How to find zeros

The current crisis has accelerated a trend towards a new normal where remote work is a rule rather than an exception. VPNs have long been a trusted and popular solution to gain remote access to company resources and while they have been trusted and popular, they may not always be the most appropriate to support and gain a changing workforce today. The full VPN idea* for all is like just having a hammer in your toolbox – various VPN flavours are important to consider alongside other technologies when improving remote access. However, the current 'new normal' shows traditional VPN limits, explaining that a new approach to gaining remote access is necessary. When the doorbell rings and it's a repair that comes to repair the electric pot, careful homeowners won't just hand over the house keys saying: the kitchen is the first door on the left, go ahead and help yourself, and then leave. They would stay and keep an eye on the service technician to check that he did his job, and they certainly wouldn't let him stroll around the house unattended. Yet when it comes to enterprise security, this is what happens: virtual private networks (VPNs), standard technology in many companies for giving users remote access to corporate resources, handing out 'keys to the palace' - once users log in, VPNs allow them to proceed without restrictions. Also, in the current crisis, many VPN arches struggle to perform under the burden of additional home office users. It is necessary to route all traffic through corporate data centers adding great latch and hurt the quality of time sensitive services such as video conferences. However, many mechanisms aimed at avoiding this effect make the VPN complex and expensive to manage. The danger of temporary unsecured beliefs approach to VPN castles is always problematic, it is more dangerous today. If you control and trust the entire network and VPN ecosystem from end to end, a full VPN is a solid option but if you do not control or cannot control the entire ecosystem, the full VPN introduces the risk of lack of vision and control. Attackers can target larger areas of attack today: when they manage to gain remote user credentials, or access to unsafe home office devices, traditional VPNs will give them a free pass to roam the company's network. Here they can find sensitive information and install malware such as data insertion tools or backdoors for easy returns. Belief involves more moving parts only VPN tunnels. Since the endpoint becomes a full participant in the network, there needs to be constant authentication of the right security tools, such as fire walls, IDS/IPS, AV tools etc. This leads to the complexity of management that leads to risks with various tools to manage and maintain. Obviously obviously must be a better way. And in fact, there is: it's called 'zero belief.' This new security approach adds a safety mind to IT architecture. Zero trust according to the principle: never trust, always verify. No user or device is considered reliable, regardless of whether they access resources from within or outside the network. For this, the first step is to know the user, ideally by using various authentication methods such as hardware tokens or soft token applications. Devices that connect to the network are inspected just as thorough, for example by checking ownership (owned by a company, privately owned) or whether the patch level is up to date. At the same time, company data is protected by limiting access to resource users actually in need of their role. Zero-trust solutionsToday zero trust uses machine learning (ML) to continue monitoring end-user activity and endpoints, comparing them to patterns of company behavior and policies. This allows security teams to quickly detect unusual activity that shows the accounts or threats of affected insiders. By giving warnings as soon as suspicious activity is identified, zero belief allows for a quick and highly targeted reaction. It significantly accelerates the incident response and shortens the time the attackers have to eye around the network. This approach - after years of 'bad guys' improved their tools and tactics while businesses and organizations were slow to respond - ultimately allowed companies to catch up on the security side, no matter where users are, or what devices they use. This makes it suitable for today's world where, accelerated by the crisis, distant work has become a new normal. The zero trust-based IT environment ensures that companies don't just hand over home keys to any 'repairman' bell rings. Instead, it will ask for repairs - and any other visitors - for a company badge with a photo ID. It will lock any door except the kitchen door, knowing exactly where the technician is and what he is doing. And if he behaves unexpectedly, it will automatically inform the homeowner. In this way, companies can always monitor users and devices, improve compromise detection, and downsize attack windows. At the same time, employees can access company resources safely – anytime and from anywhere. Darren Fields is Vice President - Cloud Networking EMEA at Citrix.We have featured the best identity management software. My wife carries the following story at any time she wants to make the point that I am pedantic: one of my daughters was in second grade, her math teacher told the class that any number divided by zero was one. I hit the emails targeted at the teacher, insisting that the results had to be defined. Supposedly this is proof that I'm sometimes hard to be around. Turns out that jokes may be turned on it is still difficult to support a second-class teacher's answer. I recently learned a bunch of things I didn't know about floating point maths:There's value for zero negative, separate from normal (positive?) zero. These two zeroes are defined equal to each other but they are different values.x ÷ 0.0, for x ≠ ±0.0, not error. On the other hand, the result is either positive infinity or negative infinity, following the convention of a common sign. The ±0.0 ÷ ±0.0 is an error (specifically it's not a number or NaN).–0.0 + –0.0 = –0.0, –0.0 + 0.0 = 0.0, and –0.0 × 0.0 = –0.0 RulesThese stems from IEEE 754 Standard for Floating-Point Arithmetic, which standardizes floating point representation across the platform. The latest version of the standard was completed in 2008 but the original version was released in 1985, so this behavior is not new. The above rules are true in both C (gcc) and Swift on my Mac, and also true at Swift on the iPhone. Python on Mac supports zero negative for floating, but throws exceptions when you try to divide by zero any sign. There are some surprising corollaries to this rule:Because 0.0 and -0.0 must be compared to the same, the tests (x & lt; 0.0) do not return true for each negative number - it fails to zero negatively. Therefore, in determining the zero value mark, you need to use the built-in functions on the platform, for example Double.sign in Swift. Or I guess you can bit-manipulate multiple raw representations, which is very much the answer of the programmer C. If a = b ÷ c, it does not necessarily follow b that = a × c, since this also fails to cases where c is zero mark either. I'm not a number theorist, but I find the concept above surprising. One immediate problem: Infinity is not a number, such as zero or 3.25 or π. On the other hand, infinity is a concept. It is true that rational numbers are about infinitable - but the infinity is not a member of the rational set of numbers. Furthermore, from a number theoretical perspective, division by zero is nonsense. You can understand why if you get exactly what the division means. Technically, division is a multiply by the inverse number, where the inverse is satisfactory: a × a^-1 = 1. Zero is the only number in the actual set of numbers that don't have a multiplicitative inverse. And because this inverse does not exist, we cannot go fold times worse by it. But surely the people who designed the floating point number know all this. So, I wonder about why the illustrated behavior came to be written into IEEE standards. To get started, let's consider the problem that the mathematics of the floating point is trying to address. The actual number could not be infinity, but we want to represent this whole set within the limits of computer memory. With a 64-bit double, there is 2^64 2^64 symbols, and standard designers of IEEE try to map these symbols into real-number sets in a way that are both useful for real-world applications and can also be economically implemented given the constraints of the early 80s silicon. Given the basic requirements, clear estimates will be applied. The reason for zero negative appeared to date to the 1987 paper [1] by William Kahan, a Berkeley professor who was considered the floating point father and who later won the Turtle Award for his work in drafting the IEEE 754. It turns out that zero negative existence is tied to the ability to divide by zero. Let's start by discussing the usual reason that division by zero is not allowed. A naive approach to division by zero is the observation that:In other words, since x gets smaller, the result of 1/x gets bigger. But this is only true when X approaches 0 from the positive side (which is why there is a little plus sign above). Conducting the same thought experiment from the negative side:As a result, the generic limits of 1/x as x 0 approach are not defined, since there is inconsistency (what Kahan calls slit) in functions 1/x.However, by introducing signed zero, Kahan and the IEEE committee can work around difficulties. Intuitively, zero marks are being taken to indicate the direction of the limit is being approached. As Kahan stated in the 1987 paper:Instead of thinking of +0 and -0 as a different numerical value, think of a bit of their mark as an additional variable that conveys one bit of information (or incorrect information) about any numerical variable that takes 0 as its value. Usually this information is irrelevant; value 3+x is no different for x := +0 of x := -0.... However, some unusual arithmetic operations are affected by zero marks; for example 1/ (+0) = +∞ but 1/ (-0) = –∞.I have made my peace with the concept by practicing the rationalisation proposed by my partner Mike Perkins: The 2^64 symbol available is clearly insufficient to represent the entire set of real numbers. So, IEEE designers set aside some of those symbols for special meaning. In this sense, ∞ does not mean infinity - on the contrary, it means a larger actual amount than we can otherwise represent in the symbol of our fixed floating point. Therefore + 0 doesn't really mean zero, but the actual amount is larger than true 0 but smaller than any positive number that we can represent. Incidentally, while scrutinising the issue, I found that although Kahan did not like the negative idea of zero:Signed zero - well, the zero signed was the pain in the ass that we could if we use projective mode. If there is only one infinity and one zero you can do just fine; then you don't care about zero marks and you don't care sign of infinity. But if, on the other hand, you insist on what I would be considered a lower option than two infinity, then you will end up with two zero signed. There's really no way around that and you're stuck with it. (From a Kahan interview conducted in 2005.) I'm not sure if writing a blog post ten years later makes the rails against a poor second-grade teacher. For my daughter's part, now in high school, just roll her eyes when I start talking about division by zero at dinner. So perhaps that the hard thing to be around is speed. [1] Kahan, W., Branch Cuts for Complex Basic Functions, or Much Ado About Nothing's Sign Bit, State of Art in Skeleton Analysis, (Eds. Iserles and Powell), Clarendon Press, Oxford, 1987, are available here. Join the Noon Hacker Create your free account to unlock your custom reading experience. Experience.