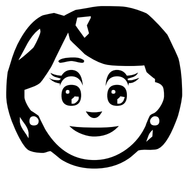
toStaticHTML() for Everyone!

About DOMPurify, Security in the DOM,
and Why We Really Need Both

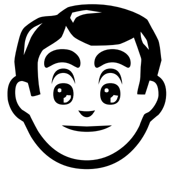
A talk by Dr.-Ing. Mario Heiderich, Cure53
mario@cure53.de || @0x6D6172696F

Here is Alice.
She wants to write an encrypted message.
And send it to Bob.



This is my company

Here is Bob.
He wants to be able to read what Alice has to write.
About NASCAR.



That's nothing to be ashamed about.
But still privacy-relevant for many!

Both Bob and Alice
want to exchange encrypted mails
with each other.

👦 ✉ 👩

But I guess you
know that already

Bob and Alice both use, let's call it… “ElectronMail”.

A fancy tool that allows to encrypt and decrypt mail messages
Right in the browser.

👦⚡ ✉ ⚡👩

You probably know
what I am referring to.

“ElectronMail” is great.

All the server sees is an encrypted piece of text.
Useless to anyone without the key.

👨🔑⚡ ✉ ⚡🔑👩

Unless that person is **really** good at mathematics

“ElectronMail” chose a very smart approach.

We refer to that as End-to-End encryption. Or E2E.

From a technical standpoint,
that whole process is easy to describe.

👨 🔐 👩

The next day, both Bob and Alice
receive an encrypted mail from Mallory.

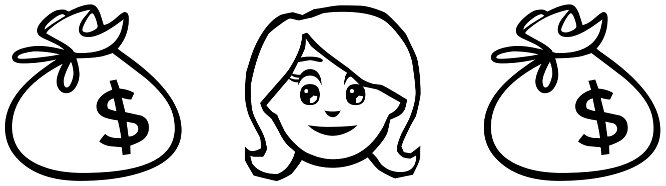
They both open the mail.

**Both their accounts get compromised
without them knowing.**

👨 💣 👱‍♀️ 💣 👩

Mallory not only gets access to all mails Bob and Alice have exchanged in the past...

but also grabs their private keys,
all contacts in Alice and Bob's address-books,
and installs a key-logger. Just in case.



Mallory isn't overly great at mathematics.

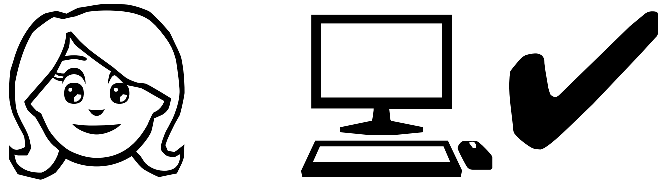
She doesn't know much about
buffer overflows and UAFs.

She is not in possession of a vast attack infrastructure, number crunchers or any dedicated soft- or hardware.

⚛✖ 👩

Mallory is pretty good at HTML and JavaScript.

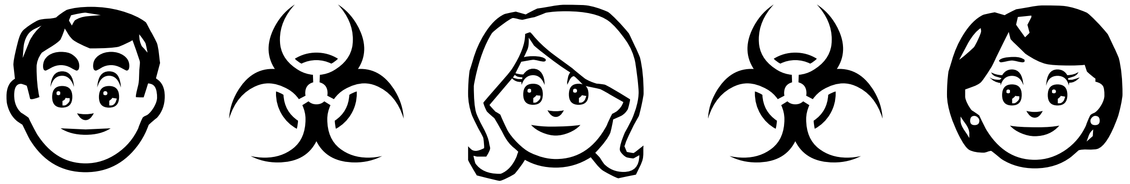
The modern cyber-crime action-pack.



There. Cyber.
I said it.

Mallory made use of a classic **Cross-Site Scripting** attack
and smuggled executable client-side code
into the message she sent out.

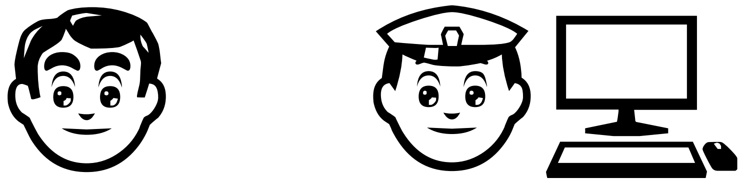
Executing in the **DOM** used by “ElectronMail”,
provided by the browsers Alice and Bob use.



Remember, when folks said "XSS is lame"?
Pepperidge Farm remembers!

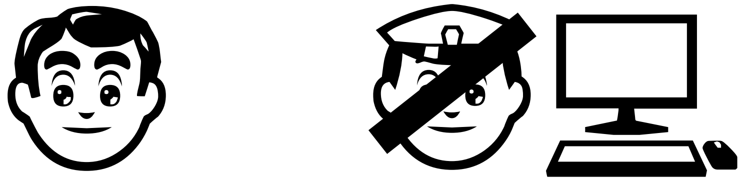
It was harder for Mallory
to do that kind of thing in the past.

Mail providers, before the dawn of cryptography in the browser, relied on strong server-side filters to scrub anything bad from HTML email bodies.



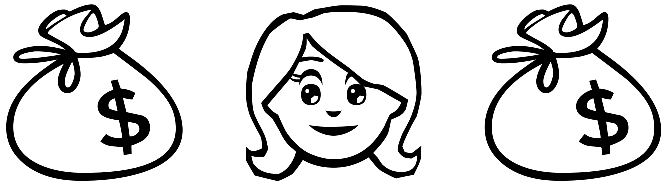
But with encryption in the browser, those days are over.
The server can no longer see,
if there is anything bad in the mail body.

Because it's encrypted.



Damn you, encryption!

Enabler of crime and malice!
All that should be forbidden! Or back-doored!



Damn you, encryption!

Enabler of crime and malice!
All that should be forbidden! Or back-doored!

Well, maybe not!

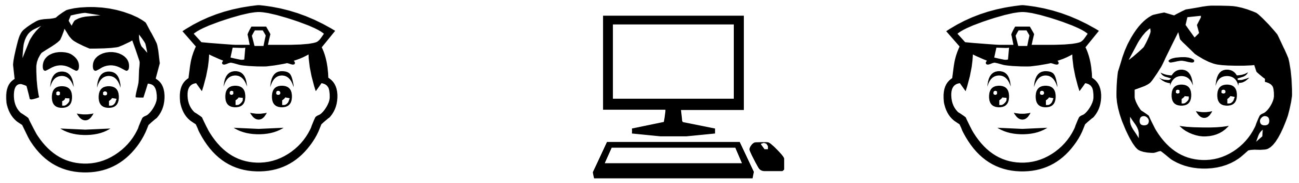
💰 👩 💰

Let's isolate our initial problem.

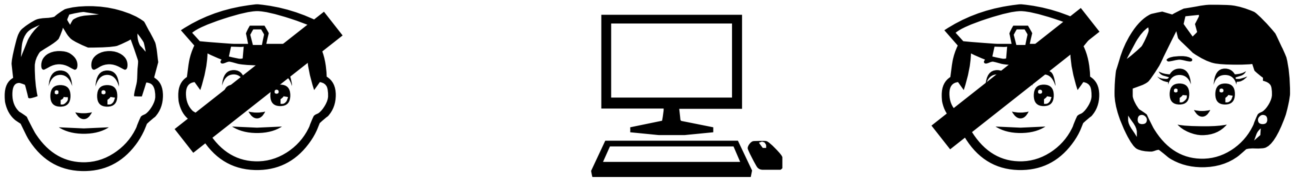
We cannot sanitize on the server any more.
Encrypted mails are left to be sanitized on the client.

So we need to sanitize on the client.

And browsers surely give us the right tools for that, correct?



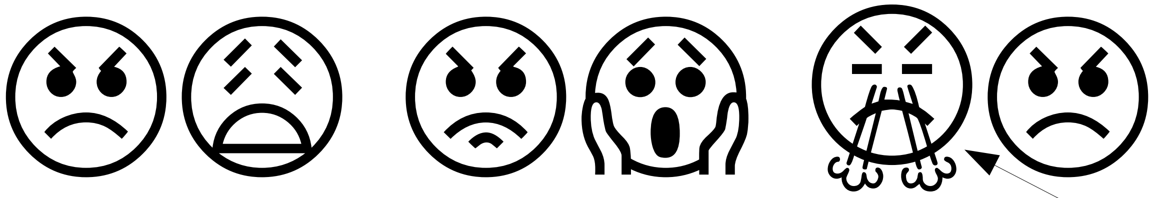
Well, sadly, they do not.



You will now jump up and start screaming.

“Sandboxed Iframes!” “HTTP Only cookies!”
“XSS filters!” “Web Workers!”
“The Same Origin Policy!” “Sub-Resource Integrity!”

And of course the cure for everything -
“Content Security Policy!”



That's Mike West!

But the more you look at those features,
the more you realize that they come close to solving our problem

But never manage to be a perfect solution.

😐😐 😐😐 😤😐

Trust me, I am a Doctor.
I have butter on my bread
Just because of that.

We are missing something that does,
what the servers did in the past.

We need to be able to tell apart the bad from the good
and only leave the good to be shown in the browser.

We don't have that.

😶😶 😶😶 😐😶

Well, we kinda do. It's called *toStaticHTML()*
and it does exactly that.

Take a string, throw any bad HTML out
and *hopefully* return a sanitized string.

😍😉 😌😒 😐😋

Mike is not amused.

But it's only available in MSIE.

Or on Firefox with NoScript installed.
So let's scratch that.

We need a *toStaticHTML()* for everyone!
In all browsers. Now!

😶😶 😶😶 😐😶

So, I created it.

It's called **DOMPurify** and its task is to do exactly what we need. Take a string of HTML. Or SVG. Or MathML. Analyze it using an isolated DOM.

Throw out the bad, leave in the good. Return a *sane* result.

😶😶 😶😶 😐😶

The browser DOM and it's really messy properties were one of the biggest problem here.

DOM Clobbering attacks, inhomogeneous APIs, HTML elements implemented in completely different ways, different attribute handling.

The DOM is a mess!

Yeah! Let's make Visual Basic Script great again!

⚇⚇ ⚇⚇ 😐⚇

But we believe to have them solved
and haven't received a bypass in months.

Despite generous bug bounty in case an issue gets spotted.

😍😌 😇😒 😐😍

Many websites already use DOMPurify and are happy with it.

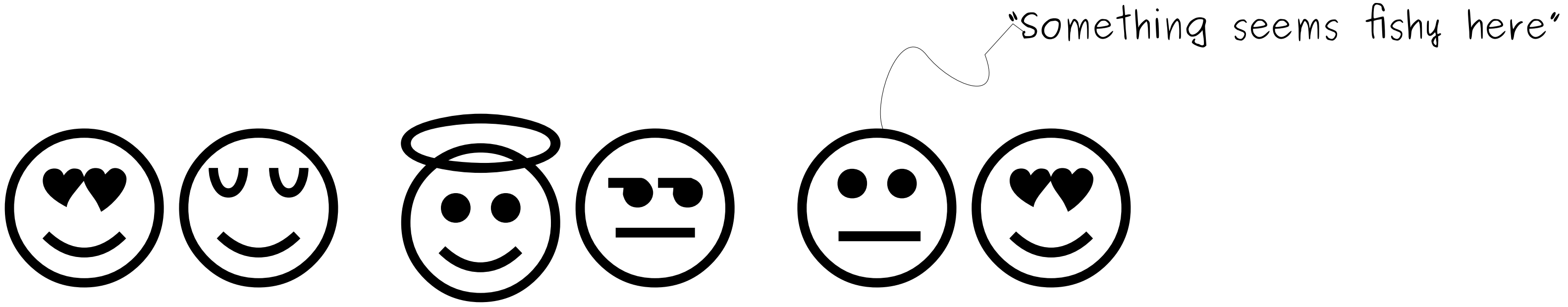
In Germany for example, we have “Gov-Approved Mail” system called “de-mail”.

They had the same problem and guess how even they solved it.
That's right, DOMPurify.

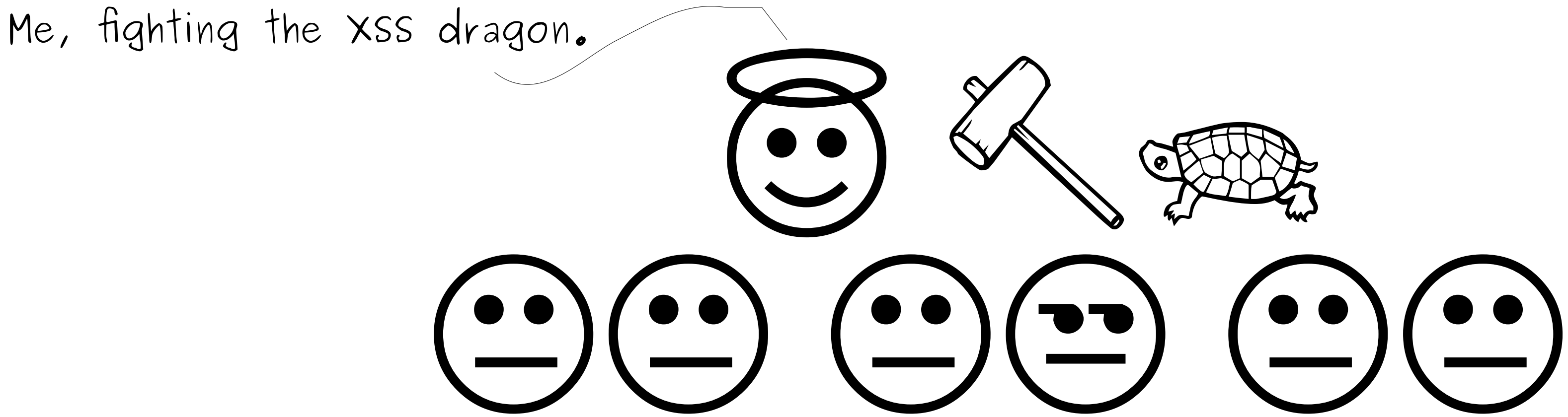
😍😍 😇😒 😐😍

There's several 100M users
protected by our library by now

So. Is our problem already solved?



Am I the savior of souls, the bringer of security, the knight in shining armor, slaying the XSS dragon?



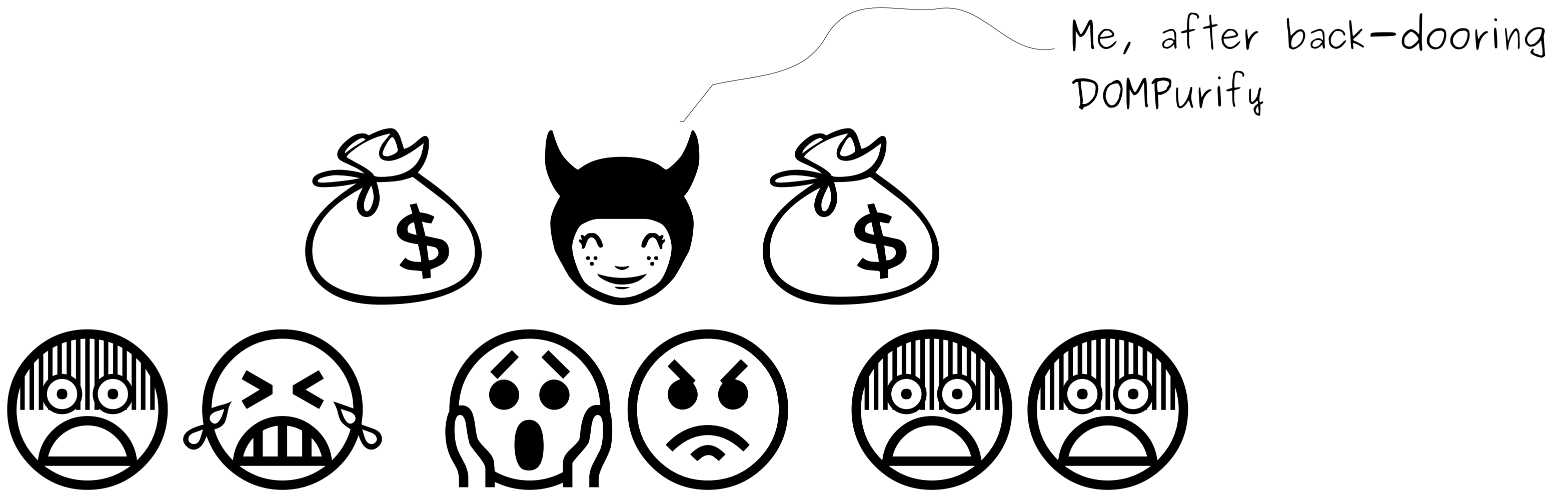
Hell no!

Because now we have a well working library. But that gives us a trust problem. **You** have to **trust** me!

😲😲 😲😲 😱😲

What if I turn rogue?

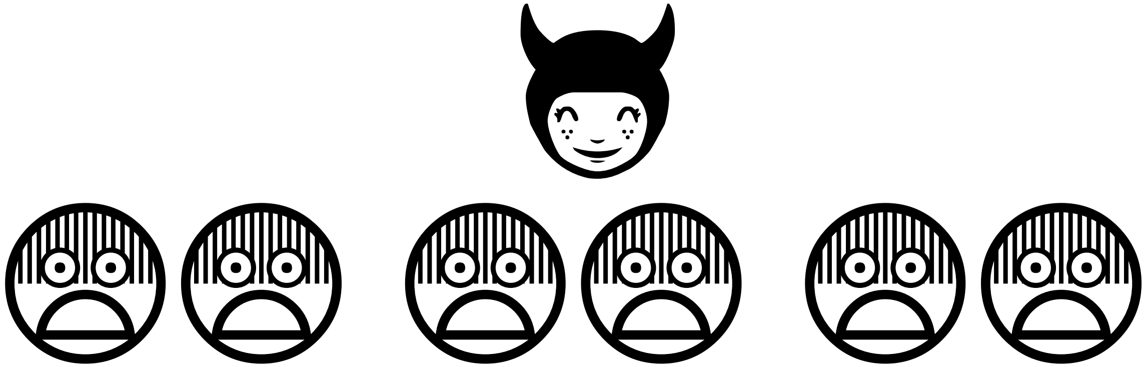
And release an update with a back-door?



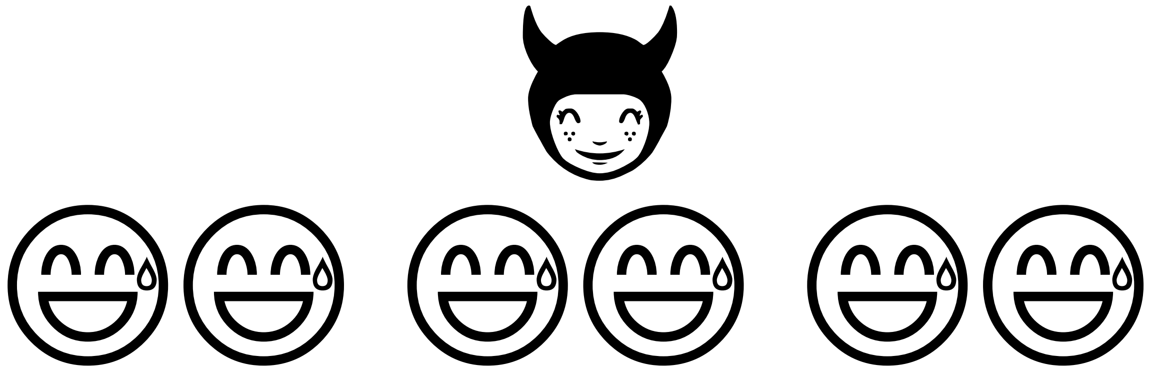
I have been writing XSS exploits for the last ten years.

I think I could hide a back-door in DOMPurify.

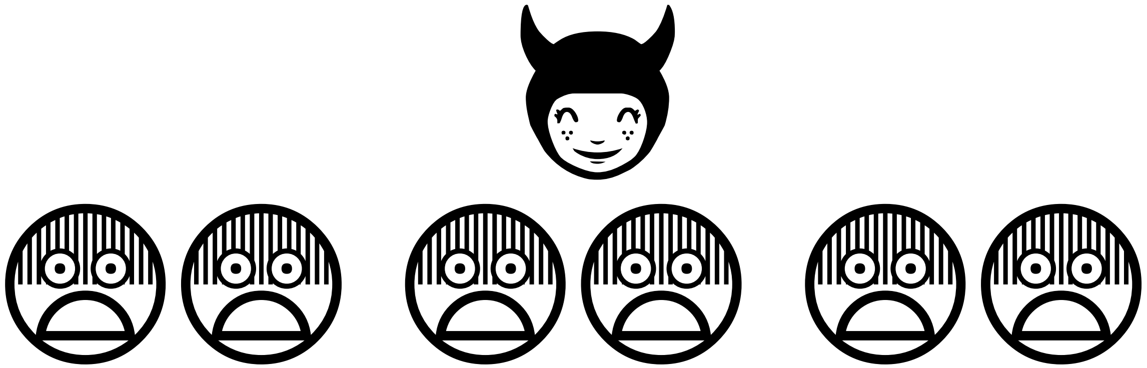
Maybe in the compressed version we ship.



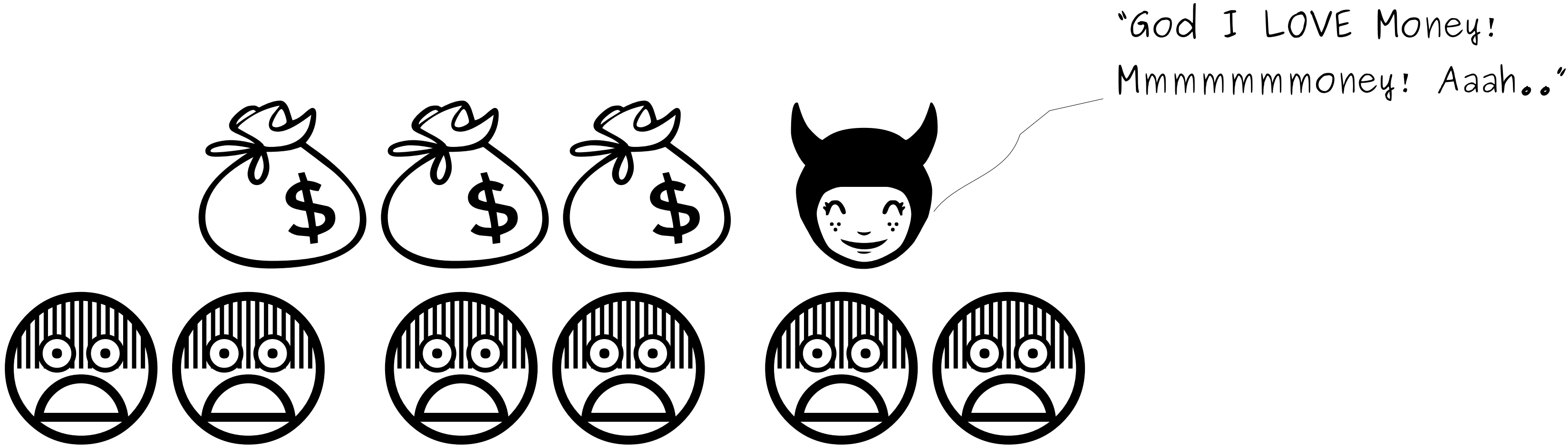
Of course I would never do that. Right?



Riiight?



Or would I?



And now what.

What options do we have? The problem now is that we have a solution to our former problem…

Now, the security of millions of users depends on a small circle of library authors and hopefully a bunch of cautious pairs of eyes noticing malicious changes.

👷 🔓 👮

We need a specification and implement DOMPurify inside browsers!

As part of the almighty DOM and its countless APIs.

We need many eyes on that, not just a few. We need a written and published description of risks models and matching solutions. We need templates that developers can use with great ease and small foot-gun potential.

🙌 💡 🙌

We need to specify what DOMPurify does and implement it
Inside the browsers' core.

Despite CSP. Because of CSP.

Because we need to fill all gaps, not just accidental
intersections.

👷 💡 👮

Now, you might point at me with your finger and ask
- why don't **you** write the specification then?

And I will respond…

😐😐 😐😠 😐😐

Why don't **YOU** write the specification!
We already did the hard work, identified the problem, proposed solutions…

…and showed, that even our own approach is fundamentally flawed because of the inherently required trust in us, the maintainers.

😲😲 😲😲 😲😲

Now, it's not just *my* but **OUR** turn.

If you agree on the problem, the proposed solutions and share our views about what needs to be done: Be the one who starts it.

We're happy to help!
But we're not gonna do it alone.

😲😃 😉😋 😉😍

Let's make all the “ElectronMails” and other tools much safer and implement XSS protection where it belongs.

Not into the server, not into WAFs and IPS.
But into the browser.

Right, where the action happens.

😃😃 😃😃 😉😃

And that concludes my presentation.

Thank you for your time :)

😍😍 😘😍 😍😍