Can you tell us a bit about yourself ? Where you live, your job, whatever you think important to present you to the french community

I grew up in a small town in sweden, but moved to Gothenburg (which is still small when compared to the large cities in the world) when i was 19. I started programming when I was seven, and once in high school me and some friends wrote a game that crashed the entire school network. I'm still sort of proud of that actually :)

In 2000 I started skydiving, and through that exceptionally great hobby I came into contact with my good friend Johannes, who hired me to do some consultancy development work for op5. I did a pretty good job apparently, because a month later I was at a job interview with the founders of op5. I think they had already decided beforehand that I should get the job, because due to an unfortunate incident with an incontinent cat in Barcelona (where I had been vacationing just before the interview), I came to the interview dressed in a sweaty sarong, a dirty tunic and flipflops. Everything else stank of catpiss, as one of the many homeless cats in spain had gone into our room and relieved itself in my suitcase two hours before the plane departed.

What is,was your implication, role with The Nagios Core project ?

I first came into contact with Nagios in 2004, when I started my job at op5, and one of my responsibilities was to make sure that it worked well for op5's customers (we had about three back then, AFAIR). I did just that, and sent in a bunch of patches and some ideas for future features. Back then, Ethan was active and pretty fun to work with, although he didn't like to share his code before he had completed it to his satisfaction.

The Nagios project stagnated pretty hard around 2007, just after Nagios 3 was out. Ethan claimed the project was feature-complete and that development would be restricted to bugfixes only. Since there were close to a hundred different patches floating around on the various mailing-lists, I had a hunch I wasn't the only one to disagree with that statement. I started taking care of patches and making sure patch contributors got at least some sort of feedback. A lot of patches were rejected because they only catered to a very small percentage of the userbase, or because they caused backwards-incompatibility, or because they were downright crap, but some were "approved" and I kept them around. Most of the approved ones made it into Nagios (sooner or later anyways), and I started gaining some developer cred in the community.

In 2008, Ethan disappeared completely from the project and no new code could go in at all. In 2009, a bunch of Germans got really tired of that, as they had good patches that they really needed in order to support their customers, so Netways forked Icinga. I didn't really approve of the way in which they did it back then (lots of blaring trumpets and things), but in retrospect I think it was a lot of political maneuvering from both Nagios Enterprises and

Netways back then, and the rest of us didn't see the whole picture.

Anyways, as a result of that Ethan realized he needed to do something to get the project going so Ton Voon, Thomas Guyot-Sionnest and my humble self were asked to become core developers and part of the Nagios Community Advisory Board. The core dev part was good, as it meant I could commit patches directly to Nagios instead of keeping them around in my own repository. The second part turned out to be just marsh gas, as we were never consulted on anything related to Nagios at all. The same year I met up with Ton and Ethan in Bolzano for a conference there, and I drew up my plans for improvements I wanted to implement in the core code. Noone argued against it, so I set out to get time from op5 in which to do the work.

In the meantime, I shared those plans with Jean Gabés, who was very put out by the fact that a patch he had submitted to Nagios that made it a bit more efficient hadn't been accepted. I expanded on the idea and in an email discussion between me and him I told him that it would be better to use worker processes to run checks rather than the way Nagios 3 used to do it. Jean built a proof-of-concept of that idea, and it turned out to be so good and so much fun that he made a real project out of it and called it Shinken.

In 2011 I started working on the changes that would make Nagios 3 become the awesomely performant Nagios 4, and in 2012 they were completed. During this period, I was becoming known more and more as "The Nagios Developer", since Ethan was still nowhere to be seen. Looking at the commit history from around 2010, I can't really argue against that.

Why did you decide to leave the project ?

I didn't decide to leave. I got kicked out. After the Nagios World Conference North America this year, Ethan approached me as I was sitting in the bar waiting for my flight home and accused me of knowingly and willingly sabotaging the Nagios 4 release. The reason for that was a patch I had submitted that broke the CGI builds. The fix took all of 30 seconds to apply, but he claimed his devs had spent a whole day working on it, and he was seriously upset. He was also very upset that the op5 marketing department had used the fact that I wrote the vast majority of the Nagios 4 code (94.5%, to be precise) in their marketing strategies, and I believe Nagios Enterprises actually lost a few customers because of that.

My attempts to persuade him that the two reasons were mutually exclusive, since I wouldn't want to take credit for something that didn't work properly, fell on deaf ears. He just wouldn't listen and he told me that from now on I'd have to submit my patches to the Nagios developers just like everyone else.

As a side-note here, I might add that the mailing lists closed a few days before the conference and everything was moved to a moderated forum instead. That's where Ethan wanted me to submit my patches. A few op5 developers actually have done just that, and it resulted in them being applied with Eric Stanley as the author of the patch. I personally don't believe Eric did that on purpose (and if he did, I'm reasonably sure he didn't think that up himself), but it does mean that Nagios Enterprises are now effectively locking out other people from gaining anything what so ever for working on the Nagios code. Ethan wants other people to do all the work while he reaps all the benefits.

Needless to say; That's not how an opensource project should be run, and it's not a project I wish to contribute too. I'm not sure what would've happened if he hadn't kicked me out, but the direction that Nagios is going is not one I agree with, and I might have started my own project at some point anyway.

Why a new fork, why not joining a forked team (Icinga, Shinken…) ?

Good question. There are many reasons.

I would't want to join Shinken because I'm a firm believer that heavy-duty system applications should be written in the most efficient language possible, and then kept so simple that they just can't break while interfacing nicely with other systems that can be written in any language it pleases. I'm also not as comfortable with Python as I am with C, and performance tests show that Naemon beats Shinken in terms of checks executed.

I wouldn't want to join Icinga, mainly because they build a *lot* of things on IDOUtils. I personally think IDOUtils is a bad technology with a very clumsy database schema, and I would want to scrap it as fast as possible. That would be a problem for Netways, who run the Icinga project, so we'd most likely run into problems right from the start.

Those are the only two projects I would even consider though, but the real reason is more likely that I don't want to follow someone elses project anymore, but run my own and see how it goes. I'm very tired of the political bullshit that happened during my recent release from the Nagios project, and I don't want to go through that again. The only way I can make sure that it doesn't is by running a new project and getting people I trust to value technical excellence higher than marketing value to join me. So far I believe I've managed to do just that, as Sven Nierlein (author of mod_gearman and Thruk and exceptionally laid-back guy) has agreed to join the project, as has Robin Sonerfors, which is a close friend from where I work. An american named Dan Wittenberg has also offered to help out, and since he runs networks far too large for anything *but* technical excellence, I've got a good feeling we're on the right track here.

What are or will be the major differences between Naemon and other competitors like Nagios, Shinken, Icinga to name a few ?

Naemon will, at least initially, not focus on bells and whistles. We will build a core product that is so stable, performant and easy to integrate other things with that it can withstand a proverbial nuclear attack (don't try that at home though; It's not good for you).

I have no interest in building user interfaces, or cool ways to display graphs. I *do* have interest in making it easy for other people to do that though, and then let those other people duke it out with their projects so the best project with the most people liking, downloading and developing it wins out. That's how it should be. It should also make it possible for companies to build extensions on top of Naemon that they don't necessarily have to share with the entire world, which creates a business opportunity that relies only on competition and best utilizing the resources available.

What are your main focus for Naemon ?

As with any program I write: Stability, performance, extendability. I want to keep it small-ish and build interfaces into it that makes it possible for others to build more things around it. I can't, don't want to and won't build everything myself. That would be stupid (and selfish) of me. I'm very good at building scalable core systems and programming interfaces, and I'm very bad at designing user interfaces.

What is your businnes model, if any, for Naemon ?

Subscriptions, enterprise version, paid support… ?

There is no business model. Naemon is free (as in beer and speech). I may end up adding links to developer's Amazon wishlists or something similar, or put up a paypal "donate if you want" button, but there is no commercial interest in Naemon by itself. I imagine companies will want to exploit it to sell to their customers though, and I welcome them to do so. I hope they will share some of their projects with the rest of the community, or invite me to a conference and get me drunk every now and then, but there's no need to. Noone *has* to pay to use it.

What place would you like to give in the project to the Nameon community ?

I'm not sure what you mean by that question, but I would like for Naemon to be a superior alternative as a core engine to all other network monitoring solutions, and I would like developers and users to think up things to do with it that I've never even thought of. That's usually a really good marker of an excellent design, by the way; When people take your program and do extraordinary things that you never even considered, you know you've done something really, really right.

Are you open to external contributions, encourage them ?

Definitely. I've already taken patches from some guys in Iceland, and I hope they keep sending their stuff. I'll also take patches that keep in line with the vision we have for Naemon, so long as they don't add horribly messy code or break backwards compatibility too bad.

Can you give us a roadmap for what is coming in next releases, even a global one ?

Yes and no. I can give you a vision, and a few steps that I believe will start the journey towards that vision.

Sometime in the future, I want Naemon to be mostly event-driven. Hosts and services should be added or deleted on the fly, and rulesets should be applied to the results they send in based on historical efforts.

In order to do that, we'll need addons that can run when the scheduler kicks in and starts doing things, so scheduler-controlled helper daemons will be the first feature to be added. To facilitate that, we'll have to build in drop-dir support so that configuration files that load modules and configure helpers can just be dropped in a directory, for easy management by things like RPM packages and Puppet.

Then we'll expand on the NERD radio and the query handler interface to make it easier for external applications to pull information out of Naemon and use it as it sees fit. Building in livestatus is a natural part of that process as well, and it's quite trivial since we already have support for "in-core" modules, much the same way that apache modules can be either built as separate modules or compiled into the core directly.

Once we have that in place we'll most likely take a break for christmas and new years, and then we'll look at modifying existing projects to make use of the new features. PNP's performance data processing could be made a lot more efficient, for example, and it's not gonna be hard to make it perform a lot better once we have the expanded NERD channels and the scheduler controlled helper demons.

Assuming we did things right when designing the core interfaces, we can then move on to experimenting with self-learning monitoring. Take a look at BisCheck if you want to see the kinds of things I mean. When that's done, it's time to add support for runtime addition and deletion of objects. It's likely we'll need object extensions as well at this point, but that won't be much of an issue.

Smaller features that I'll build "because it's better that way" are things like runtime modification of the main configuration parameters (although not all of them; One still has to be able to find the query handler socket, for example, and certain variables only make sense at startup anyway).

I also still nourish the dream of having service sets in Naemon, where one can configure a whole slew of checks in one go, and vendors such as Amazon, Google and various web hotels and "as a service" providers can create, maintain and ship their own kits, complete with plugins and suggested theshold values, so people can set up their monitoring systems in a much smoother way and the ones who actually write the APIs for getting data out of the system also write the checks that make sure they work.

If that last feature becomes really good and really useful, it's one of those things that can make Naemon really, really huge.