INTRODUCTION

This is a book about untrustworthy machines; machines, in fact, that are every bit as untrustworthy as they are critical to our well-being. But I don't need to bore you with a laundry list of how prevalent computer systems have become or with horror stories about what can happen when they fail. If you picked up this book, then I'm sure you're aware of the problems: layer upon layer of interdependent libraries hiding bugs in their abstraction, script kiddies, viruses, DDOS attacks, hardware failures, end-user errors, back-hoes, hurricanes, and on and on. It doesn't matter whether the root cause is malicious or accidental, your systems will fail. When they do fail, only two things will save you from the downtime: redundancy and monitoring systems.

Do It Right the First Time

In concept, monitoring systems are simple: an extra system or collection of systems whose job is to watch the other systems for problems. For example, the monitoring system can periodically connect to a Web server to make sure it responds and, if not, send notifications to the administrators. Although it sounds straightforward, monitoring systems have grown into expensive, complex pieces of software. Many now have agents larger than 500 MB, include proprietary scripting languages, and sport price tags above $60,000.

When implemented correctly, a monitoring system can be your best friend. It can notify administrators of glitches before they become crises, help architects tease out patterns corresponding to chronic interoperability issues, and give engineers detailed capacity planning information. A good monitoring system can help the security guys correlate interesting events, show the network operations center personnel where the bandwidth bottlenecks are, and provide management with much needed high-level visibility into the critical systems that they bet their business on. A good monitoring system can help you uphold your service level agreement (SLA) and even take steps to solve problems without waking anyone up. Good monitoring systems save money, bring stability to complex environments, and make everyone happy.

When done poorly, however, the same system can wreak havoc. Bad monitoring systems cry wolf at all hours of the night so often that nobody pays attention anymore; they install backdoors into your otherwise secure infrastructure, leech time and resources away from

other projects, and congest network links with megabyte upon megabyte of health checks. Bad monitoring systems can really suck.

Unfortunately, getting it right the first time isn't as easy as you might think, and in my experience, a bad monitoring system doesn't usually survive long enough to be fixed. Bad monitoring systems are too much of a burden on everyone involved, including the systems being monitored. In this context, it's easy to see why large corporations and governments employ full-time monitoring specialists and purchase software with six-figure price tags. They know how important it is to get it right the first time.

Small- to medium-sized businesses and universities can have environments as complex or even more complex then large companies, but they obviously don't have the luxury of high-priced tools and specialized expertise. Getting a well-built monitoring infrastructure in these environments, with their geographically dispersed campuses and satellite offices, can be a challenge. But having spent the better part of the last 7 years building and maintaining monitoring systems, I'm here to tell you that not only is it possible to get it done right the first time, but you can also do it for free, with a bit of elbow grease, some open source tools, and a pinch of imagination.

Why Nagios?

Nagios is, in my opinion, the best system and network-monitoring tool available, open source or otherwise. Its modularity and straightforward approach to monitoring make it easy to work with and highly scalable. Further, Nagios's open source license makes it freely available and easy to extend to meet your specific needs. Instead of trying to do everything for you, Nagios excels at interoperability with other open source tools, which makes it flexible. If you're looking for a monolithic piece of software with check boxes that solve all your problems, this probably isn't the book for you. But before you stop reading, give me another paragraph or two to convince you that the check boxes aren't really what you're looking for.

The commercial offerings get it wrong because their approach to the problem assumes that everyone wants the same solution. To a certain extent, this is true. Everyone has a large glob of computers and network equipment and wants to be notified if some subset of it fails. So, if you want to sell monitoring software, the obvious way to go about it is to create a piece of software that knows how to monitor every conceivable piece of computer software and networking gear in existence. The more gadgets your system can monitor, the more people you can sell it to. To someone who wants to sell monitoring software, it's easy to believe that monitoring systems are turnkey solutions and whoever's software can monitor the largest number of gadgets wins.

The commercial packages I've worked with all seem to follow this logic. Not unlike the Borg, they are methodically locating new computer gizmos and adding the requisite monitoring code to their solution, or worse: acquiring other companies that already know how to monitor lots of computer gadgetry and bolting those companies' codes onto their own.

They quickly become obsessed with features, creating enormous spreadsheets of supported gizmos. Their software engineers exist so that the presales engineers can come to your office and say to your managers, through seemingly layers of white gleaming teeth, "Yes, our software can monitor that."

The problem is that monitoring systems are not turnkey solutions. They require a large amount of customization before they start solving problems, and herein lies the difference between people selling monitoring software and those designing and implementing monitoring systems. When you're trying to build a monitoring system, a piece of software that can monitor every gadget in the world by clicking a check box is not as useful to you as the one that makes it easy to monitor what you need, in exactly the manner that you want. By focusing on *what* to monitor, the proprietary solutions neglect the *how,* which limits the context in which they may be used.

Take ping, for example. Every monitoring system I've ever dealt with uses ICMP echo requests, also known as pings, to check host availability in one way or another. But if you want to control *how* a proprietary monitoring system uses ping, architectural limitations become quickly apparent. Let's say I want to specify the number of ICMP packets to send or want to send notifications based on the round-trip time of the packet in microseconds instead of simple pass/fail. More complex environments may necessitate that I use IPv6 pings, or that I portknock before I ping. The problem with the monolithic, feature-full approach is that these changes represent changes to the core application logic and are, therefore, nontrivial to implement.

In the commercial-monitoring applications I've worked with, if these ping examples can be performed at all, they require re-implementing the ping logic in the monitoring system's proprietary scripting language. In other words, you would have to toss out the built-in ping functionality altogether. Perhaps controlling the specifics of ping checks is of questionable value to you, but if you don't actually have any control over something as basic as ping, what are the odds that you'll have finite enough control over the most important checks in your environment? They've made the assumption that they know *how* you want to ping things, and from then on it was game over; they never thought about it again. And why would they? The ping feature is already in the spreadsheet, after all.

When it comes to gizmos, Nagios's focus is on modularity. Single-purpose monitoring applets called plugins provide support for specific devices and services. Rather than participating in the feature arms race, hardware support is community-driven. As community members have a need to monitor new devices or services, new plugins are written and usually more quickly than the commercial applications can add the same support. In practice, Nagios always supports everything you need it to and without ever needing to upgrade Nagios itself. Nagios also provides the best of both worlds when it comes to support, with several commercial options, as well as a thriving and helpful community that provides free support through various forums and mailing lists.

Choosing Nagios as your monitoring platform means that your monitoring effort will be limited only by your own imagination, technical prowess, and political savvy. Nagios can go anywhere you want it to, and the trip there is usually simple. Although Nagios can do every-

thing the commercial applications can and more, without the bulky insecure agent install, it usually doesn't compare favorably to commercial-monitoring systems because when spreadsheets are parsed, Nagios doesn't have as many checks. If they're counting correctly, Nagios has no checks at all, because technically it doesn't know *how* to monitor anything; it prefers that you tell it how. How, in fact, is exactly the variable that the aforementioned check box cannot encompass. Check boxes cannot ask how; therefore, you don't want them.

What's in This Book?

Although Nagios is the biggest piece of the puzzle, it's only one of the myriad of tools that make up a world-class open source monitoring system. With several books, superb online documentation, and lively and informative mailing lists, it's also the best documented piece of the puzzle. So my intention in writing this book is to pick up where the documentation leaves off. This is not a book about Nagios as much as it is a book about the construction of monitoring systems using Nagios, and there is much more to building monitoring systems than configuring a monitoring tool.

I cover the usual configuration boilerplate, but configuring and installing Nagios is not my primary focus. Instead, to help you build great monitoring systems, I need to introduce you to the protocols and tools that enhance Nagios's functionality and simplify its configuration. I need to give you an in-depth understanding of the inner workings of Nagios itself, so you can extend it to do whatever you might need. I need to spend some time in this book exploring possibilities because Nagios is limited only by what you feel it can do. Finally, I need to write about things only loosely related to Nagios, such as best practices, SNMP, visualizing time-series data, and various Microsoft scripting technologies, such as WMI and WSH.

Most importantly, I need to document Nagios itself in a different way than normal. By introducing it in terms of a task-efficient scheduling and notification engine, I can keep things simple while talking about the internals upfront. Rather than relegating important information to the seldom-read advanced section, I empower you early on by covering topics such as plugin customization and scheduling as core concepts.

Although the chapters stand on their own and I've tried to make the book as reference-friendly as possible, I think it reads better as a progression from start to end. I encourage you to read from cover to cover, skipping over anything you are already familiar with. The text is not large, but I think you'll find it dense with information and even the most-seasoned monitoring veterans should find more than a few nuggets of wisdom.

The chapters tend to build on each other and casually introduce Nagios-specific details in the context of more general monitoring concepts. Because there are many important decisions that need to be made before any software is installed, I begin with "Best Practices" in

Chapter 1. This should get you thinking in terms of what needs to take place for your monitoring initiative to be successful, such as how to go about implementing, who to involve, and what pitfalls to avoid.

Chapter 2, "Theory of Operations," builds on Chapter 1's general design guidance by providing a theoretical overview of Nagios from the ground up. Rather than inundating you with configuration minutiae, Chapter 2 gives you a detailed understanding of how Nagios works without being overly specific about configuration directives. This knowledge will go a long way toward making configuration more transparent later.

Before we can configure Nagios to monitor our environment, we need to install it. Chapter 3, "Installing Nagios," should help you install Nagios, either from source or via a package manager.

Chapter 4, "Configuring Nagios," is the dreaded configuration chapter. Configuring Nagios for the first time is not something most people consider to be fun, but I hope I've kept it as painless as possible by taking a bottom-up approach, only documenting the most-used and required directives, providing up-front examples, and specifying exactly what objects refer to what other objects and how.

Most people who try Nagios become attached to it and are loathe to use anything else. But if there is a universal complaint, it is certainly configuration. Chapter 5, "Bootstrapping the Configs," takes a bit of a digression to document some of the tools available to make configuration easier to stomach. These include automated discovery tools, as well as graphical user interfaces.

In Chapter 6, "Watching," you are finally ready to get into the nitty-gritty of watching systems, which includes specific examples of Nagios plugin configuration syntax and how to solve real-world problems. I begin with a section on watching Microsoft Windows boxes, followed by a section on UNIX, and finally the "Other Stuff" section, which encompasses networking gear and environmental sensors.

Chapter 7, "Visualization," covers one of my favorite topics: data visualization. Good data visualization solves problems that cannot be solved otherwise, and I'm excited about the options that exist now, as well as what's on the horizon. With fantastic visualization tools such as RRDTool and no fewer than 12 different glue layers to choose from, graphing time series data from Nagios is getting easier every day, but this chapter doesn't stop at mere line graphs.

And finally, now that you know the rules, it's time to teach you how to break them. At the time of writing Chapter 8, "The Nagios Event Broker Interface," it was the only documentation I'm aware of to cover the new Nagios Event Broker interface. The Event Broker is the most powerful Nagios interface available. Mastering it rewards you with nothing less than the ability to rewrite Chapter 2 for yourself by fundamentally changing any aspect of how Nagios operates or extending it to meet any need you might have. I describe how the Event Broker works and walk you through building an NEB module.

Who Should Read This Book?

If you are a systems administrator with a closet full of UNIX systems, Windows systems, and assorted network gadgetry, and you need a world-class monitoring system on the cheap, this book is for you. Contrary to what you might expect, building monitoring systems is not a trivial undertaking. Constructing the system that potentially interacts with every TCP-based device in your environment requires a bit of knowledge on your part. But don't let that give you pause; systems monitoring has taught me more than anything else I've done in my career and, in my experience, no matter what your level of knowledge, working with monitoring systems has a tendency to constantly challenge your assumptions, deepen your understanding, and keep you right on the edge of what you know.

To get the most out of this book, you should have a good handle on the text-based Internet protocols that you use regularly, such as SMTP and HTTP. Although it interacts with Windows servers very well, Nagios is meant to run on Linux, which makes the text Linux-heavy, so a passing familiarity with Linux or UNIX-like systems is helpful. Although not strictly required, you should also have some programming skills. The book has a fair number of code listings, but I've tried to keep them as straightforward and easy-to-follow as possible. With the exception of Chapter 8, which is exclusively C, the code listings are written in either UNIX shell or Perl.

Perhaps the only strict requirement is that you approach the subject matter with a healthy dose of open curiosity. If something seems unclear, don't be discouraged; check out the online documentation, ask on the lists, or even shoot me an email; I'd be glad to help if I can.

For more information, as well as full-color diagrams and code listings, visit http://www.skeptech.org/nagiosbook.