

PA018 - Term Project - Port knocking enhancements

Stefan Miklosovic
Faculty of Informatics
Masaryk University

387090@fi.muni.cz
login: xmiklos1 uco:387090

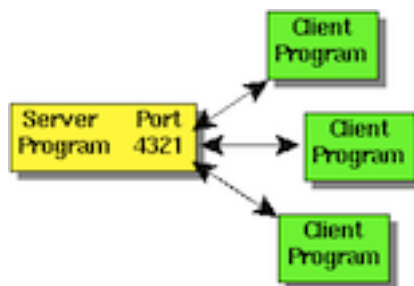
June 7, 2011

1 Port knocking and its principles

Current computer networking is vulnerable to attacks and exploits more than ever. Increasing complexity of network-oriented software opens a lot of new ways how to attack computer system. One solution how deal with attacks from outside world is indeed having a good firewall which filters unwanted and potential malicious network traffic. In these days, network firewall protecting our internal network from outside is a must.

Firewall has various tasks. First of all, it filters traffic, drop it, reject it or forward it to the other machine or network. In high abstraction, Internet traffic and its underlaying network services communicate through so called ports. Port is an application-specific or process-specific software construct serving as a communications endpoint. A specific port is identified by its number, commonly known as the port number. A port number is a 16-bit unsigned integer, thus ranging from 0 to 66535. There are two groups of ports, system port, which number is less then 1024 and user ports otherwise. In many Unix-like operating systems superuser privileges are required for creation of these ports, since these are often critical to the operation of IP networks.

From server point of view, port is like a window to the service running at server side from outside world. Clients, in general, connects to server in order to get some data from server. Server replies back.



Example 1: Client - server scenario.

Port can have two states - closed or open. All communication from clients to closed server port is just dropped. From security point, it is good idea to close all ports to the Internet. This is because services which run at our server can be exploitable or there can be a security hole which

a potential attacker can exploit and get control over our server. This is not a thing we want, for sure. This case precede in a lot of situations so called port scanning. Port scan is just a scan of all of our ports in order to getting know which ports are open and which closed. Closed ports are not interesting to attacker. Open ports are. Attacker can scan open ports and realize which services are running there. After this, attacker can try known (and unknown) vulnerabilities and our server could be exploited.

How to deal with this situation? Well, we can close all ports. But then, our services will not be accessible at all. We would like to have closed ports but in a situation when we need it, we would just open some port for that time and after our work with service is done, we can close it again.

Answer to our problem can be so-called port knocking. In computer networking terminology it is a technique which opens port(s) on a firewall of server by generating a sequence of connection attempts on a set of specific ports. When a user, knocker, who wants to connect to server, sends packets to server in correct sequence, server dynamically open a port for him. The primary purpose of port knocking is to prevent an attacker from scanning a system for potentially exploitable services by doing a port scan.

Detection of "right" knocking sequence has more than one realization. First method is based on watching a firewall log file which logs unsuccessful attempts of connection. After some pattern matching in log file, specified port is open. This port can be set in configuration or is derived from sequence of knocks by some rule. Software at server side which looks after firewall log is called port knock daemon. It is normal system unix daemon. Second approach deals with cryptography.

Consider a situation, when user has to send (= knock) three packets to server, at port 1000, 2000 and 3000. After this knocking, server opens e.g port 22 which is well known port for SSH service. In addition, client can connect to this port and normal ssh session is about to start. (Client has to send then ssh password and so on ...). Potential attacker sees deaf server. No port is open. Without knowing that three-knock sequence. He is lost. If he wanted to try each possibility of sequences, he would have to try 65535^3 combination of ports (in case he knows how much knocks is required). That's about 281 trillion packets. This is of course worst case scenario. This brute force attack can be reduced or fully stopped by e.g by detecting attacker IP address and dropping packets. We can use additional security mechanisms such as cryptographic hashes inside the port knock sequence in order to prevent network sniffing find out port knocking sequence or using traffic replay attack.

2 Secret sharing

Secret sharing is a method, how to distribute a secret amongst a group of participants. Each participant has allocated *share* of secret. The original secret can be reconstructed only when a sufficient number of shares are put together and some decryption is done.

To be more precise, in secret sharing scheme, there is two sides. One is *dealer*. Second side is formed by n *players*. At the beginning, dealer gives a secret to the players. He does this by giving each player a share in a such way that any group of t (like threshold) or more players can together reconstruct the secret but it is extremely important that no player with its own share cannot find out what the original secret is without knowledge of all other t shares. Such a system is called a t, n -threshold scheme.

There are several secret sharing schemes which are considered to be information theoretically secure but there also exist sharing schemes which give up this unconditional security for more efficiency while still maintaining enough security.

Information-theoretically secure cryptosystem is a system, in which security is derived from information theory. It says, that it is secure even when the adversary has computational unbounded computing power, for example, one-time pad is such cryptosystem.

For unconditionally secure secret sharing schemes, there are limitations as:

- a) Each share of secret must be at least as large as the secret itself. This is a result of information theory. Given $t-1$ shares, it is sure no information can be determined about the secret. So the final share must contain as much information as the secret itself.
- b) It is important that all sharing schemes use random bits. If we want to distribute a one-bit secret among threshold t people, $t-1$ random bits are necessary. Similarly, to distribute a secret of arbitrary length, entropy of $(t-1)*length$ is necessary.

If $t \neq n$ Very interesting problem of secure sharing is a fact, how to provide reconstruction of secret even when not all participants (or players) are available. Let's describe nice example. Imagine there is some company which has super-secret receipt for baking a muffins. This knowledge has to be shared among president and managers of company, but in such way, that no manager can steal that knowledge for another company. We want make sure that president of the company is able to read that receipt alone, but when some bad thing happen to president, managers can also access it and we still can bake a muffins happily. This can be solved by (t,n) scheme where n is bigger than t . E.g in case that we have $(4,10)$ scheme, we give to director 4 shares. Managers obtain 2 shares. Director can access receipt alone while two managers have to stick together in order to read it.

There are two most known sharing schemes, Shamir's and Blakley's schemes.

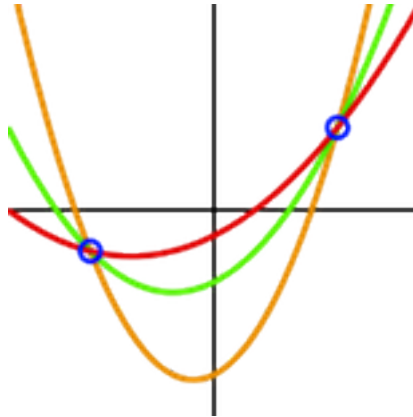
Shamir's secret sharing Shamir's secret sharing is in fact perfectly information theoretically secure. Formally, we can describe this sharing by following mathematical definition.

We want to divide data D into n parts D_1, D_2, \dots, D_n in a such way that:

- a) Knowledge of any t or more D_i parts makes D computable.
- b) Knowledge of any $t - 1$ or fewer D_i pieces leaves D completely undetermined.

This describes (t,n) threshold scheme.

The main idea behind this secret sharing lies in polynomial interpolation. It is a fact, that we only need 2 points to define a line completely. Similarly, we need 3 points, and these 3 points are sufficient, for getting parabola, or in other word, for getting polynomial of degree 2. We need only 4 points to define a cubic curve and so on. So to be general, we need k points to define a polynomial of degree $k - 1$.



Example 2: Two points are not sufficient to describe parabola.

If we want to encrypt secret S , (which is an element of finite field F), we choose $k - 1$ coefficients a_1, \dots, a_{k-1} in F randomly and we also set $a_0 = S$. Next, we build polynomial $f(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \dots + a_{k-1}x^{k-1}$. Getting this polynomial of degree $k - 1$, we compute n points out of it. E.g. setting $i = 1, \dots, n$ to get pairs $(i, f(i))$. We give these pairs to participants.

If we want to get secret, we need to perform reverse operation - polynomial interpolation - and the secret is the constant term a_0 .

Shamir's secure sharing is:

- a) secure - as described by information theoretic security
- b) minimal - each share does not exceed the size of the original data
- c) extensible - when k is not changed, new shares can be added dynamically
- d) dynamic - security can be easily enhanced without changing the secret

Blakley's secret sharing Blakley's secret sharing is very similar to Shamir's approach how to share a secret, but this secret sharing profits from different mathematical background. We do not work with points but with planes.

It is clear that two nonparallel lines in the same plane intersect at exactly one point. Further, three nonparallel planes in space intersect at exactly one point, so to be more general, any n nonparallel n -dimensional hyperplanes intersect at a specific point. We can then encrypt a secret. Secret is that point, in which planes intersect. Again, this model is perfect secure. If we distribute planes to participants, each participant only knows that the secret lies somewhere at his plane. But he does not know at all what point it is. It can be any point at this plane. If this participant contact another one and planes are joined, they only know that the secret will be somewhere at the intersection of planes - at some line. But again, they do not know which point it is. They can't know because all points at that line are possible. There has to be (in 3D), third participant, a plane, which intersect line from previous example. That point is a secret.



Example 3: Two planes are not sufficient to describe a point in three dimensions.

Blakley's secret sharing is less space efficient than Shamir's sharing. It is because Blakley's share has to be t times larger. t is the threshold number of players (participants).

3 Overview of new approach of knocking

From original concept of port knocking, there is still one person who knocks. This person has absolute power in a way of opening or closing of port. This fact is not the best approach all the time. We would like to connect more people to decision process of port opening. By this, we can have better control who access to our services running at our servers or who did what and when with our approval.

Imagine you are in a company where new colleague has arrived. Naturally, we do not trust him fully. We want to know what services he or she access while this person is outside of company's internal network. We can design our policy in way that older colleague / director of novice has to agree in some way with novice's access to service which runs at some port. And all of this has to be done remotely, through Internet.

Other situation happens during holidays. Lets say all colleagues from company are away. Server is closed from incoming connections. But we need to access some service very urgently and, as our internal policy says, we want collectively agree with service opening to outside world. How to do that? Port knocking in connection of secret sharing could be good approach.

Main principle of new approach Principle of new approach is quite simple. There is knocking daemon at server which we want to connect to and execute some action at, for example opening some port. This daemon has one share of secret. Now, some client wants to open a port. He contacts other $t-2$ clients and he informs them he wants to open a port. After individual approval of clients, these clients send their shares independently to server. Daemon sitting at the server and bound to port 10000 (by default) listens to incoming UDP packets from clients. Packets are not sent to server in some prescribed order. It does not matter. Server only checks if threshold is reached and in that case it triggers decryption of shares and, if decryption is valid and secret is parsed correctly, it executes underlying command.

It is important that server has one share. If all shares were in clients' hands, it would be possible to decrypt secret after collecting all shares from clients. Imagine there is some malicious client which want to collect all shares. If he did it successfully, he could decrypt secret on his own. In our case, even after collecting all shares, attacker is not able to decrypt it, because final share is in possession of server, which is pretty hard to get.

It is also important, that client cannot port knock many times in a row from same IP address. In model situation, we assume client has certain IP during knock. If we allowed client to knock with more than one share to server from same IP address, this user could just collect all shares from colleagues and knock $t - 1$ times with all shares from clients on his own. We want to prevent this, so we have to check source IP addresses of UDP packets which reach server.

Another issue is about time restriction. How much time such port knocking sequence should last? After arrival of first UDP packet, server starts a stopwatch. If after, say, 10 minutes, threshold is not reached with additional shares, all arrived shares are just dropped. This solution has some drawbacks. If there is some malicious actor, he could just sent shares constantly to server and if valid clients wanted to knock, they do not have "chance" to reach a threshold with valid packets, because they will be mixed with attackers' packets and consequently being dropped. But, since incoming packets are tested against original IP, this attacker would has to change IP address of each packet he sends. However, checking only IP address is not the best solution, since IP address can be relatively easily changed. At packet level, sender of share can be also identified by MAC address. This method seems to be better but we can't forget that this MAC address can be changed too.

It is also good question, for what reason attacker would broke someone's plans to open port at server. We are aware of potential risk opening a port to the outside world. So for what reason attacker does not want to allow our server to be less secure?

Knocking daemon is bound to port 10000, but it does not respond to clients at all. From clients point of view, server is just like black hole. Only when knocking is successful, server automatically open a port to the world.

All this approach is just opposite of "normal" knocking. Normally, one user knocks to several ports in order to get one specific port open. In this case, several users knocks with secret share to one specific port in order to get some port open. Open port needs to be closed. We can port knock again or after specified time, port is closed automatically. This feature is not implemented yet.

After this, one can wonder who computes the share of a participant. It is very good point. In our case, it is an administrator of computer system or server. In our scheme, there is offer one dealer of shares we use. This can be convenient, however this is also not a good approach and we would like to avoid it but it is not so easy.

It is also clear that if some participant knocks, server knows his identity (e.g. by ip address) but by our approach of knocking, no one from participants knows who else knock too. If we have threshold scheme (t, n) where t equals n , it is clear that everybody knocked. But if n is much more bigger than t , this is a question. The one who want to get access (or it may be a collective will) do not know who knocked and who did not. This ensure our anonymity as a knocking subject but on the other hand, it is problem when we need to track down concretely who did and who did not knock. This information can be kept at server side but this solution acts as additional layer to knocking when we need to protect this info on our own in some database or log.

4 Software realization

Format of secret and generation of shares What is a secret we want to protect after this all? Secret has some format, which can (well, must) be well known but after encryption, attacker can't say whats inside. Secret is simple one-line string which consists of four fields separated by semicolons. It is like:

[IP address]:[protocol]:[port]:[firewall action]

IP address - it is an IP address of server we want to knock to. After encryption, shares are distributed to participants / colleagues. Since only they personally know which server they are going to knock and which share is appropriate, even after compromising of share by attacker, he does not know where to knock.

protocol - Protocol of port we want to execute some firewall action upon. It can be "tcp" or "udp" string.

port - Port number.

firewall action - It can be string "ACCEPT" or "DENY". We apply appropriate action in order to open or close port.

Model secret could then be:

85.216.130.98:tcp:22:ACCEPT

This is not the only possibility how to make a secret and its shares. There are no restrictions what server should do after decrypting a secret or what format of secret should be. It depends on us. In our case, it is port opening and I implemented this specific case. But we could implement some different action which should be performed at server just like port opening. Imagination is the only border.

If we knock in order to open port, we also need to defined secret and related shares in order to close it. For that reason, we must generate two sets of shares. Second set of shares would contain string DENY instead of string ACCEPT in fourth field. All other fields do not change.

Shares are generated by open source implementation of Shamir's secret sharing. I choose Shamir's secret sharing instead of Blakley's sharing because of already implemented Shamir's library and because Shamir's sharing is more space efficient than Blakley's sharing. It is still possible to implement Blakley's sharing by coding it from scratch. Then server can be extended for using both sharing approaches. (Since sharing methods would be implemented in different Java threads.).

Key management After generating shares of secret, we need to get these shares to clients somehow. We can do it manually, this is the most secure way but also the slowest, or we can send it to clients through some secure channel, encrypted e-mail, (via SSL), encrypting with PGP and so on.

Good thing about Shamir's secure sharing is a fact, that we can dynamically add keys to users. So if we want to involve new user to get in touch with knocking, we just generate him a share. Also, by the time, we can easily enhanced security by changing the polynomial occasionally, but our secret will be the same. After this, we have to generate new shares and send them to users. Shares can be generated by free implementation of Shamir's secure sharing in Java, which library is located at <http://sourceforge.net/projects/secretsharejava/>.

KnockKnockClient KnockKnockClient is simple application written in Java language. After initializing from command line or config file, it sends one UDP packet to server on specified IP address and port and it terminates. UDP packet contain just encrypted share of client.

Individual share is stored in file at clients' side. Since I implemented it for unix-like systems (Linux to be more specific), by default, file with encrypted secret is stored in home folder of user, in `~/.knockc/share`. We can be more specific about storing of share. By command line argument or by external configuration file `~/.knockc/knockc.cfg`.

KnockKnockServer KnockKnockServer is simple daemon written in Java language. This daemon runs at server and listens to clients knocks - shares. Upon server's start, daemon process read configuration from external configuration, typically from `/etc/knockd/knockd.cfg`. After daemon initializing and binding on some port, by default on port 10000, for incoming shares. By default, file with encrypted secret of server is stored in `/etc/knockd/share`.

The main server logic is located in loop in one thread, in class `KnockKnockLogic`. Block structure of this loop has the following structure:

```
while (everything is ok) {
    while (packet has not arrive) {
        listen for incoming packet;
    }
    if (elapsed time < timeout) {
        if (count of shares in buffer < threshold) {
            add share to share buffer;
            add address of incoming packet to address buffer
        }
        else {
            clear buffer of addresses and shares;
            restart stopwatch;
            add share and address from incoming packet to buffers;
        }
        if (count of shares in buffer == threshold)
            command = decrypt(buffer of shares)
            if (decryption is successful) {
                execute command
            } else raise error
        }
    }
    else {
        clear buffer of addresses and shares;
        restart stopwatch;
        add client's share and address to buffers;
    }
}
```

5 Remarks

After implementing, I would like to write about drawbacks and possible solutions to related problems. First of all, if we open port, in current implementation, it is open for everyone. It would be very nice, if server was accessible only clients who knocks. This can be implemented by it is little bit tricky. Second, there is an issue about closing a port. We can close it by hand at server while connected, we can knock second time but by closing shares or there could be timeout after which port is closed automatically. If we wanted to close port, it should not be necessary to involve all people who knocks in order to open it. One user's knock should be sufficient for closing a port.

6 Summary

Port knocking with addition of secret sharing can be very interesting security enhancement for our servers. After all of this, use cases and model situation for using this approach are not very likely to be happened, but in a situation we want to have security policy of this kind, we can apply it and I believe this method of port knocking will find its place under the sun of secure enhancements and access control.

7 Resources

Papers:

Shamir, Adi (1979), "How to share a secret", Communications of the ACM 22 (11): 612613

Desmedt, Yvo, "Some Recent Research Aspects of Threshold Cryptography"

Online resources:

http://en.wikipedia.org/wiki/Port_knocking

<http://sourceforge.net/projects/secretsharejava/>

http://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing

http://en.wikipedia.org/wiki/Secret_sharing

<http://www.fi.muni.cz/~xmiklos1/knock.pdf>

<http://www.portknocking.org/>

Source code available (heavy beta state) at: <https://github.com/smiklosovic/>