;login:

THE MAGAZINE OF USENIX & SAGE
June 2002 volume 27 • number 3

inside:

SECURITY

PROTOWRAP
by Gunnar Wolf **y**

USENIX & SAGE

The Advanced Computing Systems Association &
The System Administrators Guild

protowrap

A Generic and Protocol-Specific Wrapper

by Gunnar Wolf

Gunnar Wolf is the systems administrator for a campus of UNAM, Mexico's largest University. He has been a strong promotor of both Computer Security awareness and the Free Software movement in his community, and likes doing small, useful tools in Perl.

gwolf@campus.iztacala.unam.mx

As system administrators, all of us should be very concerned, even paranoid, about security. There are simply too many threats out there waiting for us to become distracted in order to exploit a vulnerability in our systems. New vulnerabilities are found day to day, and exploits are very quickly crafted for each of them. We want to trust the programs we run at our server, but we know that a secure server is utopian. There are many ways to harden our servers against unknown attacks, but these often require heavy tinkering with the system. In this article, I will propose another alternative, much less invasive and more flexible than what I have found up to now: generic and protocol-specific wrappers, implemented through a couple of Perl modules called ProtoWrap.

A word of warning to the faint of heart: although ProtoWrap does work, it is a work-in-progress, and some important aspects (such as the interface to the operator) have been relegated to focus development on correct and full operation, not on ease of use. I am the first to recognize that I am by no means an excellent programmer, and I know that a better implementation can be made. If you think you can help make ProtoWrap work better, please do. In fact, if you want to develop a similar project sharing the ideas provided here, I would be delighted to hear about it.

General Philosophy

As you know by now (unless you like skipping introductions), ProtoWrap is a series of wrappers. Assuming we are in a hostile environment, we cannot trust a network client to be a good citizen. All incoming connections are, thus, potentially hostile and must pass through a defensive layer before reaching our real server.

I decided to focus my work on a very specific kind of protocol: line-oriented TCP protocols. By "line-oriented" I mean that all commands sent from the client to the server are one or more lines of text, delimited by a new-line character. Such protocols include HTTP, FTP, SMTP, IMAP, POP3, finger, ident, and many others. They do not, however, include Telnet (character-oriented; lines may be assembled by the server, but they simply are not relevant at the protocol level), DNS and NFS (UDP-based), SSH (traffic is encrypted, we can not guess where new lines are), and many others.

Wrapping a program should happen as transparently as possible. Many network services do not involve – on regular operation – a human at either end, and we should alter regular operation as little as possible in order not to trigger a denial of service.

ProtoWrap was designed to be useful even without knowing which protocol it would be wrapping (hence I describe it as a generic wrapper). We will see later how it can be invoked to protect almost every line-oriented protocol. It was very important for me, however, to make it easy to teach it how to intelligently wrap a specific protocol.

Another very important point is that ProtoWrap should be easy to deploy. It should not depend on a particular system configuration, and it should be able to scale. This will also be further explained later on.

Finally, ProtoWrap was designed to be able to protect heterogeneous networks. If ProtoWrap is unable to run on a particular system setup, it should be able to protect it from the outside, running on a different computer or even a different network.

In the Beginning

ProtoWrap began as my answer to stopping spam with different mail transport agents, on different architectures, with a minimum of work. I soon realized the solution I proposed could very easily be generalized, and become much more useful, to many different protocols.

I presented ProtoWrap as my final paper for graduation in computer science at Kennedy Western University. If you want to get full details on how and why ProtoWrap exists and works, I invite you to visit *http://www.gwolf.cx/seguridad/wrap/*.

I chose to develop ProtoWrap using Perl because of Perl's rich pattern-matching capabilities and because of the availability of just about any needed function in the CPAN (Comprehensive Perl Archive Network, *http://www.cpan.org*). Perl also provided me with a clean and easy-to-understand way of dealing with network sockets, an absolute requirement for the wrappers to exist.

A number of problems arose while developing ProtoWrap, as happens in any software project. Among the most challenging was the dual-input problem: data can come, at any moment, from either the client or the server. How can you make a Perl program listen to two different data sources at the same time, and react to the one that provides the whole line first? I owe the answer to Salvador Ortiz, with whom I spent several hours hacking and testing possible alternatives, until we decided to go to a lower level, using Perl's IO::Handle and IO::Select modules. Once again, for more details on the possibilities we studied and why this one was chosen, please visit *http://www.gwolf.cx/seguridad/wrap/node62.html*.

ProtoWrap is able to listen for clients and talk to its server in different ways. Some users might require that a daemon always be running, directly listening to its port, to save resources on multiple invocations and reduce startup time. Others will prefer running the wrapper from inetd, handing it the connection in the form of a STDIN/STDOUT file descriptor pair. As for the server, in some cases it will be started from ProtoWrap, also via a mechanism similar to inetd's, and in others it will always be running – maybe even on a different machine – and the connection will be done by TCP/IP sockets.

The Generic Wrapper Behavior

Using ProtoWrap in the most basic way is very straightforward. Of course, the protection it offers will not be as complete as if we were using a protocol-specific module. The protection that ProtoWrap will give to an unknown protocol is, nevertheless, very important: buffer overflows can be easily prevented altogether. Calling ProtoWrap as shown in Listing 1 results in having the IMAP server in the same system (as 127.0.0.1 is the localhost address) protected by ProtoWrap. The real server is running on port 10143, and should be protected by packet filtering or TCPWrapper rules so that it only accepts connections from the same machine. The result of this setup is shown in Figure 1.

```
#!/usr/bin/perl -w
use ProtoWrap;
use strict;

my $wrapper = ProtoWrap->new('standalone' => 1,
                'listenPort' => 143,
                'destType' => 'ip',
                'destAddr' => '127.0.0.1',
                'destPort' => 10143,
                'maxLineLength' => 25,
                'logLevel' => 0
                );
$wrapper->startServer() or die 'Can\'t start wrapper!';
sleep;
```

Listing 1

This wrapper will limit every line coming from the client to 25 characters, more than enough to send IMAP commands, and will effec-

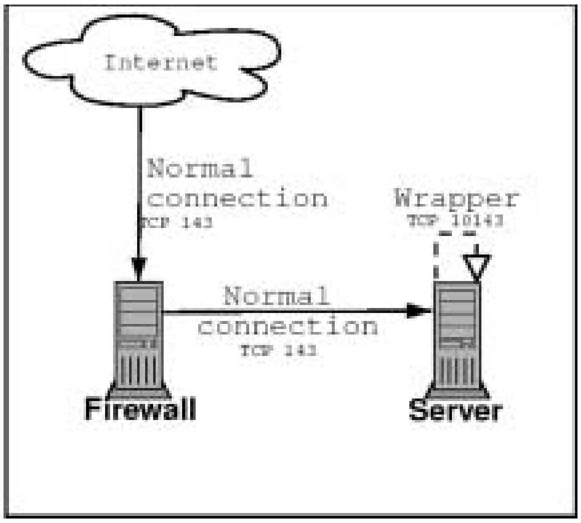

Figure 1

tively avoid any buffer overflow attack. Further, ProtoWrap will alert the system administrator via a syslog entry that an illegal line was sent to the server, reporting what the line's contents were.

Protocol-Specific Extensions

Of course, ProtoWrap can be easily extended well beyond this simple behavior. All protocols implement a specific set of instructions, and deviations from it can be easily marked as misuse and discarded with no remorse. In most protocols, it is also easy to define stages of operation – different subsets of commands will be valid or invalid at different times during the session. By making each line's validation by using the wrapper's testLine method, it is very easy to call a protocol-specific validation function. In order to demonstrate this, I wrote two protocol-specific wrappers: for POP3 and SMTP services.

```
#!/usr/bin/perl -w
use ProtoWrap::POP3;
use strict;

my $wrapper = ProtoWrap::POP3->new('standalone' => 0,
                  'destType' => 'ip',
                  'destAddr' => '127.0.0.1',
                  'destPort' => 10143,
                  'logLevel' => 0,
                  'maxLoginAttempts' => 3
                  );
$wrapper->startServer() or die 'Can\'t start wrapper!';
sleep;
```

Listing 2

A wrapper for POP3 can be called with Listing 2; maxLineLength was now omitted, as we will validate each line separately. Here, instead of running in stand-alone mode, the wrapper will now be invoked from inetd or a similar daemon, which will determine which port it will listen on. Conceptually, this setup still resembles Figure 1. We also have a new entry: maxLoginAttempts. If someone tries to log on with an incorrect password more than the specified number of attempts, the connection will be dropped.

The wrapper for SMTP is much more elaborate and able to do more. A typical startup configuration for SMTP can be seen in Listing 3. Here, instead of running SMTP at a different port and connecting to it via regular sockets, we do not run the SMTP server until a connection is received. The server will then be spawned, and when the connection is closed, only the wrapper will continue running (see Figure 2). Though it looks very similar to Figure 1, not having the server listening on a different port can make a huge difference, both from security and performance standpoints.

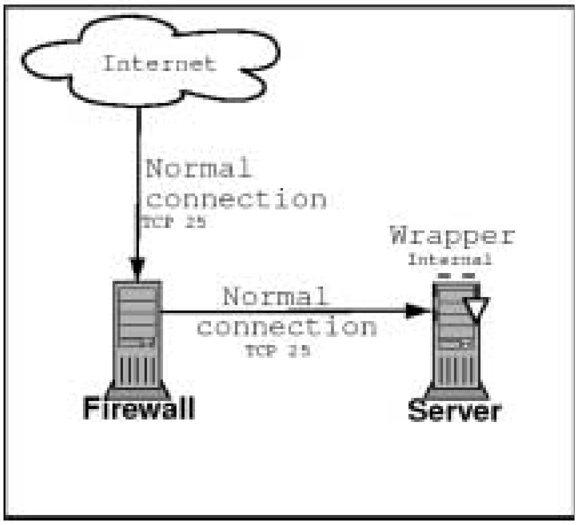

Figure 2

We see many new parameters here. Most of them were introduced to help stop spam. They are:

- blockAddrList – Addresses we do not wish to receive mail from (anchored to end of string). They can be specific mailboxes (as hahaha@sexyfun.net, a well know worm) or whole domains (everything coming from spammer.org).
- blockBodyList – Every line of the incoming message will be tested against the lines provided here, and if a line matches, the message will be discarded. In this example, most attachment viruses will be avoided, as the most common executable attachment types for Windows systems are disallowed.
- maxMsgSize – The maximum message size (in bytes). In the example, messages over a megabyte will not be allowed.
- maxRcpt – The maximum number of recipients for a message in a single SMTP session. Spammers usually send hundreds of messages at a time using open relays. If a spammer is able to use our machine as a relay, this will drastically cut its effectiveness as a spam relay. In the example, this number is set to zero, allowing for any number of recipients. This machine may be a mailing list server.

```
#!/usr/bin/perl -w
use ProtoWrap::SMTP;
use strict;
my $wrapper = ProtoWrap::SMTP->new('standalone' => 0,
                  'destType' => 'pipe',
                  'pipeCmd' => '/usr/sbin/sendmail -bs',
                  'logLevel' => 3,
                  'maxMsgSize' => 1048576,
                  'blockAddrList' => ['hahaha@sexyfun.net','@spammer.org'],
                  'blockBodyList' => ['^Content-Type: application.+\.(PIF|EXE|VBS|COM|BAT|LNK|SCR)\"'],
                  'relayIpList' => ['192.168.150.','192.168.160.'],
                  'relayDomainList' => ['mydomain.org','gwolf.cx'],
                  'maxRcpt' => 0
                  );

$wrapper->startServer() or die 'Can\'t start wrapper!';
sleep;
```

Listing 3

- relayDomainList – Domains for which we allow relay, when they appear either as senders or as recipients of a message (anchored to end of string).
- relayIpList – IP ranges or specific addresses for which we allow relay (anchored to the beginning of the string).

Some of these functions are already handled by most SMTP servers – why am I reimplementing them with ProtoWrap? First, most SMTP servers allow only for specific text matching. With ProtoWrap, we have access to the whole Perl regular expression engine, which gives us much more flexibility and ease of use. Second, if we have our wrappers at a central site such as a firewall, with a setup similar to Figure 3, configuration will be much easier to mantain than if we have them spread on each of our servers.

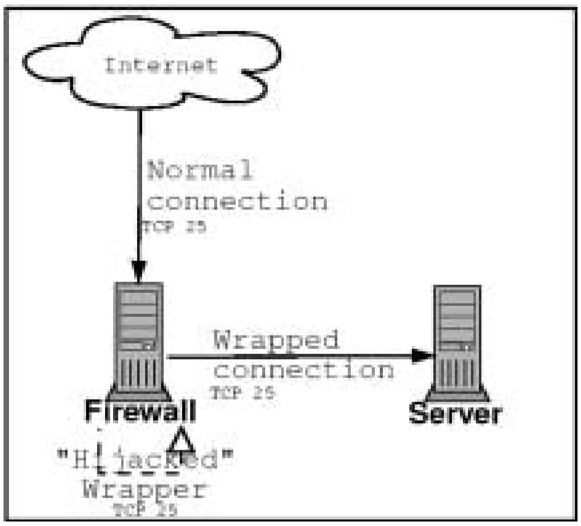

Figure 3

Wrapping Up

ProtoWrap has changed a lot as I have found and incorporated new ideas into it. I am sure it can be a very useful security tool to system administrators with almost every kind of setup. I am also sure that the ideas I have shown here are just the beginning of what can be achieved by such a wrapper.

I have been using ProtoWrap for almost a year on my production servers, since I first labeled it as usable. There are still many features pending, and I sincerely hope to have some of them done by the time this article reaches you. The actions I wish to take before labeling ProtoWrap as stable, and will be done before this goes to print, are:

- Correct Perl module packing – ProtoWrap should be installed as Perl modules. Right now, installation must be done by hand. Soon, I hope to have ProtoWrap ready to be set up as most Perl modules are.
- .rpm, .deb, .tgz packages – Most operating systems are provided with package management systems. Linux distributions handle usually either .rpm or .deb format packages; most other UNIX systems use the simpler .tgz format. These packages allow installation, deinstallation, version management and dependencies. ProtoWrap should then also be available in packaged format.

For more details on other various interesting points that ProtoWrap can be extended to cover, please visit *http://www.gwolf.cx/seguridad/wrap/node72.html*.

To sum up, ProtoWrap is just a proposal, a proof of concept, and I am more than sure it is not the ultimate security solution. It is, however, a valuable addition to most sites' overall security strategy.