All.Net Analyst Report and Newsletter

Welcome to our Analyst Report and Newsletter

Overwatch: Careful what you watch for... you may see it.

This article was written for our Security Operations Center course, but I thought I would share it with everyone else because… why not?

I made a mistake

Of course we all do, and most of them are not so bad that it matters. But this one looked like it might. We run servers that, among other things, support services for other companies. One of those servers was experiencing slow response times – and when I say slow, I mean perhaps 10% of the normal pace of user experience. Enough to make it worth bothering to fix it. No user problems were yet reported, but we knew that there would be some soon.

The slowdown became apparent in the morning of a day I had customer meetings, and for the next, perhaps 7 hours, I was in those meetings almost constantly, online of course, and while there, I was also doing things during short breaks to try to figure out what was going wrong.

Don’t Panic!!!

Of course it crossed my mind that this could be a denial of service attack or some such thing, because services were being slowed and close to denied… so I wanted to make certain it was not just happening to my infrastructure. I checked a few things from a few places and found that there was a substantial increase in reported problems with the infrastructure over the last few hours, so I figured it might be that, but “when in doubt, check it out”.

Of course if the service provider is the problem, there isn’t much I can do about it other than go to backup services, which I only do for sustained outages. I’m not running a real-time transaction environment for warfare, a time critical OT infrastructure, or anything like that, so no need to rush things.

I logged into the server that was going slow and it seemed like it was… going slow. I looked at the processes and such, and nothing unusual was apparent there. I wanted to make certain there was no data loss, so I checked the backup processes and they were running reasonably slowly but running nonetheless. I made a backup of the backups in case something went wrong in more than one place or if there was a subversion of some sort, and to make certain we could recover in case we needed to.

Shedding load

My next move was to shed some of the load off of the offending server. We run about 20 domains, most of which are not very interesting, but a few of which have multiple host names used for different purposes. The Web services for these domains share servers, so each server might have 20 or so host names operating on them at any given time. We do backup and restore to secondary servers on a semi-regular basis (some fully automated, others not). This allows us to do load balancing at a slow pace, and not automatically, by changing what servers run which services for which hosts in which domains. Not all servers can perform all services because different commercial providers provide different services for different prices.

In order to relieve any pressure on the server with the problems (and assure that users were only impacted if they needed services from that server), I went to change some of the A records for host names in the Domain Name Servers (DNSs) to point to other systems. Now over the years, I have kept my DNS Time To Live (TTL) at a few minutes (5 is usually OK for the sorts of slow changes I tend to make). However, when Google threw out its domain services and tossed them over the wall to Square Space, Square Space changed all the TTL values to 4 hours, which means that an urgent domain name to IP address switch takes 4 hours to get through the Internet to all of the servers that might be used to access any given host name. As a result, you have to keep the services running for the entire 4-hour switchover time on both servers (the original and replacement) and state information changes due to user usage end up, in some cases, not reflected on the new server until you do a backup and restore, or if you run real-time redundant storage across multiple cloud service providers (which none of them really support for the prices I am willing to pay).

So, changing the DNS tables was undertaken, the TTL values were set to the minimums for the DNS provider along the way, and services that were easily moved without state-related problems were shifted to another server, new SSL certificates were generated for the new server for the host names (because the different providers do things different ways this was easier than trying to send all the certificates to the right places manually), and users would be shifted as their DNS servers had to lookup the relevant host names.

In the meanwhile, in order to remove the slow performance from shifting host names, I changed the javascript in the include files used on the shifting domains so that services were directed to explicit servers that were running normally instead of operating on whatever server you came in on. That reduced the potential impact on users to largely limit the slowdown to a fairly small set of users using highly server-based interactions (as opposed to those running applications that merely download from the server then run locally in the browser). None of this mitigated the slowness of the server at issue, so it was none of those things...

Make sure the backups are working right

Next I decided to bring up the backup server (which is kept on cold standby rather than risk it being contaminated in an infective attack), and updated the content to reflect the current data state (not the operating environment which is kept separate and different in case there is a supply chain or common mode failure). When I checked it out, it was running at normal speed, and since this backup server runs through the same infrastructure as the primary, I concluded it was not the infrastructure or anything on the systems I was testing from. Thus the problem was, presumably on the server running the primary service. I then also shifted a few services to the backup server and started a higher paced backup of changes from the primary server and restoration of those changes to the backup server so that no more than perhaps 15 minutes of user-related data changes would be lost in the worst case.

Fix the server?

Now that I knew it was a problem on the server, I set about drilling down further, figuring someone might have gotten in and corrupted something, but wanting to be careful about it and not jump to conclusions. I tried a variety of simple things to find the obvious, but it turned out not to be obvious, so eventually I decided to start looking at the logs. My process, good or bad, started by going to my internal logs, which yielded nothing of interest, followed by going to the system logs in /*var*/log and looking at the change dates and sizes of the log files (ls -l).

Ahah! Doing a tail on the log files that were being updated more recently and actively, I saw that two of them were updating quite often, more often that I figured they normally should, so I did a ‘tail -F’ on them to watch as they logged stuff. With that being done I had a ringside seat to what was going on. The system was being repeatedly, and at a pace of about one per second, logged into remotely over an encrypted channel. Better yet, these were legitimate authentications. They were getting in, but then not doing anything substantial, and doing that often enough that it was pretty not good.

Not the server...

The server was going slow because the remote authentication process and key setup for session keys was taking enough CPU cycles and memory to bump up the load average, even though it was not producing any unusual usage patterns or abnormal processes. It was just sustained so long that memory cache to disk was getting involved, and so it goes. When you move customer services into a mode where things no longer stay in cache, it slows by a factor of 8-10 as it thrashes cache memory into and back from RAM, and worse, from there in and out of disk. This persisted after reboots as well, because it was a remote system invoking the server repeatedly, all through legitimate channels and means, with nothing was going at a pace over the threshold of normal behavior for these interactions. It was just happening for a sustained period and over a few minutes after reboot, built up to cause the slowdown.

Unfortunately, the source was not apparent because the systems initiating the communication are behind Network Address Translation (NAT) gateways. So I had to go behind the gateways to debug further. There are redundant gateways involved including double NATs, and while we can sniff traffic on the various network segments if we have to, I figured it would be easier than that to simply go into the systems and check what they were doing to find the offender.

So one after another I checked the automated processes, and nothing was out of order. But I stopped them to make it easier to see what was left causing problems. Of course this means remembering to restart them later, which I would have forgotten if I wasn’t keeping track.

Don’t watch the watchers too hard

As it turns out, for visibility into the systems in the network, we have periodic processes that we can look at from one of several locations, and these processes that were watching things were not helpful for the purpose. They more or less showed everything operating normally. No disk usage problems, no network traffic problems, no interface problems, and on down the line. Everything I could see was operating normally.

So I figured it must be something I couldn’t see. Back to the sniffers. I regularly run sniffers that look for new things and there was nothing new. We regularly automatically have hundreds of encrypted channels open per minute for users, for processes that happen every once in a while, and nothing was operating over those thresholds. If we lower the thresholds we will start to see lots of irrelevant stuff, and ignoring what we think is irrelevant (selective blindness) to focus more closely at things not yet excluded was not helpful. Eventually I ignored everything and there was nothing abnormal to see. That’s because it was operating within the normal operating parameters and mixed in with other normal things.

Sometimes, often, you cannot see the trees for the forest. And if this were malicious, I would be seriously concerned about infiltration. So I started to think about advanced persistent threat actors in my internal infrastructure as a possibility.

And then it came to me

The watchers had gone through some updates recently, and among the updates were ‘improvements’ in looking at network interfaces. However, there were also software updates on some of the systems, and the software updates updated to a new old version of a systems program that was failing because an option in the software didn’t exist in all the versions.

We try to avoid updates because they tend to cause things that work to fail. That’s because folks out in the Internet keep ‘improving’ things by ignoring backward compatibility, and as a result, they keep breaking things for everyone else. We try to not be broken as often by not doing updates as often. And when we do do updates, we then need to go through an incomplete regression testing process. Then when we find a problem and fix it, we have to make sure the fix didn’t break something else, and then go and update all the other affected systems and make sure it didn’t break anything else on those systems by testing more.

I especially dislike ‘security’ updates because they tends to break more things by adding some layer somewhere, and I cannot think of more than one such update in the last 15 years that I actually did because of an identified problem that could actually be invoked by a threat actor if they got past our other protective mechanisms. Mostly they effect things we just don’t use or load on our systems, so we avoid these sorts of problems (updates) wherever feasible.

Back to the failed execution

None of the mechanisms we were then using were related to the failed execution, as it turns out, but as it also turns out, I don’t reboot my systems very often. I prefer once every few years, but some systems just crash after a few months, in some cases because hardware has internal counters that overflow or some such thing, but usually because of software flaws.

One of my redundant systems used for recovery and surveillance while waiting to recover is also an R&D system, and recent updates to try to automatically continue the surveillance processes after systems under surveillance crash or get rebooted had a bug that was subsequently fixed. When I say ‘had a bug’ that means I screwed up.

After fixing the bug, everything seemed to be working, and rather than reboot the system, which we rarely do if we don’t have to, I just kept on going. The ‘bug’ was in a mechanism that restarted surveillance when a remote system failed. The failed execution on the remote system because of the update on the remote made the watching processes end prematurely, and the restart mechanism initiated multiple sessions that ended up slowing down the server. It was only noticed long after the original bug was fixed, because of the server update, which I guess we never should have done in the first place. All I had to do was reboot the surveillance system to stop an old unrepair process loop, and the symptom would disappear. However...

So I updated the updated the remote to revert to mechanisms that worked, the problem stopped, and everything came back to normal in a few minutes. Then I rebooted the system doing the surveillance to stop the old method from continuing to operate in the background in favor of the new method, and I think it’s all OK now… until something else gets updated.

Conclusions

There really is no big conclusion here. It’s just a typical thing that happens in networks today that I thought I would share with the class and you. We see what we want to see, and we selectively decide what we want to see when we implement surveillance systems.

After action adaptation

The incident handling process documented in our approaches includes deter, interdict, prevent, detect, react, and adapt. These operate across the protection objectives, which at a high level include integrity, availability, confidentiality, use control, accountability, transparency, and custody. In this case, it was:

- Human detection of reduced availability caused by integrity loss from an update
- Interdiction by rerouting custody to alternative resources thus changing use controls
- Reaction using accountability and transparency (visibility and audit trails) to investigate
- Adaptation by restoring integrity, and the process improvements detailed here.

We run at managed maturity, which ultimately means we try not to make the same mistakes again by putting process in place to measure and manage. The interdiction and adaptation in this case happened in real-time by the load shifting and fixing the DNS TTL values which we missed in some cases in the forced conversion from Google to Square Space. The use control URL changes in the include files will end up getting changed (back or not) again, and the re-balancing of load across servers and types continues over time regardless.

This report is part of the after action adaptation process, although this particular write-up is more for the student and external audience than for internal purposes. The internal changes associated with this process are fairly small. We updated our playbook for system setups to update the system software components to reflect the changes needed to fix the broken vendor updates, and we deployed the updated setups and tested it across relevant systems.

We are in the process of updating internal infrastructure adding additional redundant NAT gateways and trying to automate shifts from ISP to ISP when one ISP goes down, and we are considering whether to permanently install the digital diode sniffers in key locations or continue to only place them when needed. This is part of another ongoing effort to do real time surveillance of all sides of network segment separation mechanisms for traffic differentials to verify that firewall rules and other limitations in place are actually working by a redundant and independent mechanism.

We are also looking at the backup processes which, after the mass consolidation of all versions of all backups documented in our 2025-01 article (Build Backups Better) started to become fragmented again. The automated restore to the standby system is problematic when it is a cold standby, and takes more time than I would prefer. But on the other hand, we don’t want that system to be up all the time because then it is susceptible to attack all the time. I think we will go to a more frequent testing regimen where we bring the cold standby up once a month or once a quarter, do the restoration for changes, do minimal testing, and then shut it down again. The idea is that it will reduce the recovery time because fewer things will get changed in the restoration process, which does differential restoration (using rsync).

The other issue is the network surveillance system which is too ad hoc for the long run. Of course there is also a key balance here. Very little ever actually happens in our internal networks, and we haven’t had an actual attack penetrate these networks as far as I can tell since we started running them in the mid 1990s. Over the years we have tried various surveillance and integrity approaches to see if we are missing something, but we tire of watching for things that don’t happen. So we document our mistakes, try not to make them again, try to share what we learn with others, and read what they choose to share with us.

