

Simplified Single Packet Authorization

Richard Lundeen
lundrich@isu.edu
 University of Idaho
 921 South 8th Avenue
 Pocatello, ID, USA 83209

Charles Burns
burnchar@isu.edu
 Idaho State University
 921 South 8th Avenue
 Pocatello, ID, USA 83209

Abstract — Port Knocking and Single Packet Authorization (SPA) are relatively new (circa 2004 and later) techniques used to enable anonymous, temporary activation of remote network services that are otherwise blocked by means of a firewall. These techniques greatly enhance the so-called "zero-day" exploit resilience of systems which properly implement them, but they have weaknesses and more importantly share a weakness common to most common security augmentation system: human nature. This paper presents a framework for securely enabling remote services in a manner which focuses on the human factor, a concept often neglected in security research and the key reason that such systems rarely see widespread usage in the real-world. The primary focus is to make SPA easier for humans to interact with.

All code written for this project can be found at <http://webstersprodigy.net/pages/software.php> prior to conference proceedings.

Keywords — Firewall, Human Factor, Network Security, Single Packet Authorization, Zero Day Countermeasure

I. INTRODUCTION

THE Internet has been a core part of people's lives and the world economy for over a decade, but it is only as useful as the network services that run on top of it, and only as safe as the security of those services. Despite vast amounts of research into making these systems difficult to break, security concerns are more important now than they have ever been. Nearly all network applications in common use have had flaws which allowed unwanted access to important data.

Many security practices involve significant inconvenience to the people who use them, and many security applications have required substantial effort to fully implement properly. While many systems can provably enhance the safety of sensitive data, the weakest links in the security chain are usually the people using those systems and their tolerance to inconvenience.

Unfortunately, security research often neglects the human factor in favor of models with a more abstracted approach. Historically, the most successful security tools are those which are the easiest to use. Single Packet Authorization is a proven method of greatly increasing security. However, its adoption may be stifled due to a comparatively complex installation and usage process. In this paper we present a framework for simplifying interaction between humans and Single Packet Authorization (SPA).

II. BACKGROUND

A. Secure Shell

Some of the most successful instances of secure software are the ones that are both secure and easy to use. OpenSSH is a prime example of such a tool.

Two of the primary reasons SSH has been so successful are 1. security advantages over previous software solutions such as Telnet and 2. the ease of use, including ease of installation, operation, and configuration when compared to previous software. OpenSSH includes all the advantages and eases of use when compared to its predecessors, but has the obvious advantage of security.

OpenSSH is one of the flagships of internet security and of open source software. It is written and maintained by some of the most talented and security-conscious programmers in the world—the OpenBSD team.

However, for all their knowledge and attention to detail, OpenSSH had a remote root vulnerability, the most serious type of vulnerability, in 2002 [17] and another a year later [15]. Even as recently as 2008, the combination of a Debian flaw and OpenSSH resulted in one of the most notable security flaws of that year. [16].

The OpenSSH vulnerabilities and the vulnerabilities of many other important network services could have been mitigated with some variation of port knocking or SPA.

B. Port Knocking and SPA

Port Knocking implementations were first released in 2004, allowing a service to be rendered invisible to all outside systems by means of a firewall, yet still be selectively accessible to outside users by allowing them to open the appropriate firewall ports. This is done by running log monitoring software on the remote server which tracks attempted connections and instructs the system firewall to briefly open services when certain conditions are met.

As described in [12], there are several severe limitations to port knocking. Though it provides many security benefits, replay attacks and various means of denial of service are unavoidable.

SPA retains all the benefits of port knocking, but fixes many of the limitations.

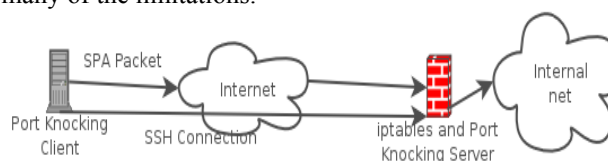


Fig. 1. The basic network architecture assumed for an SPA configuration.

SPA moves the authorization information away from Level 3 and into the more suitable application layer of the network stack. With TCP/IP-based port knocking, only 2 bytes of data per packet may be sent due to limitations of reasonably configured logging software on the server side. However, with SPA one minimum transmission unit (MTU) worth of data can be sent per packet (1,500 bytes on an Ethernet network).

Here it is helpful to examine specific examples of software. One of the most popular of the available SPA software suites is *fwknop*. At the application layer, *fwknop* defines: 16 bytes of random data (preventing replay attacks), the client username (useful for allowing different users different levels of access), a client timestamp, the SPA version number, the mode (which tells the *fwknop* server whether to execute a specific command or give access to a service), requested access (the service to be accessed or the command), and an MD5 sum of the packet.

Many of these fields allow a variable length, each base64-encoded and separated with a colon (':') character. This packet is encrypted using either the symmetric Rijndael cipher with a 128-bit shared key or using the asymmetric ElGamal algorithm with up to a 2,048-bit public/private key pair. The specific algorithm used is set in a configuration file or on the invocation of the program. The packet is sent to a configurable UDP port on the server. This process is described in greater detail in [12].

As a concrete example outlining the benefits of SPA, the Debian OpenSSH vulnerability mentioned earlier would not have been a significant threat even on affected systems were they using SPA, because additional layers of security would have prevented the vulnerable service from being exposed to any unauthorized activity regardless of any flaws that might exist in the service. This scenario is outlined in detail in [11].

For all the benefits, however, most implementations of SPA can still be difficult to implement. *Fwknop*, mentioned earlier, is one of the most easily deployable implementations, but even it requires source compilation, key generation, and editing various text configuration files, not to mention the associated research, before the service is functional. This is beyond the ability level of many computer users and requires a greater time investment than many systems administrators have available.

III. SIMPLIFIED SINGLE PACKET AUTHORIZATION

The primary goal of this paper is to make SPA extremely easy to use. Through common sense and observation of projects such as OpenSSH, it is intuitive that if a security service is simple to use and has a positive impact on security, it is much more likely to be adopted than a project that is difficult to use regardless of benefits. The concepts employed in this work consist of the following steps:

A. Binary Package

An *fwknop* binary package was built for Debian-based distributions including Ubuntu, one of the most popular Linux Distributions today.

B. Repositories

The binary package created in section A will be ported to the Ubuntu package repositories for easy installation.

C. Simple Configuration

Configuration files with reasonable defaults are included in the binary package. Additionally, *debconf* configuration is provided to setup custom *fwknop* installations within a simple interface.

D. Documentation

Clear and complete documentation is included in the package and made available online.

E. Integration with SSH

One of the most popular uses of *fwknop* is in securing SSH. A "wrapper package" is included to simplify this integration to end users.

Each of these steps will be described in detail in the following sections.

IV. BINARY PACKAGE

While creating a complete Debian binary package can be a time-consuming and technical process, it is well-documented and procedural [14]. With *fwknop* and Debian, this process proved to be relatively straightforward.

First, we created *data.tar.gz* including all the files with their destination paths. Some redundancy exists in the *fwknop* source code, which includes the destination paths in the *install.pl* script. However, to function properly on a Debian package management system, *data.tar.gz* is necessary, and the existence of the *install* script made its creation straightforward.

According to the Debian Package Maintainer's Guide, the files listed in the *data* file are identical to the upstream source. However, there are variations in the *preinst* file. *Fwknop* is similar to OpenSSH in that if the user selects to authenticate using *gpg* (see the configuration section below) the package includes functionality (using *snakeoil*) to generate the key. Unlike a default OpenSSH *install*, this key must still be manually copied over to every client wanting to use the system. This is similar encryption to authenticating with a public key in OpenSSH, which also requires manually copying keys.

Although possibly more secure, *gpg* can be difficult for the lay user. Symmetric authorization is defined as the default because of its ease of use. Again, this is parallel to how OpenSSH chooses its defaults. In OpenSSH, key authentication is superior to password authentication, but key authentication involves copying keys so it is almost never the default setting. Most users are unfamiliar with public key authentication, but are accustomed to typing a password.

The Debian control files are also fairly straightforward. For control, all the dependencies are available in the repositories, and are described in the official *Fwknop* documentation[13].

One of the most interesting files is the *postinst* file, which interacts with *debconf*. It first sources the *debconf* library, and then uses the *get* command to communicate

with the debconf frontend (*get* retrieves both the user input and defaults out of the debconf database). This separation is very convenient, since it allows the configuration to be performed on installation, but also can be called on an explicit `dpkg-reconfigure` call. Some user interaction is probably necessary - if it uses symmetric authorization a password or key must be entered. However, most users understand the concept of a password so this should not detract much from usability.

V. REPOSITORIES

It can be a difficult and sometimes tedious process to port a package to an official repository. In the case of Ubuntu, the package must be prepared in an extremely specific way, as described by [18]. The sheer amount of documentation to read through and follow can be daunting. There are many small but important steps to follow which we will not document here. Despite the difficulty, getting a package into the official repositories is extremely important to make installation as easy as possible and to encourage widespread use.

Why did we choose the Ubuntu distribution? Ubuntu is one of the most popular Linux distributions today, and it is widely used in the Idaho State University (ISU) Engineering Department (where both of the authors are employed). Although there are many rules in order for a package to be accepted into the official tree, the process is similar to any other distribution. Distributions need these rules to maintain consistency, and distribution maintainers need them to maintain simplify ongoing maintenance. The choice of distribution is less of an issue because similar steps can be repeated at a later time to encourage official support in Redhat, Solaris, BSD, and other Unix distributions.

At the time of this writing, the `fwknop` package is finished. Unfortunately, the Ubuntu maintainers are currently extremely busy with the upcoming release of Ubuntu 9.04 and are close to a so-called package freeze. When the package freeze ends, we will begin putting our package through the submission process.

One of the major goals of this project is to include an `fwknop` package in the next official release of Ubuntu after 9.04, which should be available in October, 2009. The Ubuntu Packaging Guide was followed throughout the process of creating the binary. If all goes well, it should be accepted without too much trouble.

VI. SIMPLE CONFIGURATION

A. Default Configuration

One of the reasons for the success of Ubuntu and Debian has been the *apt* package management system. Package management can make complicated software with complex and intricate configuration details much easier to install and setup.

By default, it is the job of a binary package to place these files in appropriate locations. We included well-commented configuration files with sane defaults in our binary package, so at a fundamental level, `fwknop` can work immediately using symmetric encryption after the installation with no user-interaction other than the user entering a password.

B. Debconf

As mentioned earlier, we wrote much of the configuration for the binary package using debconf. Debconf has the advantage of being the defacto standard for automatic, non-trivial configurations on Debian-like systems. There are many existing front-ends to debconf, including a curses-based option (Fig. 2) and a graphical tool named Synaptic (Fig. 3), both of which included with Ubuntu by default.

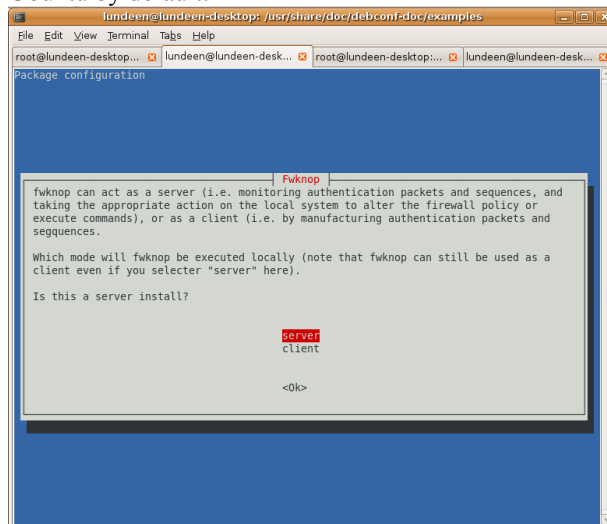


Fig. 2. The debconf included with the `fwknop` package we wrote, using the ncurses frontend invoked by `dpkg-reconfigure`.



Fig 3. An example default application (`x11-common`) making use of debconf invoked through Synaptic Package Manager.

Debconf makes use of two important files. The first is a templates file, and the other is a configuration file.

The templates file contains all the questions to ask the user, default answers to the questions, and the way the user can answer the questions. Determining applicable questions and asking them in an intuitive way is critical for end-user ease of use.

Here is an excerpt from `fwknop.templates`, which contains two questions it could ask the user:

```
Template: fwknop/operation
Type: select
```

Choices: \${operation}, server, client

Default: server

Description: Is this a server install?

Fwknop can act as a server (i.e. Monitoring authentication packets and sequences, and taking the appropriate action on the local system to alter the firewall policy or execute commands), or as a client (i.e. By manufacturing authentication packets and sequences.

Which mode will fwknop be executed locally (note that fwknop can still be used as a client even if you select "Server" here.

Template: fwknop/netdev

Type: string

Default: eth0

Description: What network Device will fwknop be running on?

Type the network device fwknopd will be listening on here (eth0, lo, etc):

Fwknop.templates contains all the questions which may need to be asked to the user, but does not determine when and where these questions are asked. The file, *fwknop.config*, is responsible for determining which questions to ask, and when to ask them. The code is a bash script that sources debconf to save the user's answers. Questions are given a priority (low, medium, etc.) so that by default users will only see the most important questions, but users who wish to do so can configure the application with a higher degree of control. A finite state machine is also employed here (in the form of a loop) so users who want to change an answer to a previous question can do so.

For the most part, these two files determine what is written to the main fwknop configuration files in the /etc/fwknop directory (it does this in the Debian postinst file primarily using *db_get* as described in the Debian Binary section). One notable exception is the firewall generation, which is unique to this Debian package.

Since fwknop opens a port in a firewall in order to work, a default drop policy must be in place for fwknop to function properly. The configuration files contain an option for the user to create a basic firewall script (also stored in /etc/fwknop) that can optionally start on boot up.

For a default Ubuntu user to install this package, the only questions that are marked critical (these will be asked on the install) are what the password should be and whether the fwknop package should start a default drop firewall. This should be much easier for an average user than the current level of configuration options presented.

VII. DOCUMENTATION

One of the more common criticisms regarding software which is "hard to use" is that the documentation is incomplete or unclear. One of the goals of this project is to include complete and clear documentation both online and with the binary package itself.

There exists quite a bit of documentation at fwknop's home page [13]. It is imperative that this is incorporated into our package. Additionally, we have included information pertaining to the Debian packages and additional concrete examples to supplement the existing documentation. In our binary package, we placed this documentation in the standard /usr/share subdirectories.

VIII. INTEGRATION WITH SSH

One thing that can help reduce the burden on the user is to integrate seamlessly with a variety of applications. Because the most common use of fwknop is with OpenSSH, we present one solution here. This serves as an example and template from which other services, such as FTP and VPN access, may be added onto the proposed framework.

The wrapper for ssh (fwknop-ssh) is included as a separate package that is currently hosted at google code.

Fwknop-ssh's current form is a program which parses the hostname and port from OpenSSH using the Python *optparse* libraries. An alternative method is to patch OpenSSH directly to allow for this integration, but wrapping it with python causes no serious execution delays and allows for additional flexibility from a developer's standpoint.

As an example, a user can enter:

```
fwknop-ssh -X -p 33 user@192.168.1.1
```

This produces output similar to the following:

```
fwknop -A tcp/33 -s -k 192.168.1.1;  
ssh -X -p 33 user@192.168.1.1;
```

The *-s* option, which indicates a symmetric fwknop authentication key, is determined by a configuration file that has reasonable defaults generated from user configurations similar to those previously described.

Fwknop-ssh can help simplify the use of fwknop on the client by reducing options that the user needs to remember and by reducing the number of options the user must type.

Using the standard *optparse* libraries, the code to parse the ssh arguments is very simple. Below is pseudo-code demonstrating how this parsing takes place. Some details are omitted or simplified for clarity.

```
from optparse import OptionParser  
parser = OptionParser()  
  
#include all options for ssh  
parser.add_option("-1", action="store_true",  
                 dest="1", default=False,  
                 help="Forces ssh to try protocol version 1")  
parser.add_option("-2", action="store_true",  
                 dest="2", default=False,  
                 help="Forces ssh to try protocol version 2")  
parser.add_option("-4", action="store_true",  
                 dest="4", default=False,  
                 help="Forces ssh to use IPV4 only")  
...  
  
(options, args) = parser.parse_args()  
  
#include all these options as a string  
#which will eventually be passed to SSH  
for option in options:  
    sshargs += option.dest + " " + option.value
```

By default, the fwknop-ssh information is stored in a configuration file similar to that of the fwknop package. However, this file can be overridden with the *--fwconf* option. Though the *optparse* libraries certainly have many advantages, we chose to use the *getopt* libraries for this part of the code to obtain greater control. The program parses this configuration file for fwknop-specific options.

```

#optionally override the default conf file
parser.add_option("--fwconf", dest="fwconfig"
    default="/etc/fwknop-ssh.conf",
    help="Configuration file for fwknop-ssh")

#read the contents of the file into a string
confopts = open(options.fwconfig).read()

#parse the configuration file using getopt
foptions, args = getopt(confopts,
    "D:A:Rasa:R...") #all fwknop client options
for o, a in foptions:
    if o in ("-D"):
        destIP = o
    #continue setting vars for fwknop options
...

```

Additionally, the configuration file options can be overwritten with the `--fwknop` option. This allows for usage such as the following:

```

ssh --fwknop "--gpg-recv AAAAAAAAAA --gpg-sign
BBBBBBBB -A tcp/22 -R -k myserver"
user@myserver

```

This functionality is accomplished with additional option parsing.

```

#include an optional override flag
#for additional fwknop command-line arguments
parser.add_option("--fwknop",
    dest="fwknopOpts",
    help="options to pass to fwknop")

```

```

#omitted here is logic for combining this with
#the output from the configuration file
foptions += option.fwknopOpts

```

The final parsing element remaining after the arguments taken is the host and destination. This is stored in the `args` variable in the tuple above. It has the following form:

```
[user@]hostname
```

This is trivial to parse. The final piece of logic simply executes the `fwknop` and `ssh` commands sequentially using a system call and the strings parsed in the above code, using the hostname above with the `-D` option in `fwknop` and with no option in `SSH`.

When writing this wrapper, we made the design decision to guarantee that the `openSSH` options will always work. Though we considered combining the `fwknop-ssh` package with `fwknop`, we eventually decided to separate these as they have different uses and dependencies. Because we are porting both of these packages to Ubuntu, each release will test that all three packages are compatible. This means that all the options in the `OpenSSH` package provided with Ubuntu will be kept up to date in `fwknop-ssh`.

Though the code for `fwknop-ssh` mainly only consists of code to parse through and sort various options, `fwknop-ssh` has proven to greatly simplify the adoption of `fwknop`.

IX. FUTURE WORK

Though we have helped to simplify the use of `fwknop`, there is still much to do. Some future projects may include.

A. Further Integration

Integration of `fwknop` into other `SSH` software would be extremely beneficial. This may be impossible for the authors to do with many proprietary `SSH` software tools. However, there are popular `SSH` clients for Windows such as `PuTTY`, which could potentially be enhanced for smooth `fwknop` integration.

Additionally, further integration with `fwknop` is possible with a variety of applications not explored here. This integration greatly eases the use and could greatly enhance adoption.

B. Porting to Other Distributions

We have ported `fwknop` to Ubuntu. There are many Linux distributions that have unique (but similar) ways of customizing configuration files. Microsoft Windows and other operating systems are important targets as well. In this process, much of our work could be duplicated to hopefully make the work of others easier.

X. CONCLUSIONS

`SPA` is a capable and powerful authorization system, but its complexity limits real-world use to all but the highest tiers of security requirements. In this paper, we provided an equivalently secure service that people can more easily interact with.

ACKNOWLEDGMENTS

Dan Tappan, Brigitte Hodson, and Denise Duong from ISU; and Milos Manic from University of Idaho have helped a great deal in proofreading and editing this paper.

REFERENCES

- [1] P. Barham, S. Hand, R. Isaacs, P. Jardetzky, R. Mortier, and T. Roscoe, "Techniques for Lightweight Concealment and Authentication in IP Networks," Intel Research Berkeley (IRB-TR-02-009), 2002.
- [2] R. Degraaf, "Enhancing Firewalls: Conveying User and Application Identification to Network Firewalls," Computer Science Master's Thesis, Department of Computer Science, University of Calgary, 2007.
- [3] M. Doyle, "Implementing a Port Knocking System in C," Honors Thesis Defense, University of Arkansas, 2004.
- [4] J. Hess, "The Debconf Programmer's Tutorial," 2002. Available: <http://www.fifi.org/doc/debconf-doc/tutorial.html>
- [5] M. Krzywinski, "Port Knocking," in *The Linux Journal*, Jul. 2003, pp. 28-30.
- [6] S. Jeanquier, "An Analysis of Port Knocking and Single Packet Authorization," Computer Science Master's Thesis, Department of Computer Science, University of London, 2006.
- [7] C. Lee, "Debian Binary Package Building Howto: Revision 4," TLDP, Aug. 2005.
- [8] A. Narayanan, "A Critique of Port Knocking," in *Linux.com*, Aug. 2004. Available: <http://www.linux.com/articles/37888>
- [9] M. Rash, "Combining Port Knocking and Passive OS Fingerprinting with `fwknop`," in *USENIX: Login*, Dec. 2004, pp. 56-59.

- [10] M. Rash, "Linux Firewalls: Attack Detection and Response with Iptables, Psad, and Fwknop," No Starch Press, 2007, pp. 231-255.
- [11] M. Rash, "Fwknop vs Weak SSH Keys," in Fwknop Developer's List, Sept. 2008. Available: <http://cIPHERdyne.org/blog/2008/08/fwknop-vs.-weak-ssh-keys.html>
- [12] M. Rash, "Single Packet Authorization," in The Linux Journal, Apr. 2007.
- [13] M. Rash, "Fwknop Documentation," Available: <http://cIPHERdyne.org/fwknop/docs/>
- [14] J. Rodin and O. Aoki, "Debian New Maintainer's Guide," in Debian Official Documentation, Oct. 2008. Available: <http://www.debian.org/doc/maint-guide>
- [15] SecurityFocus, OpenSSH Remote Boot Authentication Timing Side-Channel Weakness, Bugtraq ID 7482, May 2003. Available: <http://www.securityfocus.com/bid/7482>
- [16] SecurityFocus, Debian OpenSSL Package Random Number Generator Weakness, Bugtraq ID 29179, May 2008. Available <http://www.securityfocus.com/bid/29179>
- [17] Simon Fraser University, "Alert: remote root exploit in openssh daemon", Jun. 2002. Available: <http://www.sfu.ca/~siebert/linux-security/msg00076.html>
- [18] Ubuntu, "The Ubuntu Packaging Guide", 2006. Available: <https://help.ubuntu.com/6.10/ubuntu/packagingguide/C/index.html>