instead points the program to an alternative directory that contains compiled terminfo database files. Moreover, there are interesting common programs that use *ncurses*, most notably *top* on macOS, which is a set-UID binary (which runs with elevated privileges) that also uses the TERMINFO environment variable. Therefore, finding vulnerabilities in *ncurses* have the potential to affect many programs and possibly elevate privileges. It's noteworthy that the potential of poisoning the TERMINFO environment variable was highlighted several times in the past (for example, here), but we have not seen comprehensive research on the topic of terminfo capabilities for offensive security purposes.

For completeness, while this blog post focuses on how attackers could poison the TERMINFO environment variable to potentially exploit *ncurses* vulnerabilities, the HOME environment variable could have been similarly manipulated. Assuming the TERMINFO environment variable was never defined, *ncurses* looks for a *$HOME/.terminfo* directory. This could have been abused by planting a *.terminfo* directory at an arbitrary path and poisoning the HOME environment variable, so the technique is quite similar.

Stack-based terminfo capabilities

The terminfo capabilities are richer than they first appear. In a nutshell, capabilities are allowed to receive up to nine parameters (p1-p9) and use them in a stack data structure. Furthermore, capabilities work with a stack-like structure and instructions that can push (place an item in the stack) and pop (get an item from the stack) data, perform logical-arithmetic operations, and even support conditions. Here are some examples:

Operation	Description
%{number}	Push a constant value to the stack.
%p_x	Push the parameter to the stack.
%+, %-, %*, %/, %m	Pop two numbers from the stack and push the arithmetic result of the stack. Addition, substruction, multiplication, division, and remainder operations are supported.
%&, %|, %^	Pop two numbers from the stack and push the bitwise result to the stack. Bitwise OR, AND, and XOR are supported.
%=, %<, %>, %A, %O	Pop two numbers and compare them, pushing the logical result back to the stack. The operations of comparison, less-than, and greater-than are supported, as well as logical AND and OR operations.
%l	Pop a string from the stack and push its length back to the stack.
%? [condition]%t[body$_1$]%e[body$_2$]%;	Perform a condition. The *%t* operation pops a numeric value from the stack and compares it to 0. The result determines what body to execute (the "else" body is optional and comes after the *%e* delimiter).
%s, %c	Pop a string from the stack and print it out to the terminal.
%d, %x	Pop a number from the stack and print it out to the terminal.

While not Turing-complete, terminfo offers functionality that resembles very basic programming. Due to the complicated logic required by *ncurses*, security issues are expected to be found, and indeed there have been numerous *ncurses* vulnerabilities in the past.

It's interesting to note that while the version of *ncurses* we checked was 6.4 (latest at the time of research), the *ncurses* version on macOS was 5.7, but had several security-related patches maintained by Apple. Nevertheless, all our findings are true for all *ncurses* versions, thus affecting both Linux and macOS.

Discovered vulnerabilities

We discovered several memory corruption vulnerabilities through code auditing and fuzzing. In addition to using our own AFL++ based fuzzer, the use cases contributed by Gergely Kalman assisted in advancing this research.

The discovered vulnerabilities could have been exploited by attackers to elevate privileges and run code within a targeted program's context. Nonetheless, gaining control of a program through exploiting memory corruption vulnerabilities requires a multi-stage attack. The vulnerabilities may have needed to be chained together for an attacker to elevate privileges, such as exploiting the stack information leak to gain arbitrary read primitives along with exploiting the heap overflow to obtain a write primitive.

Stack information leak

The function that runs the capability logic is called *tparm*. It is a C variadic function, meaning its number of arguments is not predefined (similarly to *printf*). The way variadic functions work in C is usually with the *va_list* structure and its macros, *va_start*, *va_arg*, and *va_end*. The common scenario for such functions is to parse a format-string, conclude the number of parameters it expects, and use the *va_arg* macro iteratively to fetch those arguments. However, since an attacker can be in full control of the capability's string, we can make *tparm* call *va_arg* more times than it should, effectively leaking information from the call stack. Since we are allowed up to nine parameters, we can leak up to eight unintended arguments, including arguments from the program's stack:

Figure 2. Demonstrating an information leak proof of concept

Parameterized string type confusion

The stack used by the *tparm* function is just an allocated array with 20 entries (referred as *STACK_FRAME* in the source code). Each frame can hold either a number (32-bit signed integer) or a string (pointer). To distinguish between a number and a string, the frame uses a Boolean value, which represents whether the data is numeric or not:

Figure 3. A terminfo stack entry

Certain push operations can be easily concluded, for example, when pushing an arithmetic result (such as *%+*) or a literal (*%{number}*). However, for parameters, things are different. There is no easy way to know ahead of time whether a parameter is expected to be a string or numeric. Therefore, *tparm* uses a heuristic—it walks the capability string statically, and when it sees *%s* or *%l*, it concludes that the last parameter push was a string. This approach can be abused in multiple ways. For example, the macOS *top* utility calls *mvcur*, which in turn calls *tparm* with the *cup* capability, along with integer parameters. Treating the parameter as a string can trigger *strlen* on the integer address:

Figure 4. Type confusion causes strlen to be invoked on low addresses

The crash we triggered occurs during an initialization of the *mvcur* operation, which assesses the "cost" of moving the cursor by invoking *tparm* with a constant, non-attacker-controlled value. We can improve the attack by using conditions—if the parameter's value is not that constant value, then treat the parameter as a string, otherwise treat it as a number. Implementation with capabilities is straightforward:

This should be read as:

Figure 5. Using conditions to only trigger strlen when desired

This primitive is quite powerful, as we can trigger *strlen* on an arbitrary number, effectively gaining a read primitive. Gaining a read primitive defeats the Address Space Layout Randomization (ASLR) security mechanism to leak address information and, if the binary happens to contain valuable secrets in its memory (like passwords), an attacker could potentially read those as well.

Cost calculating padding off-by-one

We have mentioned *mvcur* uses a cost-calculating function to determine the costs of certain capabilities. The cost-calculation is done by the function *_nc_msec_cost*, and it assesses the number of milliseconds it takes to print out a capability, which is strongly derived by delays that could be a part of a capability. Delays are numeric literal values wrapped between '*$<*' and '*>*', and they even support a decimal point. We discovered an off-by-one error—if the function sees a decimal point character, it skips one character assuming a digit, with an insufficient check after:

Figure 6. Off-by-one bug causes the string to be assessed beyond its boundaries

Therefore, it's possible to have the cost-calculating function read beyond the boundary of the capability string by closing the delay markup with a '>' character immediately following the decimal dot.

Figure 7. Reading past the capability string limit might cause a segmentation fault

Heap out-of-bounds during terminfo database file parsing

The terminfo database files are binary files commonly compiled from the text representation with a utility called *tic*. The format of the database consists of the following parts:

- The header: contains a magic value, the size of the terminal name, the number of Boolean capabilities, the number of numeric capabilities, the number of string capabilities, and the total size of string capabilities.
- The terminal name
- The capabilities:
  - The Boolean capabilities
  - The numeric capabilities
  - The string capability offsets
  - The string capabilities themselves
- Optional extended entries (in the same order: Boolean, numeric, and strings)

The optional extended entries are user-defined entries. We discovered that the function that performs that database parsing (*_nc_read_termtype*) can write beyond the boundaries of a heap-allocated chunk, as such:

Figure 8. Heap out-of-bounds due to realloc call

1. The code uses *calloc* to allocate room for the strings. While *STRCOUNT* is a constant representing the maximum length of standard string capabilities (414), *str_count* is attacker-controlled and defined in the header of the attacker's terminfo file. This controls the size of the allocated chunk saved in *ptr->Strings*.
2. After parsing all the standard capabilities, *ncurses* starts parsing the extended capabilities. The code assigns *ptr->num_Strings* to *STRCOUNT+ ext_str_count*, which might be **smaller** than the non-extended string count, effectively shrinking *ptr->Strings* with a *realloc* call.
3. Immediately after the *realloc* call, we can see *ptr->Strings* being written beyond its boundaries. Extended string capabilities are parsed and appended after standard string capabilities. The *convert_strings* function attempts to achieve this by storing data in *ptr->Strings + str_count*. However, while *ptr->Strings* was shrunk to *STRCOUNT+ext_str_count*, *str_count* is user-controlled and can be greater than *STRCOUNT*.
4. If *str_count >= STRCOUNT*, then *ptr->Strings + str_count + ext_str_count* will be greater than *ptr->Strings + STRCOUNT + ext_str_count* and *convert_strings* will cause a heap buffer overflow.

Denial of service with canceled strings

The *ncurses* library has a notion of marking strings as "cancelled". This is useful for terminfo database inheritance and skipping absent capabilities in general. As an example, the function *convert_strings* that converts strings from the terminfo database file format to the appropriate data structures in memory sets strings as *CANCELLED_STRING* if the index referring to them is negative.

Figure 9. convert_strings setting a string to be CANCELLED

The value of the *CANCELLED_STRING* constant is -1, and before processing, the *ncurses* codebase looks for these strings and converts them to *ABSENT_STRING* (constant 0). Unfortunately, it does so only for ordinary strings; extended strings do not get that treatment. Specifically, a heuristic determines that strings that begin with the "k" character should be treated as keypad functionality. This allows an attacker to specify an extended string in a way that will make *ncurses* dereference -1 (0xFFFFFFFFFFFFFFFF):

Figure 10. ncurses dereferencing -1 when attempting to parse a cancelled string for keypad functionality

Protection and detection with Microsoft Defender for Endpoint

While organizational devices and networks may become increasingly secure, attackers continue to exploit unpatched vulnerabilities and misconfigurations as a vector to access sensitive systems and information. Exploiting vulnerabilities in the *ncurses* library could have notable consequences for users, allowing attackers to perform malicious actions like elevating privileges to run code in a targeted program's context and access or modify valuable data and resources. Responding to the evolving threat landscape requires us to expand our expertise across devices and platforms as part of our commitment to continuously improve security *from* Microsoft, not just *for* Microsoft.

This case displays how responsible vulnerability disclosure and collaborative research informs our comprehensive protection capabilities across platforms. Microsoft Defender Vulnerability Management is able to quickly discover and remediate such vulnerabilities on both Linux and macOS. Additionally, Microsoft Defender for Endpoint has similar detections for potential abuse of terminfo databases for set-UID binaries, such as macOS's *top*:

Figure 11. Microsoft Defender for Endpoint detecting suspicious TERMINFO use

After discovering the vulnerabilities in the *ncurses* library, we worked with the maintainer, Thomas E. Dickey, and Apple to ensure the issues were resolved across platforms. Additionally, this case displays the value of community engagement to improve security for all as researcher Gergely Kalman's use case contributions assisted our research efforts. We wish to again thank Thomas and the Apple product security team for their efforts and collaboration in addressing CVE-2023-29491, as well as Gergely for his contributions in furthering this research.

As the threat landscape continues to evolve and threats across all platforms continue to grow, Microsoft strives to continuously secure users' computing experiences, regardless of the platform or device in use. We will continue to work with the security community to share vulnerability discoveries and threat intelligence in the effort to build better protection for all.

Jonathan Bar Or, Emanuele Cozzi, Michael Pearse

Microsoft Threat Intelligence team

References

- https://invisible-island.net/ncurses/
- https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2023-29491
- https://invisible-island.net/ncurses/NEWS.html#index-t20230408
- https://twitter.com/gergely_kalman
- https://linux.die.net/man/1/infocmp
- https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2023-22809
- https://linux.die.net/man/8/sudoedit
- https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-0563
- https://linux.die.net/man/1/chsh
- https://linux.die.net/man/1/chfn
- https://en.wikipedia.org/wiki/Setuid
- https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-9934
- https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-30970
- https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2023-32369
- https://developer.apple.com/documentation/security/disabling_and_enabling_system_integrity_protection
- https://man7.org/linux/man-pages/man8/ld.so.8.html
- https://attack.mitre.org/techniques/T1574/006/
- https://ss64.com/osx/top.html
- https://blog.trailofbits.com/2023/02/16/suid-logic-bug-linux-readline/
- https://www.cvedetails.com/vulnerability-list/vendor_id-72/product_id-38464/GNU-Ncurses.html
- https://github.com/apple-oss-distributions/ncurses
- https://owasp.org/www-community/Fuzzing
- https://github.com/AFLplusplus/AFLplusplus
- https://linux.die.net/man/3/tparm

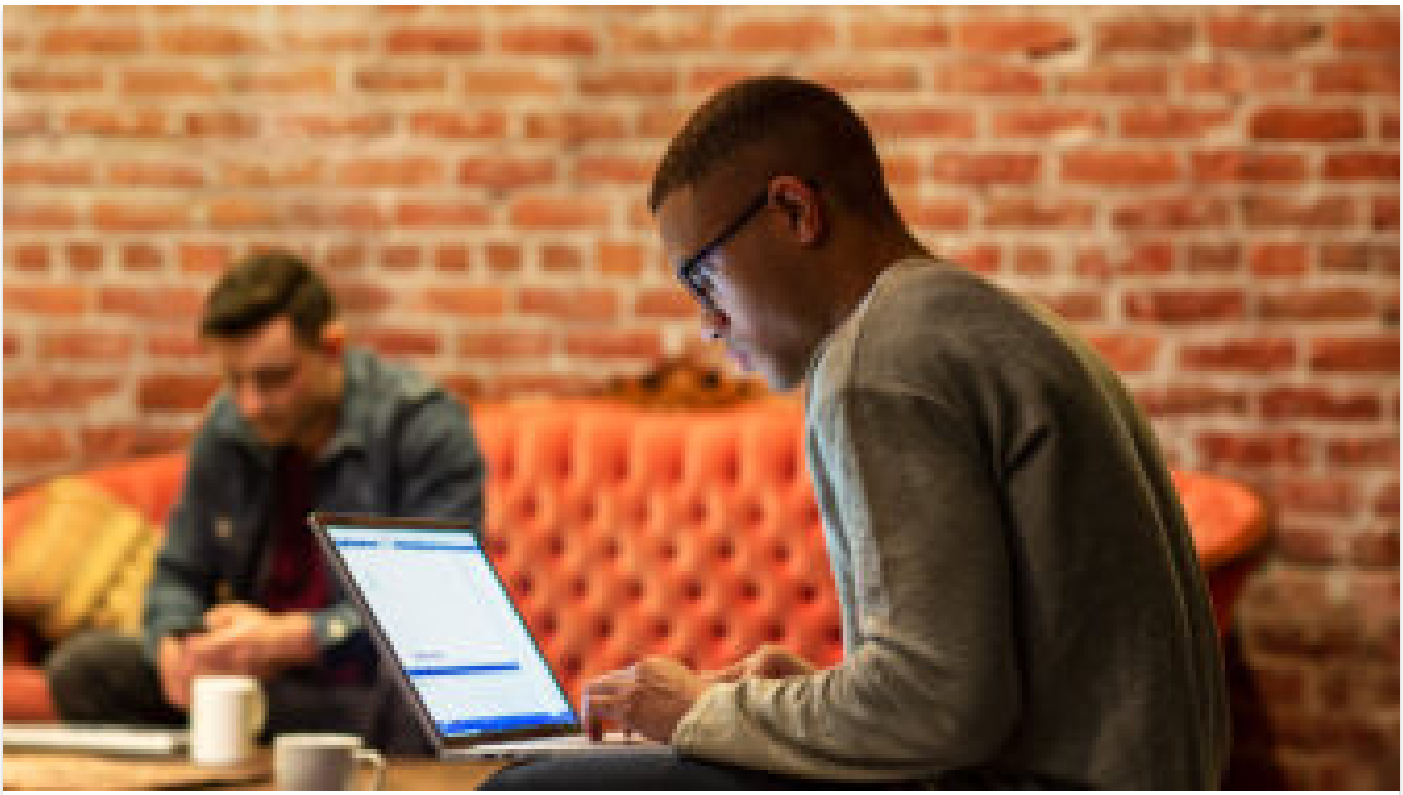
- https://en.cppreference.com/w/c/variadic/va_list
- https://en.cppreference.com/w/c/variadic/va_start
- https://en.cppreference.com/w/c/variadic/va_arg
- https://en.cppreference.com/w/c/variadic/va_end
- https://linux.die.net/man/3/mvcur
- https://cplusplus.com/reference/cstring/strlen/
- https://linux.die.net/man/1/tic
- https://linux.die.net/man/3/calloc
- https://linux.die.net/man/3/realloc

Further reading

For the latest security research from the Microsoft Threat Intelligence community, check out the Microsoft Threat Intelligence Blog: https://aka.ms/threatintelblog.

To get notified about new publications and to join discussions on social media, follow us on Twitter at https://twitter.com/MsftSecIntel.

Related Posts

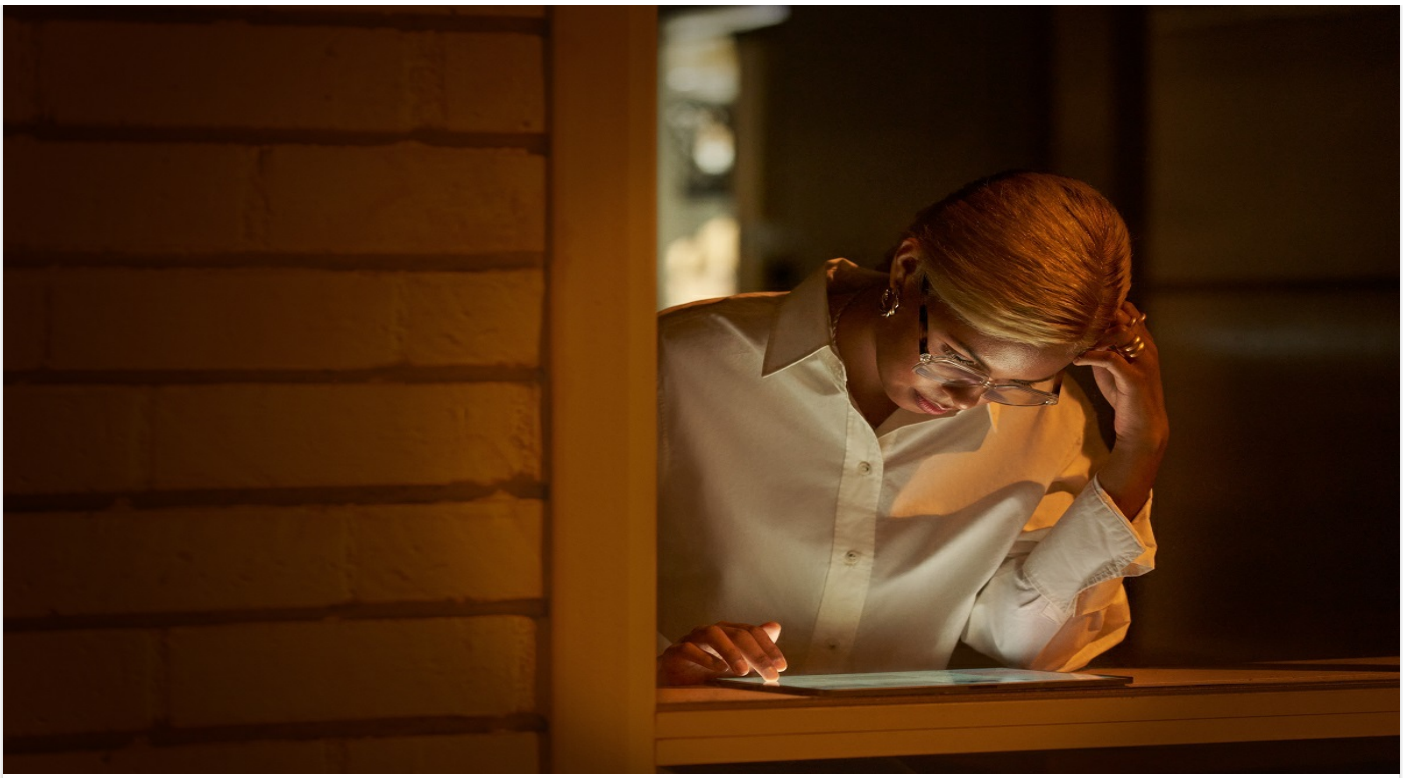


Research
Threat intelligence
Microsoft Defender for Endpoint
Vulnerabilities and exploits

May 30
10 min read

New macOS vulnerability, Migraine, could bypass System Integrity Protection

A new vulnerability, which we refer to as "Migraine", could allow an attacker with root access to bypass System Integrity Protection (SIP) in macOS and perform arbitrary operations on a device.



[Research]
[Threat intelligence]
[Microsoft Defender]
[Vulnerabilities and exploits]

Dec 19
9 min read

Gatekeeper's Achilles heel: Unearthing a macOS vulnerability

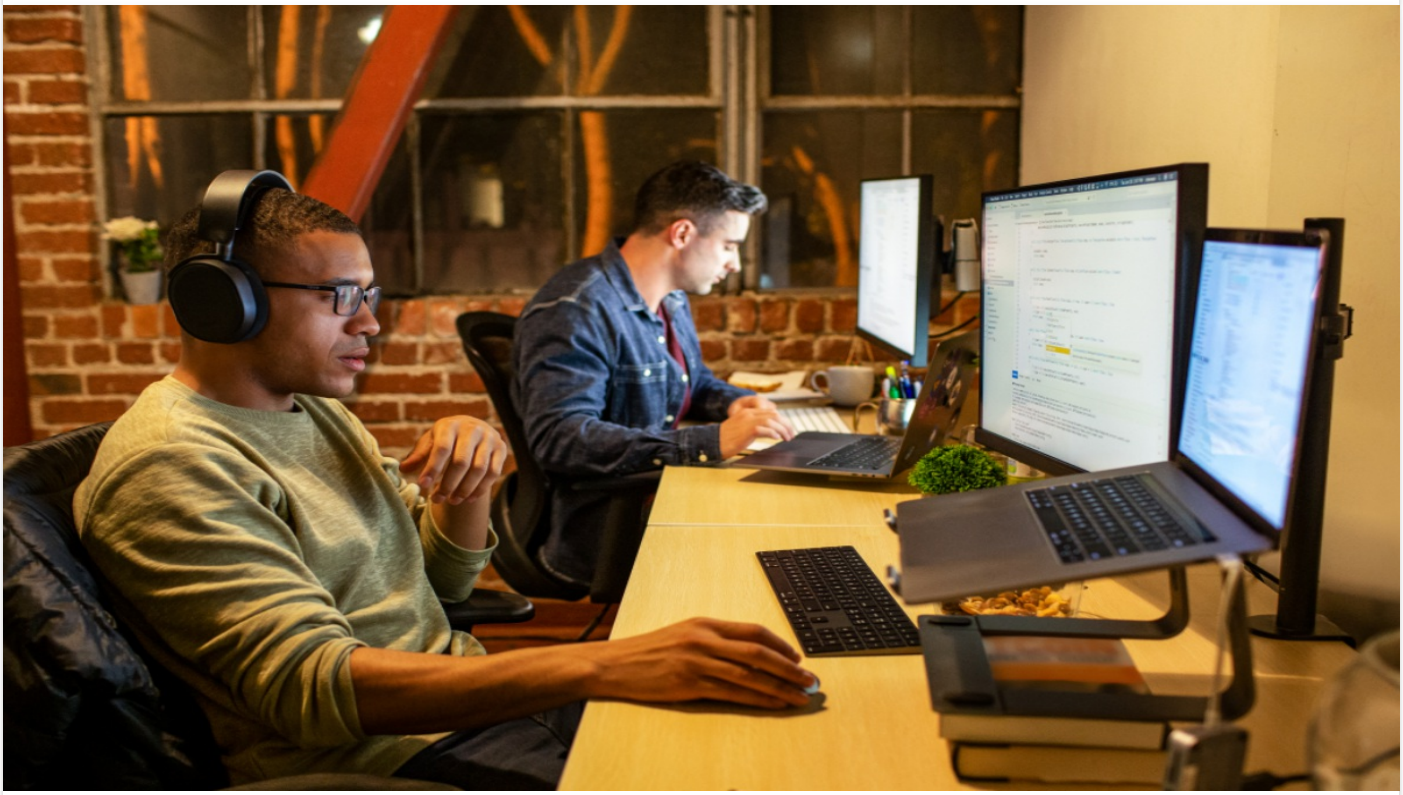
Microsoft discovered a vulnerability in macOS, referred to as "Achilles", allowing attackers to bypass application execution restrictions enforced by the Gatekeeper security mechanism.

Research
Threat intelligence
Vulnerabilities and exploits

Aug 19
8 min read

Uncovering a ChromeOS remote memory corruption vulnerability

Microsoft discovered a memory corruption vulnerability in a ChromeOS component that could have been triggered remotely, allowing attackers to perform either a denial-of-service (DoS) or, in extreme cases, remote code execution (RCE).



Research
Threat intelligence
Vulnerabilities and exploits

Apr 26
9 min read

Microsoft finds new elevation of privilege Linux vulnerability, Nimbuspwn

Microsoft has discovered several vulnerabilities, collectively referred to as Nimbuspwn, that could be chained together, allowing an attacker to elevate privileges to root on many Linux desktop endpoints. Leveraging Nimbuspwn as a vector for root access could allow attackers to achieve greater impact on vulnerable devices by deploying payloads and performing other malicious actions via arbitrary root code execution.

Get started with Microsoft Security

Microsoft is a leader in cybersecurity, and we embrace our responsibility to make the world a safer place.

Learn more

Connect with us on social

What's new

Surface Laptop Studio 2

Surface Laptop Go 3

Surface Pro 9

Surface Laptop 5

Microsoft Copilot

Copilot in Windows

Explore Microsoft products

Windows 11 apps

Microsoft Store

Account profile

Download Center

Microsoft Store support

Returns

Order tracking

Certified Refurbished

Microsoft Store Promise

Flexible Payments

Education

Microsoft in education

Devices for education

Microsoft Teams for Education

Microsoft 365 Education

How to buy for your school

Educator training and development

Deals for students and parents

Azure for students

Business

Microsoft Cloud

Microsoft Security

Dynamics 365

Microsoft 365

Microsoft Power Platform

Microsoft Teams

Copilot for Microsoft 365

Small Business

Developer & IT

Azure

Developer Center

Documentation

Microsoft Learn

Microsoft Tech Community

Azure Marketplace

AppSource

Visual Studio

Company

Careers

About Microsoft

Company news

Privacy at Microsoft

Investors

Diversity and inclusion

Accessibility

Sustainability

English (United States)

Your Privacy Choices

Consumer Health Privacy