

Forum:[Hack Forums](#)

Thread: [Build your own shell for linux](#)

deltaxflux - July 17, 2017, 1:56 am

Hey guys

I build my own shell written in cpp. The shell is fast and support basic custom commands and bash. It is modular and support easy extenibility

Features Random string generator
Shell lock with random string Debug log
Modular build Colors
Fast

Moduls

* Arp spoofing integrated

Build

PHP Code

<?php

g++ <source> -o Shell

./Shell

Source

<http://lpaste.net/356975>

Phizo - July 17, 2017, 3:31 am

This is honestly not very good. Not only is XOR encryption weak and trivially broken, you seed the PRNG wrong, as you call it every time your RandomString() function is called, which causes the same value to be returned each invocation. It is hence not random. You must only seed RNGs once for a given application. As for how trivial it is to recover the key given XOR-encrypted ciphertext, it is the same as breaking a polyalphabetic substitution cipher. You determine the period p (length of the key) using the Index of Coincidence, then group each ciphertext character $c[n*p + i]$ for all n, i such that $0 \leq n < \text{length}[c] / p$ and $0 \leq i < p$, and then solve each group as a monoalphabetic substitution cipher, which is trivial with frequency analysis. This is introductory cryptanalysis. You can only do this if $\text{length}[k] \leq \text{length}[c] / 2$ for a key k, though, which is typical. Otherwise, only by brute-force. All this aside, the shell is still not very good. I had to write a much more advanced one as an assignment once when I was at university, so I do know what I am talking about. It's good to have little projects to become accustomed to the language, but you must be aware that there are a lot of problems with your solution. I'm not trying to shit on your work. Best of luck for the future.

Dostoevsky - July 17, 2017, 4:07 am

Some of the code could do with some work (along with the points Phizo makes about the actual logic behind what you're doing).

I recommend passing around your std::string instances by (const) reference. Right now you're essentially copying every string instance when you pass it around. This is obviously bad for performance reasons and, since I imagine the strings you're using in your program to be relatively short, I can't tell you to even move them given the nature of moving of SSO'd strings. So, in cases where you're using the strings as simply immutable, pass by const reference.

In your encrypt function, you can avoid using types like `std::string::size_type` by simply replacing the instance of it with `auto`.

In your lock function, you should replace the `while(1)` with `while(true)`.

Where you compare a `std::string` with an empty string literal, you could simply use `std::string's empty()` method to check that instead of invoking a non-member comparison overload.

Similarly, later on, you assign a `std::string` an empty string literal. That's why the `.clear()` method exists.

In main, simply declaring your `std::string` should call the default constructor (since you're not initialising it). There's no need for explicitly doing so with the empty uniform initialisation.

Smaller petty notes would be:

- In most cases there's no need to use `std::endl` since flushing the input buffer is often undesirable (by that I mean: you don't always need to and most beginners aren't aware of that nature of `std::endl`). Just use a `'\n'` character appended to ostream.
- You should show intention with more use of `const` and in another case where you pass a signed `int` to `RandomString` to indicate length. A length of a string can never be negative thus an unsigned data type such as `size_t` would be more appropriate for showing intention.
- I dislike any code that uses the `std` namespace in the scope that you have. I only allow it if it's for nested things within methods or something (where the scope is limited). E.g. for chrono literals (`std::literals::chrono_literals`) or for placeholders when used w/ `std::bind`.

deltaxflux - July 17, 2017, 8:23 am

(Jul 17, 2017 03:31 AM)Phizo Wrote: [\[CITE\]](#) This is honestly not very good. Not only is XOR encryption weak and trivially broken, you seed the PRNG wrong, as you call it every time your `RandomString()` function is called, which causes the same value to be returned each invocation. It is hence not random. You must only seed RNGs once for a given application. As for how trivial it is to recover the key given XOR-encrypted ciphertext, it is the same as breaking a polyalphabetic substitution cipher. You determine the period p (length of the key) using the Index of Coincidence, then group each ciphertext character $c[n \cdot p + i]$ for all n, i such that $0 \leq n < \text{length}[c] / p$ and $0 \leq i < p$, and then solve each group as a monoalphabetic substitution cipher, which is trivial with frequency analysis. This is introductory cryptanalysis. You can only do this if $\text{length}[k] \leq \text{length}[c] / 2$ for a key k , though, which is typical. Otherwise, only by brute-force. All this aside, the shell is still not very good. I had to write a much more advanced one as an assignment once when I was at university, so I do know what I am talking about. It's good to have little projects to become accustomed to the language, but you must be aware that there are a lot of problems with your solution. I'm not trying to shit on your work. Best of luck for the future.

Thanks for your replay. I start learning cpp for couple of weeks ago. I already are fam. with other languages like bash python and webdevelopment. I know that the XOR encryption was not the best idea I will replace it later with an aes encryption. I maybe got the point wrong with the `RandomString` it differs every time I call the function:

PHP Code

```
<?php
postfaktisch in X
$ rand
XFSw0hfV00x7wKrQOI9EeGSUujNwKwyt
postfaktisch in /home/deltax/dash/src
$ rand
QQnrCppeeHhUxSCvMa9sCL6MfJ7HES2e
postfaktisch in X
```

In general this shell is very basic and doesn't support much, I know but it is for me a start to get into cpp and it is intended for people who want to build their own shell if it doesn't matter there are design issues. I will definitely fix the issues but help is every time appreciated. If you have time you can still fix bugs if you want to. In general I like script languages more but these are very slow compared to c / cpp. I made a benchmark a couple of weeks ago. The task was that the program write an int into a file for one second. I found following out: Bash 700 lines

Python 3 mio lines

* C / Cpp 9 mio lines

Anyway I still want to develop in C or Cpp. It is still a matter of time.

Appendix:

Thank you for your feedback.

Regards,
deltax

(Jul 17, 2017 04:07 AM)Dostoevsky Wrote: [\[CITE\]](#) Some of the code could do with some work (along with the points Phizo makes about the actual logic behind what you're doing).

I recommend passing around your `std::string` instances by (const) reference. Right now you're essentially copying every string instance when you pass it around. This is obviously bad for performance reasons and, since I imagine the strings you're using in your program to be relatively short, I can't tell you to even move them given the nature of moving of SSO'd strings. So, in cases where you're using the strings as simply immutable, pass by const reference.

In your `encrypt` function, you can avoid using types like `std::string::size_type` by simply replacing the instance of it with `auto`.

In your `lock` function, you should replace the `while(1)` with `while(true)`.

Where you compare a `std::string` with an empty string literal, you could simply use `std::string's empty()` method to check that instead of invoking a non-member comparison overload.

Similarly, later on, you assign a `std::string` an empty string literal. That's why the `.clear()` method exists.

In main, simply declaring your `std::string` should call the default constructor (since you're not initialising it). There's no need for explicitly doing so with the empty uniform initialisation.

Smaller petty notes would be:

- In most cases there's no need to use `std::endl` since flushing the input buffer is often undesirable (by that I mean: you don't always need to and most beginners aren't aware of that nature of `std::endl`). Just use a `'\n'` character appended to `ostream`.
- You should show intention with more use of `const` and in another case where you pass a signed int to `RandomString` to indicate length. A length of a string can never be negative thus an unsigned data type such as `size_t` would be more appropriate for showing intention.
- I dislike any code that uses the `std` namespace in the scope that you have. I only allow it if it's for nested things within methods or something (where the scope is limited). E.g. for chrono literals (`std::literals::chrono_literals`) or for placeholders when used w/ `std::bind`.

Thanks for your feedback. I will improve the performance of the shell anyway I have following questions.

It is in general not a good idea to use namespace?

Better use "\n" than endl at the end?
Why is while(true) better than while(1) ?

Thanks in advance

Regards,
deltax

0xDEAD10CC - July 17, 2017, 10:38 am

You had the right idea with a `DEBUG_LEVEL` if that's your approach, but executed it poorly. What's the point of having it as a level if you only use 1 and 0? I would've used it to indicate the verbosity of the debug messages used, and had a function which checks the level so you can determine what the severity of the debug message calls throughout your code.. I didn't spend too much time looking at anything else, by the comments above I wasn't too interested and by some of the members who have already posted I'm sure they covered quite a bit for you already.

deltaxflux - July 17, 2017, 11:19 am

(Jul 17, 2017 10:38 AM)0xDEAD10CC Wrote: [\[CITE\]](#) You had the right idea with a `DEBUG_LEVEL` if that's your approach, but executed it poorly. What's the point of having it as a level if you only use 1 and 0? I would've used it to indicate the verbosity of the debug messages used, and had a function which checks the level so you can determine what the severity of the debug message calls throughout your code.. I didn't spend too much time looking at anything else, by the comments above I wasn't too interested and by some of the members who have already posted I'm sure they covered quite a bit for you already.

Well the `DEBUG LEVEL` are used for verbosity. If you use the debug mode the shell is faster and you get errors and warnings which are hide from the code but I haven't much use for this yet.

verbosity makes only sense for me if I have more source with more possibl. failure.

0xDEAD10CC - July 17, 2017, 11:53 am

(Jul 17, 2017 11:19 AM)deltaxflux Wrote: [\[CITE\]](#) Well the `DEBUG LEVEL` are used for verbosity. If you use the debug mode the shell is faster and you get errors and warnings which are hide from the code but I haven't much use for this yet.

verbosity makes only sense for me if I have more source with more possibl. failure.
"Well the `DEBUG LEVEL` are used for verbosity"

No it's not. In your case it's either a 1 or 0 for ON or OFF. There's no concept of changing the level of debug message verbosity here. Conceptually there is absolutely no way the "shell" is faster with debug mode on either... Explain how that's possible when it means more messages need to be written?

Your code has lots of potential for failure, but that's not entirely the point of debug messages. You can use a debug level of 1 to write informal messages which may not indicate runtime problems, and 2 for warnings, 3 for serious problems for example (exceptions if you use them, etc...), with the highest level of debug level hierarchy allowing for all less minor messages to be written as well.

deltaxflux - July 17, 2017, 1:47 pm

(Jul 17, 2017 11:53 AM)0xDEAD10CC Wrote: [\[CITE\]](#) "Well the `DEBUG LEVEL` are used for verbosity"

No it's not. In your case it's either a 1 or 0 for ON or OFF. There's no concept of changing the level

of debug message verbosity here. Conceptually there is absolutely no way the "shell" is faster with debug mode on either... Explain how that's possible when it means more messages need to be written?

Your code has lots of potential for failure, but that's not entirely the point of debug messages. You can use a debug level of 1 to write informal messages which may not indicate runtime problems, and 2 for warnings, 3 for serious problems for example (exceptions if you use them, etc...), with the highest level of debug level hierarchy allowing for all less minor messages to be written as well.

Yes I understand that but I will improve first of all the encryption and fix possible failure and later but not last I will improve the debug logs.

Phizo - July 18, 2017, 10:23 am

(Jul 17, 2017 08:23 AM)deltaxflux Wrote: [\[CITE\]](#) Thanks for your replay. I start learning cpp for couple of weeks ago. I already are fam. with other languages like bash python and webdevelopment. I know that the XOR encryption was not the best idea I will replace it later with an aes encryption. I maybe got the point wrong with the RandomString it differs every time I call the function:

```
PHP Code
<?php
postfaktisch in X
$ rand
XFSw0hfV00x7wKrQOI9EeGSUujNwKwyt
postfaktisch in /home/deltax/dash/src
$ rand
QQnrCpopeHhUxSCvMa9sCL6MfJ7HES2e
postfaktisch in X
```

In general this shell is very basic and doesn't support much, I know but it is for me a start to get into cpp and it is intended for people who want to build their own shell if it doesn't matter there are design issues. I will definitely fix the issues but help is every time appreciated. If you have time you can still fix bugs if you want to. In general I like script languages more but these are very slow compared to c / cpp. I made a benchmark a couple of weeks ago. The task was that the program write an int into a file for one second. I found following out: Bash 700 lines

Python 3 mio lines
* C / Cpp 9 mio lines
Anyway I still want to develop in C or Cpp. It is still a matter of time.

Appendix:

Thank you for your feedback.

Regards,
deltax

Thanks for your feedback. I will improve the performance of the shell anyway I have following questions.

It is in general not a good idea to use namespace?
Better use "\n" than endl at the end?
Why is while(true) better than while(1) ?

Thanks in advance

Regards,
deltax

I should have been more specific. As you reseed the PRNG every time you invoke RandomString() with the same value, you acquire the same sequence. This makes the output predictable. Your output is only seemingly random due to the time it takes you type "rand" into your shell, as time(NULL) returns the current time, which is used as the seed. Allow me to demonstrate.

This code in main():

Produces the following output on my system:

```
Code phizo@jizzo /tmp> g++ shell.cpp -Wall; ./a.out
shell.cpp:181:9: warning: unused variable 'C' [-Wunused-variable]
    int C = 1;
    ^
```

1 warning generated.

```
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
ZpCAGsNRcBWOL6x7S7ZeuTSjrPiLwljF
```

Notice that the first 10 values are the same because of how quickly RandomString(), and hence rand(), were called. You can see the last string is different because I halted execution for a second to allow my clock's value to change (UNIX/epoch time). If you seed the PRNG once (by placing srand(time(NULL)) in main()), as you are supposed to in this context, you will acquire a pseudo-random sequence as expected:

```
Code phizo@jizzo /tmp> g++ shell.cpp -Wall; ./a.out
shell.cpp:181:9: warning: unused variable 'C' [-Wunused-variable]
    int C = 1;
    ^
```

1 warning generated.

```
6w3excCaEHY7uHsVIM2RH5Tc2yY6Svxb
VAVnNfbqDEGCMpLNus0Wo9sVnNUxYIIN
miiarOgi55os0ctcBBtB7jLxu3xvEQP0
QYe1WgZaxl0ilUISgjQ4lfGixUCX4Xti
L2SLWVO2SaidvQbn3tE5cWKNXj5aWVLe
TjYRuVYomgyHOMGI5CjImREudsOq2IEp
qX5MthdG2mk0qjEfhxZs61ryc2wWNMr1
s2vmSfqWt8mpOcQauPtCvROAKiVqOs7H
pQF3upJbk2qPwGTLrFPAIGW712MCh5QO
9w4lBb9Z6DA1yGKKanqlYGb8aSyQaRXd
xoUUREyKfBVdglsgHv05AHEYQw26opWL
```

deltaxflux - July 23, 2017, 12:08 pm

(Jul 18, 2017 10:23 AM)Phizo Wrote: [\[CITE\]](#) I should have been more specific. As you reseed the PRNG every time you invoke RandomString() with the same value, you acquire the same sequence. This makes the output predictable. Your output is only seemingly random due to the time it takes

you type "rand" into your shell, as time(NULL) returns the current time, which is used as the seed. Allow me to demonstrate.

This code in main():

Produces the following output on my system:

```
Code phizo@jizzo /tmp> g++ shell.cpp -Wall; ./a.out
shell.cpp:181:9: warning: unused variable 'C' [-Wunused-variable]
```

```
    int C = 1;
    ^
```

1 warning generated.

```
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
ZpCAGsNRcBWOL6x7S7ZeuTSjrPiLwljF
```

Notice that the first 10 values are the same because of how quickly RandomString(), and hence rand(), were called. You can see the last string is different because I halted execution for a second to allow my clock's value to change (UNIX/epoch time). If you seed the PRNG once (by placing srand(time(NULL)) in main()), as you are supposed to in this context, you will acquire a pseudo-random sequence as expected:

```
Code phizo@jizzo /tmp> g++ shell.cpp -Wall; ./a.out
shell.cpp:181:9: warning: unused variable 'C' [-Wunused-variable]
```

```
    int C = 1;
    ^
```

1 warning generated.

```
6w3excCaEHY7uHsVIM2RH5Tc2yY6Svxb
VAVnNfbqDEGCMpLNus0Wo9sVnNUxYIIN
miiiaROgi55os0ctcBBtB7jLxu3xvEQP0
QYe1WgZaxI0ilUISgjQ4IfGixUCX4Xti
L2SLWVO2SaidvQbn3tE5cWKNXj5aWVLe
TjYRuVYomgyHOMGI5CjImREudsOq2IEp
qX5MthdG2mk0qjEfhxZs61ryc2wWNMr1
s2vmSfqWt8mpOcQauPtCvROAKiVqOs7H
pQF3upJbk2qPwGTLrFPAIGW712MCh5QO
9w4IBb9Z6DA1yGKKanqIYGb8aSyQaRXd
xoUUREyKfBVdglsgHv05AHEYQw26opWL
```

Thank you for your detailed report. I will rewrite this function so it is random not just pseudo random.

(Jul 18, 2017 10:23 AM)Phizo Wrote: [\[CITE\]](#) I should have been more specific. As you reseed the PRNG every time you invoke RandomString() with the same value, you acquire the same sequence. This makes the output predictable. Your output is only seemingly random due to the time it takes you type "rand" into your shell, as time(NULL) returns the current time, which is used as the seed. Allow me to demonstrate.

This code in main():

Produces the following output on my system:

```
Code phizo@jizzo /tmp> g++ shell.cpp -Wall; ./a.out
shell.cpp:181:9: warning: unused variable 'C' [-Wunused-variable]
```

```
    int C = 1;
```

```
    ^
```

1 warning generated.

```
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
33KI7N8fOu5mA3oEHAn5HUrAdCv8DrV1
ZpCAGsNRcBWOL6x7S7ZeuTSjrPiLwljF
```

Notice that the first 10 values are the same because of how quickly RandomString(), and hence rand(), were called. You can see the last string is different because I halted execution for a second to allow my clock's value to change (UNIX/epoch time). If you seed the PRNG once (by placing srand(time(NULL)) in main()), as you are supposed to in this context, you will acquire a pseudo-random sequence as expected:

```
Code phizo@jizzo /tmp> g++ shell.cpp -Wall; ./a.out
shell.cpp:181:9: warning: unused variable 'C' [-Wunused-variable]
```

```
    int C = 1;
```

```
    ^
```

1 warning generated.

```
6w3excCaEHY7uHsVIM2RH5Tc2yY6Svxb
VAVnNfbqDEGCMpLNus0Wo9sVnNUxYIIN
miiarOgi55os0ctcBBtB7jLxu3xvEQP0
QYe1WgZaxI0ilUISgjQ4IfGixUCX4Xti
L2SLWVO2SaidvQbn3tE5cWKNXj5aWVLe
TjYRuVYomgyHOMGI5CjImREudsOq2IEp
qX5MthdG2mk0qjEfhxZs61ryc2wWNMr1
s2vmSfqWt8mpOcQauPtCvROAKiVqOs7H
pQF3upJbk2qPwGTLrFPAIGW712MCh5QO
9w4IBb9Z6DA1yGKKanqIYGb8aSyQaRXd
xoUUREyKfBVdglsgHv05AHEYQw26opWL
```

Thank you for your detailed report. I will rewrite this function so it is random not just pseudo random.