Please Put OpenSSL out of Its Misery

OpenSSL must die, for it will never get any better.

Poul-Henning Kamp

The OpenSSL software package is around 300,000 lines of code, which means there are probably around 299 bugs still there, now that the Heartbleed bug—which allowed pretty much anybody to retrieve internal state to which they should normally not have access—has been fixed.

That's really all you need to know, but you also know that won't stop me, right?

Securing a computer network connection is not really hard in theory. First you let exceptionally skilled cryptographers design some cryptographic building blocks. You will need a good hash-function, a good symmetric block cipher, and a good asymmetric cipher. Next you get exceptionally skilled crypto-protocol designers to define how these building blocks should be tied together in a blow-by-blow fashion. Then an exceptionally skilled API designer defines how applications get access to the protocol, via a well-thought-out and error-resistant API with well-chosen and reliable default values and a good error reporting mechanism. Then exceptionally skilled programmers implement the algorithms and protocols according to the API in high-quality, fully-audited and analyzed library source code. And after that the application programmer—who's usually anything but exceptionally skilled—finally gets to write code to open a secure connection.

But we're not quite done.

We need to ensure that the compiler correctly translates the high-level language to machine instructions. And only exceptionally skilled compiler programmers will do! We also need to make sure that the computing environment is trustworthy—there can be no bugs, mistakes, backdoors, or malware in system libraries or in the operating system kernel. And written by? You got it, exceptionally skilled kernel programmers obviously! Of course this is all in vain if the CPU does not execute instructions faithfully, but exceptionally skilled CPU engineers will no doubt see to that.

Now—finally—we can securely transmit a picture of a cat from one computer to another.

That wasn't so bad, was it?

Specifying a dream team as I did is not without pitfalls, as OpenSSL contains this code:

```
/*  
 * The aim of right-shifting md_size is so that the compiler doesn't  
 * figure out that it can remove div_spoiler as that would require it
```

```

    * to prove that md_size is always even, which I hope is beyond it.
    */
    div_spoiler = md_size >> 1;
    div_spoiler <=< (sizeof(div_spoiler)-1)*8;
    rotate_offset = (div_spoiler + mac_start - scan_start) % md_size;

```

Be honest—would you have considered a proven, correct, compiler improvement a security risk before I showed you that ?

Of course you wouldn't! It was proven correct, wasn't it?

And that's where our security model, if you can call it that, breaks down. There is so much code involved, that no one has a clue if it all works or not.

On my computer the numbers are roughly:

Operating system Kernel:	2.0 million lines of code
Compiler:	2.0 --/--
C language runtime library:	0.5 --/--
Cryptolibrary:	0.3 --/--

Each additional programming language involved will add about a million lines. Using a graphical user interface roughly doubles the number, and a browser almost doubles it again. A reasonable estimate is that the computer on which you read this has at least 100, but more likely 1,000 bugs through which your security can be compromised. If those bugs were unique to your computer, that wouldn't be too bad. But that's not the case, as they are the exact same bugs on millions and millions of computers, and, therefore, every bug has pandemic potential. And—as if the task were not difficult enough—there are people actively trying to sabotage the result, in order to make “intelligence gathering” easier and less expensive. Apparently some of those exceptionally skilled engineers and scientists on our dream-team have hidden agendas—otherwise the leadership of the NSA would be deeply incompetent.

Then there are Certificate Authorities. Their job is to act as “trust-anchors” so that once you have established a secure connection you also know who you are talking to.

CA's trustworthiness is a joke.

Do you trust

TÜRKTRUST B LG LET M VE B L M GÜVENL H ZMETLER A . .

to authenticate a connection to your bank?

You don't even know what "GÜVENLİ HİZMETLER" means, right?

I certainly don't, and I don't trust them either!

In August 2012 they issued bogus certificates with which the holder could claim to be Google. TÜRKTRUST, of course, maintains that it was all "a one-time mistake," that "no foul play was involved" and that "it can never happen again."

Nobody believes a word of it.

Yet, your browser still trusts them by default, just as it trusts hundreds of other more or less suspect organizations to tell the truth. So we're connecting using bloated buggy software, trusting shady government-infiltrated CAs to assure us who we talk to?

There must be a better way, right?

Well there isn't.

We have never found a simpler way for two parties with no prior contact, to establish a secure telecommunications channel. Until the development of asymmetric cryptography (1973-1977) it was thought to be impossible. And if you want to know who is at the other end, the only way is to trust some third party who claims to know. No mathematical or cryptographic breakthrough will ever change that, given that our identities are social constructs, not physical realities.

So if we want e-commerce to work, we have to make the buggy code work and we should implement something more trustworthy than the "CA-Mafia."

And that brings me back to OpenSSL—which sucks. The code is a mess, the documentation is misleading, and the defaults are deceptive. Plus it is 300,000 lines of code that suffer from just about every software engineering ailment you can imagine:

- No central architectural authority
- 6,740 `goto` statements
- Inline assembly code
- Multiple different coding styles
- Obscure use of macro preprocessors
- Inconsistent naming conventions
- Far too many selections and options
- Unexplained dead code
- Misleading and incoherent comments

and so on and so on.

And it's nobody's fault.

No one was ever truly in charge of OpenSSL, it just sort of became the default landfill for prototypes of cryptographic inventions, and since it had everything cryptographic under the sun (somewhere, if you could find out how to use it), it also became the default source of cryptographic functionality.

I'm sure more than one person has thought "Nobody ever got fired for using OpenSSL"

And that is why everybody is panicking on the Internet as I write this.

This bug was pretty bad, even as bugs in OpenSSL go, but my co-columnist at *ACM Queue*, Kode Vicious, managed to find a silver lining: "Because they used a 'short' integer, only 64 kilobytes worth of secrets are exposed."

And that is not the first nor will it be the last serious bug in OpenSSL, and, therefore, OpenSSL must die, for it will never get any better.

We need a well-designed API, as simple as possible to make it hard for people to use it incorrectly. And we need multiple independent quality implementations of that API, so that if one turns out to be crap, people can switch to a better one in a matter of hours.

Please.

LOVE IT, HATE IT? LET US KNOW

feedback@queue.acm.org

Poul-Henning Kamp (phk@FreeBSD.org) is one of the primary developers of the FreeBSD operating system, which he has worked on from the very beginning. He is widely unknown for his MD5-based password scrambler, which protects the passwords on Cisco routers, Juniper routers, and Linux and BSD systems. Some people have noticed that he wrote a memory allocator, a device file system, and a disk encryption method that is actually usable. Kamp lives in Denmark with his wife, son, daughter, about a dozen FreeBSD computers, and one of the world's most precise NTP (Network Time Protocol) clocks. He makes a living as an independent contractor doing all sorts of stuff with computers and networks.

© 2014 ACM 1542-7730/14/0400 \$10.00