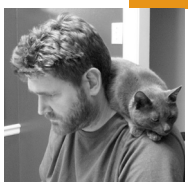
DAVE JOSEPHSEN

iVoyeur: message in a bottle



REPLACING EMAIL WARNINGS WITH SMS

David Josephsen is the author of *Building a Monitoring Infrastructure with Nagios* (Prentice Hall PTR, 2007) and Senior Systems Engineer at DBG, Inc., where he maintains a gaggle of geographically dispersed server farms. He won LISA '04's Best Paper Award for his co-authored work on spam mitigation, and he donates his spare time to the SourceMage GNU Linux Project.

dave-usenix@skeptech.org

WE FINALLY DID IT. DEGREE BY TINY degree, feature by "unsafe" feature, we killed email. This perhaps comes as no surprise to you, but I arrived at the realization only recently. Somewhere between SPF, DKIM, and the four-millionth RBL a critical mass was reached. It might be that one too many of the engineers who believed in the robustness principle [1] is now retired or dead, or perhaps it's the result of one too many overzealous vendors readjusting the corporate perception of "normal"; it might just be a question of scale. Whatever the reason, there's no question that today the onus is on you to convince your recipients' hostile, unwilling MX that your message is worth delivering, and attempting to do so makes you a spammer. These days any means justifies the claim of a little less spam.

Sending too many mails or not enough; having SPF or not; using domain keys, DKIM, both, or none; having reverse lookups; using text or HTML: all of these are indicators of spam and all of them aren't; there is nothing you can do in 2008 to appear to be a legitimate sender to every recipient. That after decades of flawless operation email has finally become an untrustworthy means of communication is as shocking as the fact that the overtly lazy, idiotic, wholesale destruction of SMTP was carried out in the name of "security"—by us, no less—in a vain attempt to outsmart Viagra peddlers one lazy quick fix at a time. Today the only way you can reliably get your messages delivered is to personally know the sysadmin at your recipients' organizations and get on their white-lists. There are secret invitation-only mailing lists for this purpose. You know who you are.

So what do you do when the quick fix of the week decides your monitoring server is questionable and you can no longer deliver email alerts to your pager or that of your peers? Move to a new messaging protocol? Maybe one such as SMS, that is younger and not yet broken? It'll work for a while, and hopefully something will be there to replace it when we break it too. You could move backward to an older, more trustworthy system such as the PSTN, a retro yet dependable option. A third option might be to use both, a gsm/gprs modem for SMS with the PSTN used as a backup, for example.

In this article we'll explore the third option, using a Nokia cell phone tethered to our Nagios server with a serial cable to send SMS, and alternately calling people on the phone with Asterisk if our Nokia is not up to the task.

I chose the Nokia 6170 for my own SMS implementation because it is readily available for about $100, is compatible with several U.S. carriers (we use T-Mobile), and has excellent compatibility with gnokii [2], the software we'll be using to interact with the phone. You'll also need a Nokia USB DKU-2 cable to connect the phone to the Nagios server. Beware of after-market cables, for they sometimes don't work as expected, and spend the extra $5 for the Nokia cable. I got mine on ebay for $20.

gnokii is a user-space driver and tool suite for communicating with mobile phones. It supports the usual free UNIX OSes and is scriptable via the command line and via a C library called libgnokii, for which the usual scripting language wrappers exist. Originally the intent of the gnokii authors was to operate with Nokia phones, but the tool suite can be made to work with any AT-compatible phone using serial, USB, IRDA, or Bluetooth. I can't speak to how hard this is, since I took the easy road and just got a Nokia, but the Wiki [3] lists a few non-Nokia phones that folks have gotten to work.

Gnokii supports a litany of features including getting/putting calendar events, editing phone-book entries, and dialing voice calls (useful for pranks and, I imagine, for war-dialing). This article will only use the `--send-sms` feature, although gnokii is capable of receiving SMS as well. Gnokii installs with the typical automake commands (`configure/make/make install`) and requires libusb for USB support and bluez for bluetooth. When gnokii starts up it checks the current user's home directory for .gnokiirc and then /etc/gnokiirc. You can specify a custom config file with the `--config` option.

My configuration file for the 6170 looks like this:

```
model = 6170
port = /dev/ttyS0
connection = dku2libusb
```

Once gnokii is installed and the phone is connected to the USB port of the Nagios server, issuing a `gnokii --identify` command should return some information on the phone. If it doesn't, adding `debug = on` to the config file might print some helpful error messages.

Integration with Nagios, as you might guess, involves defining a notification command. Mine is this:

```
define command{
   command_name notify-by-sms
   command_line /usr/bin/printf "%b" "`echo $NOTIFICATIONTYPE$
| /usr/bin/cut -c '1-3'`: $HOSTNAME$/$SERVICEDESC$ $SERVICESTATE$"
| /usr/bin/gnokii --sendsms $CONTACTPAGER$ 2>&1 | /usr/bin/logger -t
Nagios -p local4.info
}
```

That command might be a tad difficult to parse because of the nested system call to echo. This is intended to abbreviate the value of Nagios's `$NOTIFICATIONTYPE$` macro (either `PROBLEM` or `RECOVERY`) to a three-letter word (PRO or REC). The message also doesn't contain many of the details you might expect in a Nagios notification, such as the date or plug-in output, because the notification is designed to always fit within the 140 bytes allowed in an SMS message. The logger command at the end is intended to catch any error output from gnokii and send it to syslog.

Like any well-behaved UNIX program, gnokii exits 0 if everything went OK. This means that instead of piping to logger after gnokii, we could use a logical or operator (||) to launch a different command if gnokii is unsuccessful. This is a good place to put our Asterisk script.

Asterisk, as you probably know, is an open source PBX system. It is packed to the brim with features and is the subject of at least one article in just about every issue of *;login:* in recent memory. Asterisk is so featureful, in fact, that it feels silly to be using it for something as small as Nagios notifications, but it works excellently in this regard and is especially worth thinking about if you already have an Asterisk implementation of some kind.

The general strategy here is to use the festival-lite text-to-speech engine to create an error report, and pass this to Asterisk, which will call people and recite it to them over the phone. To do this you need an existing Asterisk system, or a telephony card of some type in the Nagios box. We use the TDM410 from Digium [4]. Installing Asterisk is a snap; I recommend using the packages from your distro, as several drivers need to be built and there are kernel dependencies involved.

Asterisk is normally a beast to configure, but in this context there isn't much to do. Normally you'd spend a lot of time configuring dialplans in extensions.conf, but since nobody will be calling this Asterisk server, all there really is to do is set up your hardware. For the TDM410 this means editing zapata.conf. The relevant section of mine looks like this:

```
context=incoming
signalling=fxs_ks
include => [default]
channel => 2
```

The easiest way to make Asterisk call people from the shell is to use the call files interface. Simply create a text file with the relevant data and drop it in /var/spool/asterisk/outgoing: Asterisk will immediately make the phone call. Here's what a typical call file looks like:

```
Channel: Zap/g2/15558675309
WaitTime: 15
Application: Playback()
Data: /var/spool/nagios/alerts/tmp.ihbgAO3751.gsm
```

I tend to use shorter wait times than the default (45 seconds) so that voicemail doesn't have a chance to answer. Reliably leaving voicemail gets more complicated, so I prefer to just not deal with it. If I have a missed call from the Nagios box, I'll log in and see what's up. The data file is created by simply echoing the alert text to festival, like so:

```
echo "Nagios ${NOTIFICATIONTYPE}, Host ${HOSTNAME}, Service
${SERVICEDESC} is in state ${SERVICESTATE}" | flite -o somefile.wav
```

We can refer to these variables inside a shell script called by the notify-by-sms Nagios command because any script called by Nagios is given global variables that correspond to Nagios macros relating to the service outage. So many times I see people using macros as arguments to shell scripts called from within Nagios notification commands when it's always unnecessary; even many of the FAQ answers on nagios.org do this. Dear Internet: You don't have to mess with argument passing; your script already has all the macros as global vars. I digress. Asterisk can't read wav files, so we need to convert it to GSM, like so:

```
sox somefile.wav -r 8000 somefile.gsm resample -q1
```

When we create the call file, we need a way to map some Nagios macro to a phone number. Since we're using Asterisk to backup SMS in this example, the $CONTACTPAGER$ macro will work, but if you were backing up email you'd either need a lookup table of some type or a custom notification macro that specifies our contact's phone number. Nagios has for some time supported addressX macros that are perfect for this; just make sure address4, for example, always has a phone number, and you're all set.

That gives you pretty much all the pieces you need. A simple shell script can then be written that is called with a logical or in the notification command, like so:

```
command_line   /usr/bin/printf "%b" "`echo $NOTIFICATIONTYPE$ | /usr/bin/
cut -c '1-3'`: $HOSTNAME$/$SERVICEDESC$ $SERVICESTATE$" | /usr/bin/
gnokii --sendsms $CONTACTPAGER$ || /usr/local/nagios/bin/contact_by
_phone.sh
```

Now if the SMS message fails, Nagios will call the contact_by_phone shell script, which will use various Nagios macros to create a GSM audio message and an Asterisk call file and place the call file into /var/spool/asterisk/outgoing.

If you want to get fancy, you could specify a dialplan context instead of an application name in the call file, conceivably allowing the person being called to do things such as "press 1 to acknowledge this alert," "press 2 to run event-handler X," etc. Asterisk has some pretty cool remote management potential in this regard, which is perhaps fodder for a future article. If you're currently doing any Asterisk/Nagios integration stuff, I'd love to hear about it. Feel free to drop me an email (you know, if anybody still uses email; my MX promises to be nice) or to post a comment on my blog, www.skeptech.org.

Take it easy.

REFERENCES

[1] Robustness principle: The ancient fallacy that one should be liberal in what one accepts and conservative in what one sends.

[2] http://www.gnokii.org.

[3] http://wiki.gnokii.org/index.php/Config.

[4] http://www.digium.com/en/products/analog/tdm410.php.