computers are bad

You are receiving this facsimile because you signed up for fax delivery of this newsletter. To stop delivery, contact Computers Are Bad by email or fax.

https://computer.rip - me@computer.rip - fax: +1 (505) 926-5492

2021-05-01 simple as in snmp

Very early on in my career as an “IT person,” when my daily work consisted primarily of photocopier and laptop warranty service with a smattering of Active Directory administration (it was an, uh, weird job), I was particularly intimidated by SNMP. It always felt like one of those dark mysteries of computing that existed far beyond my mortal knowledge, like distributed algorithm optimization or modern JavaScript.

The good news is that SNMP is actually, as the name suggests, quite simple. The reason for my SNMP apprehensions is a bit silly from the perspective of computer science: SNMP makes extensive use of long, incomprehensible numbers. That is, of course, basically a description of all of computing, but SNMP exposes them to users in a way that modern software generally tries to avoid.

Today, we’re going to learn about SNMP and those numbers. Surprise: they’re an emanation of an arcane component of the OSI stack, like at least 50% of the things I talk about.

But let’s step back and just talk about SNMP at a high level. SNMP was designed to offer a portable and simple to implement method for a manager (e.g. an appliance or administrator’s workstation) to inspect the state of various devices and potentially change their configuration. It’s intended to be amenable to implementation on embedded systems, and while it’s most classically associated with network appliances there is a virtually unlimited number of devices and software packages which expose an SNMP interface.

SNMP often acts as a “lowest common denominator:” it’s a simple and old protocol, so just about everything supports it. This makes it very handy for getting heterogeneous devices (especially in terms of vendor) into one monitoring solution, and sometimes allows for centralized configuration as well, although that gets a lot trickier.

At its core, SNMP belongs to a category of protocols which I refer to as remote memory access protocols (this is my taxonomy and does not necessarily reflect that of academic work or your employer). These are protocols which allow a remote host to read and (possibly subject to access controls) write an emulated memory address space. This does not necessarily (and often doesn’t) have anything to do with the actual physical or virtual memory of the service, and the addressing scheme used for this memory space might be eccentric, but the basic idea is there: the “server” has memory addresses, and the protocol allows you to read and write them.

These remote memory access protocols, as a category, tend to be very common with embedded systems because if they *do* happen to align with physical memory, they are very simple to implement. A prominent example is Modbus, a common industrial automation protocol that consists of reading and writing registers, coils, etc., which are domain-specific terms for addresses in the typed memory of PLCs (historically these were physical addresses in the PLC's unusually structured memory, but today it's generally just a software construct running on a more general-purpose architecture).

Unsurprisingly, then, the basic SNMP "verbs" are get and set, and these take parameters of an address and, if setting, a value. On top of this very simple principle, SNMP adds a more sophisticated feature called a "trap," but we'll talk about that later. Let's call it an "advanced topic," although it's actually one of the most useful parts of SNMP in practical situations.

What is perhaps most interesting to consider, as far as arcane details of SNMP, is the structure of the addresses. This is the scary part of SNMP: just about the first time you have to interact directly with SNMP you will encounter an address, called a *variable* or more properly *object identifier (OID)* in SNMP parlance, like .1.3.6.1.4.1.140.305. It's like an IP address, if they were substantially less user-friendly. That is to say, an IPv6 address [1].

These OIDs are in fact hierarchical addresses in a structure called the Management Information Base (MIB). The MIB is an attempt to unify, into one data structure, the many data points which could exist across devices in a network. This idea of a grand unification of the domain of knowledge of "configuration of network appliances" into one unpleasant numbered hierarchy has a powerful smell of golden era Computer Science with a capital CS, and indeed it is!

You see, from a very high level, the MIB is actually viewed as something akin to a serialization format--it is, after all, fundamentally concerned with packing the state of a device (Management Information) into a normalized, strictly structured, interoperable format. To achieve this, the MIB is described using something called SMI (e.g. RFC2578), which is best understood as a simplified (or perhaps more formally "constrained") flavor or ASN.1.

ASN.1 is the most prominent of the interface description and serialization formats developed for the OSI protocol suite. You might be tempted to call ASN.1 an example of the "presentation layer," although like most invocations of the OSI model, you would be misunderstanding the OSI model in saying so (the OSI presentation layer protocols are, as the name suggests but is often ignored, full on request-reply network protocols, not just serialization formats). Nonetheless, people say this a lot, and at least ASN.1 truly dates back to OSI, unlike a lot of things people relate to the OSI model.

You might be familiar with ASN.1 because it is widely used in cryptography, and by this I mean that cryptography applications are widely saddled with ASN.1. Most cryptographic certificates, the formats we tend to variously (and confusingly) call X.500, PKCS#11, DER, PEM, etc, are ASN.1 serialized. This is a whole lot of fun since ASN.1 is significantly divergent from modern computing conventions, including the use of length-prefixed rather than terminated strings (in some cases). I bring this up because it has lead to a rather famous series of vulnerabilities in TLS implementations, because apparently not even the people implementing TLS have actually read the ASN.1 specification that closely.

Anyway, back to SMI. Basically, SMI allows vendors of devices (or anyone really) to write, in SMI, a description of an MIB "module." A "module" is basically a list of OIDs (hierarchically structured) with their types and other metadata. This SMI source is then compiled into the binary representation actually used by SNMP clients. If you are unlucky, you may need to write SMI yourself for devices whose vendors implemented SNMP but did not provide the supporting materials. But, in most cases, device vendors provide a file (commonly called an MIB file) which is the SMI description of the MIB module(s) implemented by the device. This MIB file can then be fed to your SNMP tool to be compiled into its “whole picture” binary MIB.

Knowing that it is a result of compiling together SMI produces by various vendors, let’s take a look at the structure of the MIB. Each dot-separated number identifies a subtree, which for extra fun are called “arcs” in the context of the MIB. At the very top of the OID hierarchy is a top level which identifies the standards authority. This is 0, 1, or 2, which refer to ITU, ISO, and ITU/ISO together, respectively. Of course these three parts of the tree use different internal structures so I can’t generalize past this point, but I will focus on the ISO tree because it’s the one most commonly used in practice.

Under the .1 ISO hierarchy are arcs for ISO standard OIDs, registry authorities (somewhat difficult to explain and also not widely used, basically a metadata space), ISO member organizations by country (e.g. ANSI in the US), and then identified organizations, which are just companies and organizations that have asked for OID space. This can be somewhat confusing because many national ISO member organizations also allocate OID space within their arcs, but major vendors (e.g. Cisco) are often found at this top level instead.

So let’s take a look at a somewhat arbitrary example, an MIB for Juniper’s Junos. I’m using this as an example rather than the more obvious Cisco IOS because I got mad at Cisco’s website for getting MIBs which did not appear to have seen an update in a decade. In any case, the MIB starts out at .1.3.6.1.4.1.2636.

In terms of the hierarchy this means: ISO standard, identified organization, DOD, internet, private projects, private enterprises, Juniper.

Haha, wait, that just goes against most of what I said. What’s going on with the DOD thing?

The entire Internet, big-I, TCP/IP world is considered to be a subset of the DOD, for OID purposes. This .1.3.6.1.4.1 space is actually managed by IANA, and if you would like your own .1.3.6.1.4.1 number they will be happy to give you one upon application.

This is all particularly interesting historically, because unlike a lot of protocols I talk about SNMP does *not* predate IP. It was designed specifically for use on IP networks, over UDP. SNMP is based on several earlier protocols also used with IP. So, where does this weird rendition of IP to a small subset come from?

Well, it really has more to do with politics than technology. The MIB tree essentially belongs to ITU and ISO, but ITU and ISO are both organizations which are not especially known for swiftly and cheaply adopting standards proposed by vendors. It was fairly obvious from an early stage that vendors would need to produce MIB modules for their own devices fairly quickly, but ISO and ISO member organizations were not especially enthusiastic about issuing a large number of arcs to these vendors. So instead, IANA stepped in--but not quite IANA yet, instead IANA’s predecessor, Jon Postel. Postel, who *was* the IANA for quite some time, worked on contract for DOD, and so he assigned OIDs out of their space. There’s no really good reason for it to be this way, but if you work with SNMP a lot then typing .1.3.6.1.4.1 will have become basically reflex.

Now, what is found inside of this Juniper space? Well, for example, there's .1.3.6.1.4.1.2636.3.58.1.2.4.1.3. This is an integer value which provides the average power used, in watts, by whatever's plugged into a particular outlet of a managed PDU. The MIB structure allows OIDs which contain other OIDs (object identifier type OIDs) to actually contain *tables* of those OIDs, so .1.3.6.1.4.1.2636.3.58.1.2.4 is a table of all of the outlets on the PDU, and .1.3.6.1.4.1.2636.3.58.1.2.4.1 within it is a list of useful properties of the outlet such as name, status, and various useful electrical measurements like current and power factor.

After all of this talk of ASN.1 and MIBs and etc, these examples are actually very useful and concrete. SNMP is, after all, actually a useful protocol for real-world situations, such as centralized monitoring of your PDUs to identify problems and catch your colo customers exceeding their power budgets.

And remember, SNMP even allows writing. So .1.3.6.1.4.1.2636.3.58.1.2.4.1.8, the status of the outlet, can not only be used to determine whether the outlet is on or off but also to turn the outlet on or off, which is a fun move when your colo customer doesn't pay their bill for months.

SNMP is not limited to as concrete of devices as managed PDUs. For example, RFC4113 provides an MIB for UDP. That is, it permits you to describe and modify UDP messages using SNMP, if that's a thing you really want to do. In fact, the entire concept of the MIB is far more general than SNMP, and ISO protocols and standards often use MIB OIDs for identification purposes having little to do with the application we're discussing here. For example, many MIME types have an associated OID because the OSI email equivalent, X.435, uses OIDs to identify the types of message parts. In general, OSI standards are lousy with OIDs used as identifiers and, less frequently, to describe data structures and field sets.

The fact that you can set via SNMP, and get all kinds of potentially sensitive questions, raises the concern of security. Fortunately, SNMP provides an airtight solution to this problem: "communities." A community is really just a shared password, if the SNMP manager has the same community string as the SNMP agent then it is allowed access. Even better, many SNMP agents have well-known default community strings. Perfect. To be fair, SNMPv3 adds a more rigorous authentication support including support for different authentication methods, but there are still plenty of SNMPv2 devices out there with community string set to "public."

One final thing to complete our discussion of SNMP is to mention the trap. More technically, I am going to conflate traps and inform requests which are actually slightly different, but everyone conflates them so I feel okay about it. A trap is an extremely useful feature of SNMP which allows you to configure an agent (e.g. device) to immediately inform a manager when certain events occur. This is essentially a basic alarm capability built in to many devices. Traps are identified by OIDs, and can bind other OIDs, so that the generated trap message includes not only which trap was triggered, but also some other related data if so configured. To be complete, an inform request is really just a trap where the agent acknowledges receipt (this is not the case with normal traps) so that the agent can resend if it is not acknowledged.

In order for traps to work, the manager first needs to listen for traps, which is usually fairly straightforward to set up. Then, various OIDs are set on the agent to enable traps and set the destination for those traps (e.g. the IP of the manager). In some cases agents also provide a web interface or other more convenient mechanisms to set these up, which is much appreciated since SNMP is unpleasant to have to think about directly.

That’s about it for SNMP. Simple, right? Well, it really is pretty simple, as long as you agree to just take OIDs as magic numbers that come from wherever it is computers do and not ask too many questions. Where SNMP can become rather rough is when you run into issues with MIBs, or if you are using SNMPv3 where the authentication and configuration can be amazingly, maddeningly complex for some vendors.

As an aside, the whole reason I’m talking about SNMP is because a reader asked me to. For much the same reason, from the same reader, I’ll be talking about LDAP soon. LDAP is even more an out-of-place artifact of OSI than SNMP, and it is basically impossible to describe as used in short form, but I will take a shot at illustrating the odd historical components of LDAP and the ways they matter today. It will at least serve as a teaser for my yet to be written book, “Survival Under LDAP.” LDAP *is* survivable for as many as 70% of Americans, but you must know how to protect yourself!

[1] I continue to seriously question the merits of the complex address representation used with IPv6. If we had stuck to decimalized bytes separated by dots, we’d be doing a lot more typing, but we wouldn’t be trying to remember what :: means when it’s *there*.