ROPEMAKER: Manipulating Email Content Post-Delivery [DRAFT]

Francisco Ribeiro <francisco@ironik.org>

Abstract

When we think about Email as a metaphor for physical mail and how it is used on a daily basis, a message sent effectively becomes a *document* in the possession of the recipient. These messages can be used for various purposes such as *e-commerce*, both internal and external company communications, auditing and can even be used as evidence in a court of law. That said, it is important to realise that this is only possible on the premise that an email sent is now on someone else's mailbox and therefore it can no longer be modified by the sender. This is not just an important security property for secure communications but it is also fundamental to Email as a concept and to users expectations. Unfortunately, this assumption can be broken; the perceived content of a message may be modifiable after it was delivered regarless of the adoption of technologies to digitally sign emails such as OpenPGP and S/MIME. Consequently, the Integrity and *non repudiation* in Email is jeopardised for popular email clients and, as we will see, an attacker might even be able to perform such an attack with plausible deniability. Finally, we will see how these techniques may also be effective at bypassing most anti-spam/malware solutions, particularly their URL detection mechanisms.

Contents

1 Technique

It is well known that loading external images within a HTML email is an issue for Privacy reasons that Microsoft Outlook, in its desktop version may even warn us about. Now, one may be surprised to know that these clients are also able to load externally hosted CSS files. That is where things start to go wrong from a Security perspective. Whilst CSS is only meant to define the style of our HTML email, it is possible to abuse it to remotely manipulate the perceived textual content of a message even after delivery (let alone the Privacy issue). Further ahead we will also see that this is not just about CSS.

1.1 Switch: Flipping Between Text versions

The first technique and probably the easiest to explain works by sending an email with multiple versions of the text where each of these is contained in its own HTML tag. The email is linked onto a remote CSS in a server controlled by the attacker and with that, it then becomes possible to hide previously shown content from the victim and display another of version of his/her choice at any point in time. This is how the HTML and respective CSS would look like:

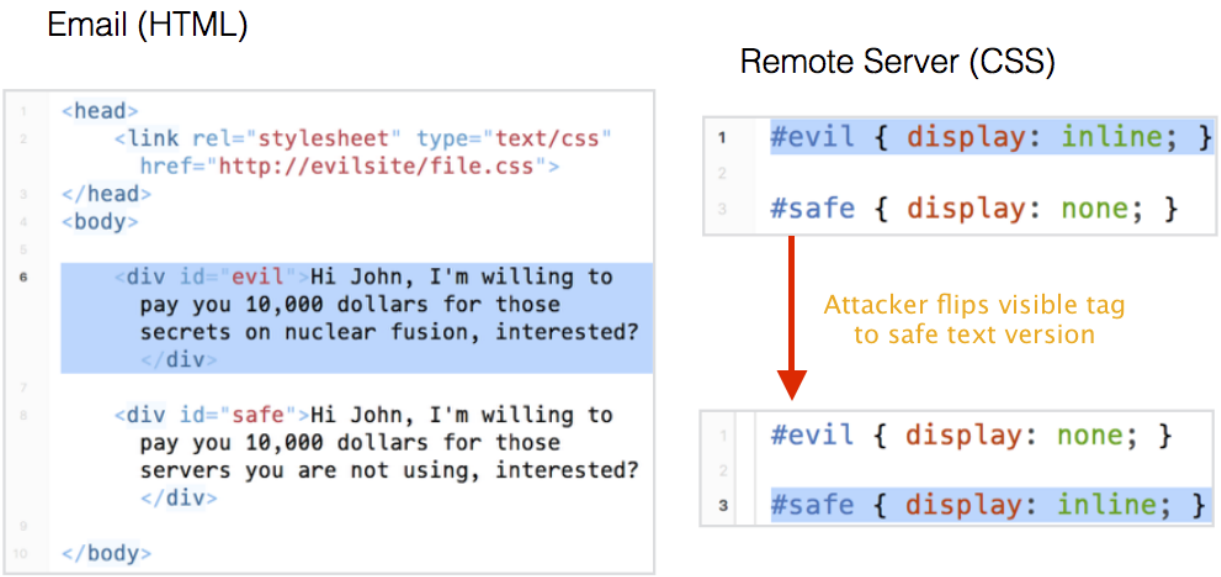


Figure 1: Switch: source code

By flipping the visible tags so that the *good* version becomes visible and *bad* becomes hidden, one could make it so that the email displayed becomes as follows:

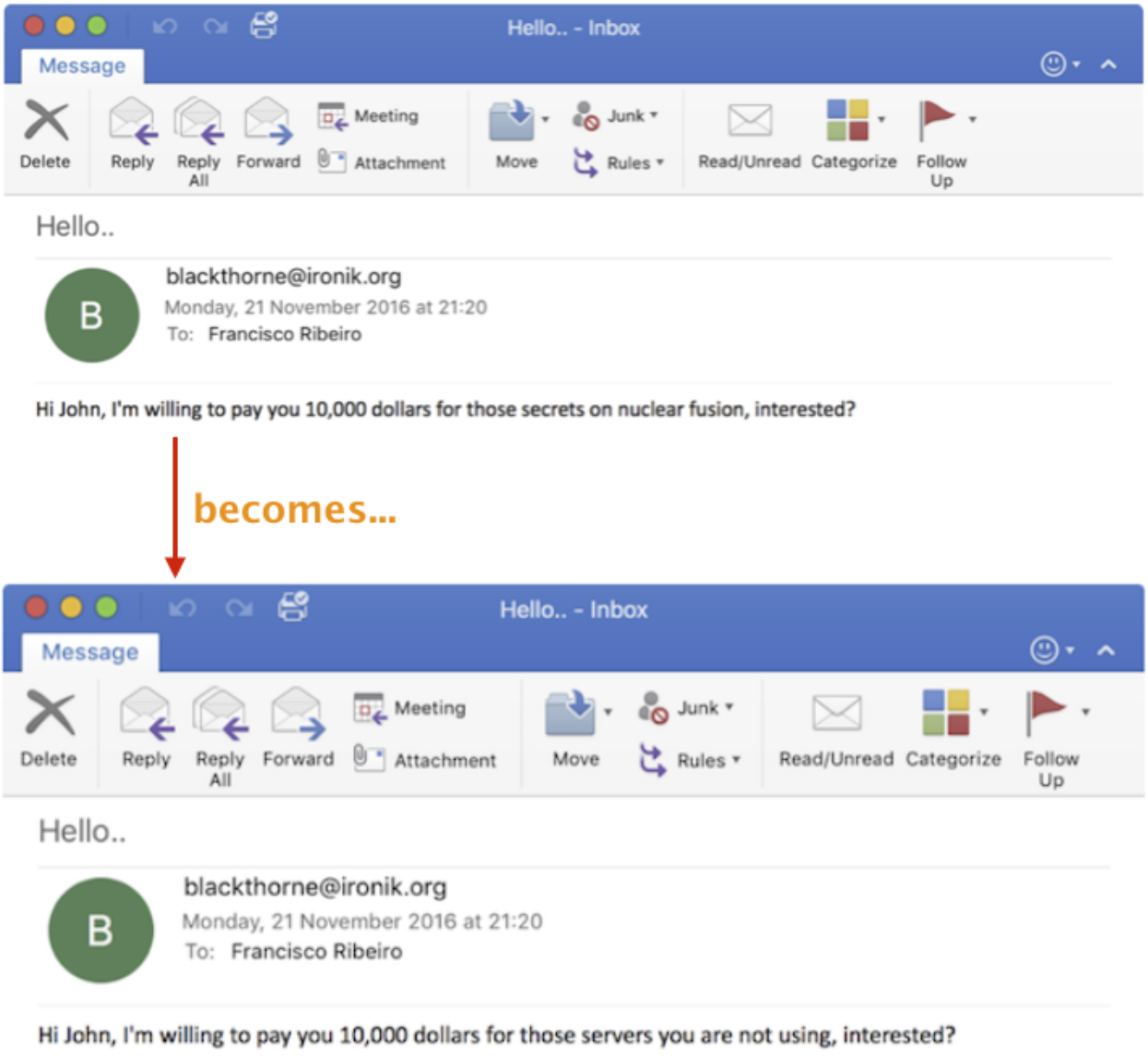


Figure 2: Switch: email rendered

Now, this is still quite constrained from an offensive perspective since the attacker needs to know in advance the different text versions to be sent and does not have full control over the message displayed. Also, it can be trivially detected by reading the source code of the email message and if the victim happens to remember a word contained in the previous version, a search for such term in the email client may reveal it (even if then it is not displayed on the email itself) which would be quite bizarre and suspicious. This technique is also not good at bypassing URL scanning filters because any anti-malware solution will see the malicious URL, regardless of its perceived visibility, as it is part of the sent message. Nevertheless, this is enough to demonstrate the problem as a proof of concept.

1.2 Matrix: Arbitrary Text Manipulation

Let us take the *Switch* approach on to its logical extreme and place all possible characters for all positions (each one contained in its own HTML tag just like before). With that, an attacker can toggle the visibility of individual characters for specific positions in the text and "write" any desired message. We can think of it like a sequence of arrays with all possible characters repeated for all possible positions in the text. The number of all those alphanumeric sequences should be repeated for as many characters as one may want to use (although we can easily think of ways to optimise it).

To univocally manipulate all tags, a unique ID shall be assigned using a simple convention:

Character	Tag convention
lowercase characters	`#<lowercase char><position>`
uppercase characters	`#up<uppercase char><position>`
numbers	`#nb<digit>_<position>`
commas	`#cm<position>`
question marks	`#qm<position>`
whitespace	`#ws<position>`
quotes	`#qt<position>`
dots	`#dt<position>`
colon	`#cl<position>`
slash	`#sl<position>`

A sample of this code can be seen on figure 3:

Email (HTML)

```
<div id="a0">a</div><div id="b0">b</div><div id="c0">c</div><div id="d0">d</
  div><div id="e0">e</div><div id="f0">f</div><div id="g0">g</div><div
  id="h0">h</div><div id="i0">i</div><div id="j0">j</div><div id="k0">k</
  div><div id="l0">l</div><div id="m0">m</div><div id="n0">n</div><div
  id="o0">o</div><div id="p0">p</div><div id="q0">q</div><div id="r0">r</
  div><div id="s0">s</div><div id="t0">t</div><div id="u0">u</div><div
  id="v0">v</div><div id="w0">w</div><div id="x0">x</div><div id="y0">y</
  div><div id="z0">z</div><div id="upA0">A</div><div id="upB0">B</div><div
  id="upC0">C</div><div id="upD0">D</div><div id="upE0">E</div><div
  id="upF0">F</div><div id="upG0">G</div><div id="upH0">H</div><div
  id="upI0">I</div><div id="upJ0">J</div><div id="upK0">K</div><div
  id="upL0">L</div><div id="upM0">M</div><div id="upN0">N</div><div
  id="upO0">O</div><div id="upP0">P</div><div id="upQ0">Q</div><div
  id="upR0">R</div><div id="upS0">S</div><div id="upT0">T</div><div
  id="upU0">U</div><div id="upV0">V</div><div id="upW0">W</div><div
  id="upX0">X</div><div id="upY0">Y</div><div id="upZ0">Z</div><div
  id="nb0_0">0</div><div id="nb1_0">1</div><div id="nb2_0">2</div><div
  id="nb3_0">3</div><div id="nb4_0">4</div><div id="nb5_0">5</div><div
  id="nb6_0">6</div><div id="nb7_0">7</div><div id="nb8_0">8</div><div
  id="nb9_0">9</div><div id="dt0">.</div><div id="cm0">,</div><div
  id="ws0">&amp;nbsp;</div><div id="qt0">'</div><div id="qm0">?</div><div
  id="sl0">&amp;#47;</div><div id="cl0">:</div><div id="a1">a</div><div id="b1">b</
  div><div id="c1">c</div><div id="d1">d</div><div id="e1">e</div><div
```

▪▪▪

Remote Server (CSS)

```
div {
    display: none;
}

#h0, #e1, #l2, #l3, #o4 {
    display:inline;
}
```

→

```
div {
    display: none;
}

#b0, #y1, #e2 {
    display: inline;
}
```

Figure 3: Matrix - source code

and with this, an email originally saying “hello” would be turned into a “bye”.

This technique is far more powerful since it can be used to generate arbitrary text while content is not prone to show up in any search query as before and past messages can no longer be perceived just by looking at the HTML code.

Even without an anchor tag, if we use this technique to write a sequence of characters that matches a URL pattern, Apple Mail will make this a 1st class link *i.e.* highlighted and clickable. This is interesting because it can be used to send and modify text and links post delivery in a way that can not be perceived by most anti-spam/malware tools.

Spammers who are not particularly interested in repudiation and do not really need to modify things post delivery, could simply apply the same *Matrix* logic within an embedded `<style>`, so they would not need to link the email with a remote CSS (that can actually raise alerts due to a sudden a spike in outgoing requests if these emails are sent to many) and also do not risk triggering any warning banner from the email client.

However, one downside of this technique is the size of the emails generated which can be a limiting factor if we try to generate long messages.

1.2.1 Side-effects

We are not quite done yet though, there is some makeup work that needs to be done to make these attacks more credible due to some side effects that were not yet discussed. One of the things that might look suspicious is the text preview displayed within the email listing, as follows:

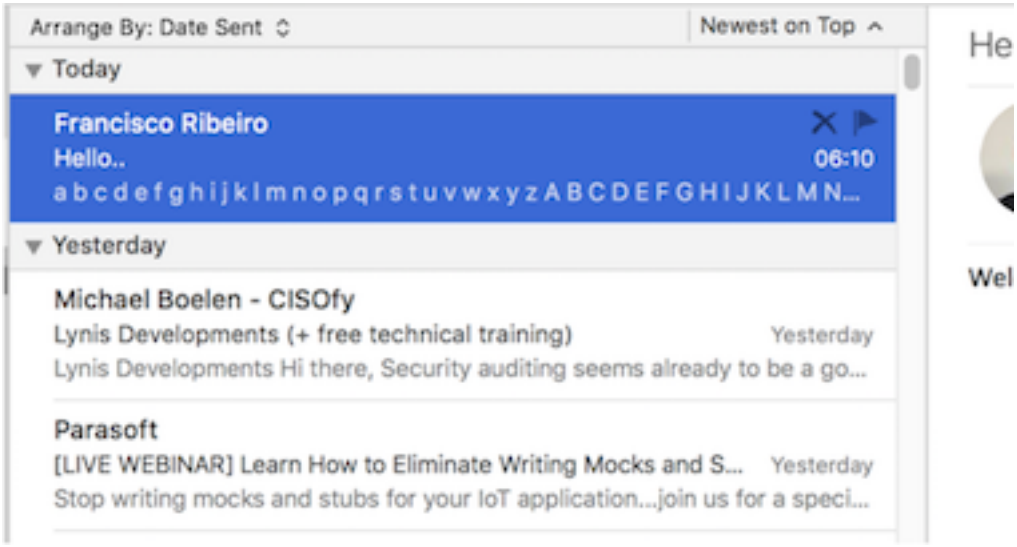


Figure 4: Matrix - message preview

Given those contents have not yet been rendered by CSS, email clients will display them right there. Adding a bunch of white spaces will not help because these will be stripped out. One way to get around this is by adding an HTML comment or a hidden tag with text right at the top of the email that could contain a common prefix for both *good* and *bad* versions of the email or some evasive / meaningless text, as follows:

Figure 5: Fake preview message in Microsoft Outlook

An alternative approach for Apple Mail is to emulate the loading effect that we temporarly see on a newly delivered email. As said, white spaces are stripped but we can do so by adding the text “Loading” followed

by the unicode <U+2029> (unicode character confusable with whitespace) repeated multiple times to get something like this:

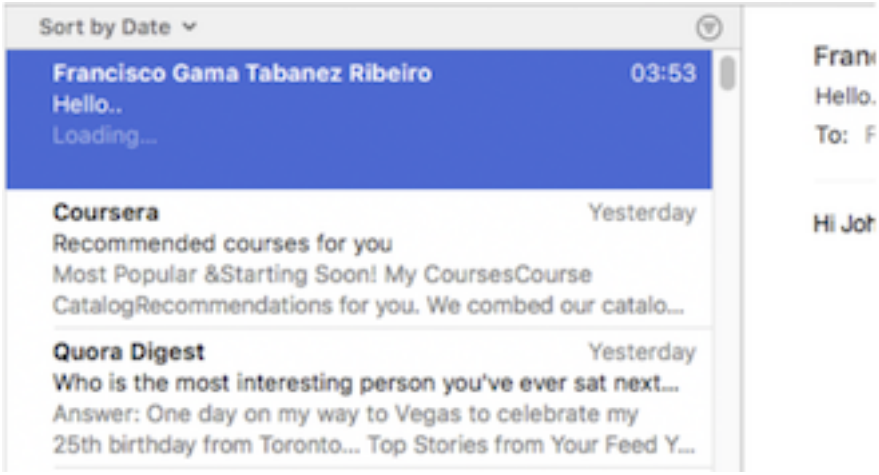


Figure 6: Fake preview message in Apple Mail

This approach is not very reliable as it does not work with some versions of Apple Mail and never on Outlook.

Another side effect has to do the way Outlook interprets HTML since it breaks sequential HTML tags onto different lines. This affects its representation so that the email to be rendered will have whitespaces between each character, as we can see here:

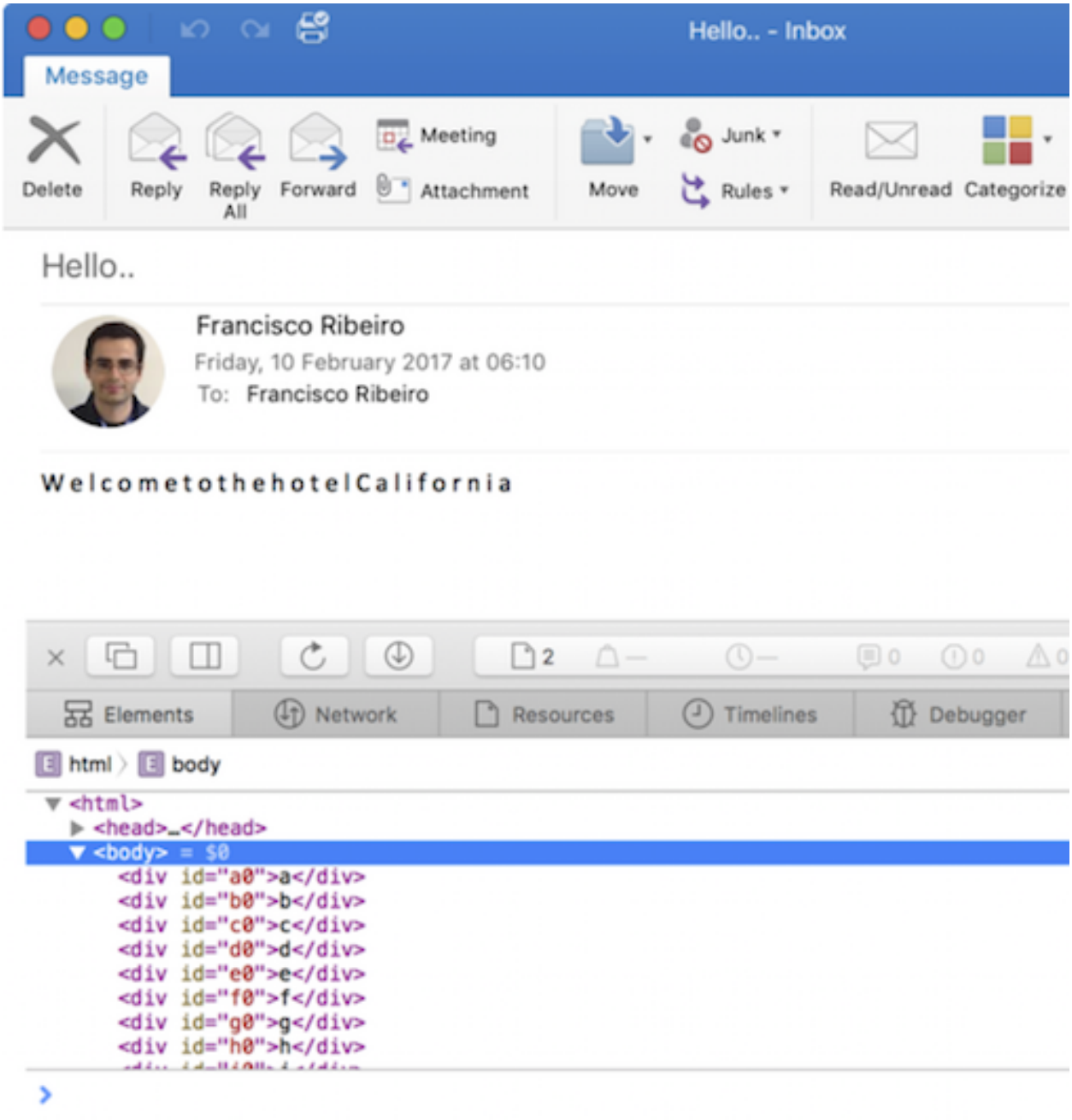


Figure 7: Broken textual representation in Microsoft Outlook

One way to get around this is by setting the *font-size* to 0px as default which shall be applied to everything (including of course the white-spaces that we want to hide) and overriding it with 14px (the default) for the characters that we want to be seen:

```
body {
  font-size: 0px;
}

div {
  font-size: 14px;
  margin: 0px;
  padding: 0px;
  display: none;
}

/* message */
#upW0, #e1, #l2, #c3, #o4, #m5, #e6, #ws7, #t8, #o9,
  #ws10, #t11, #h12, #e13, #ws14, #h15, #o16, #t17,
  #e18, #l19, #ws20, #upC21, #a22, #l23, #i24, #f25,
  #o26, #r27, #n28, #i29, #a30 { display:inline; }
```

Figure 8: CSS code to fix message in Outlook

and now our email looks sane:

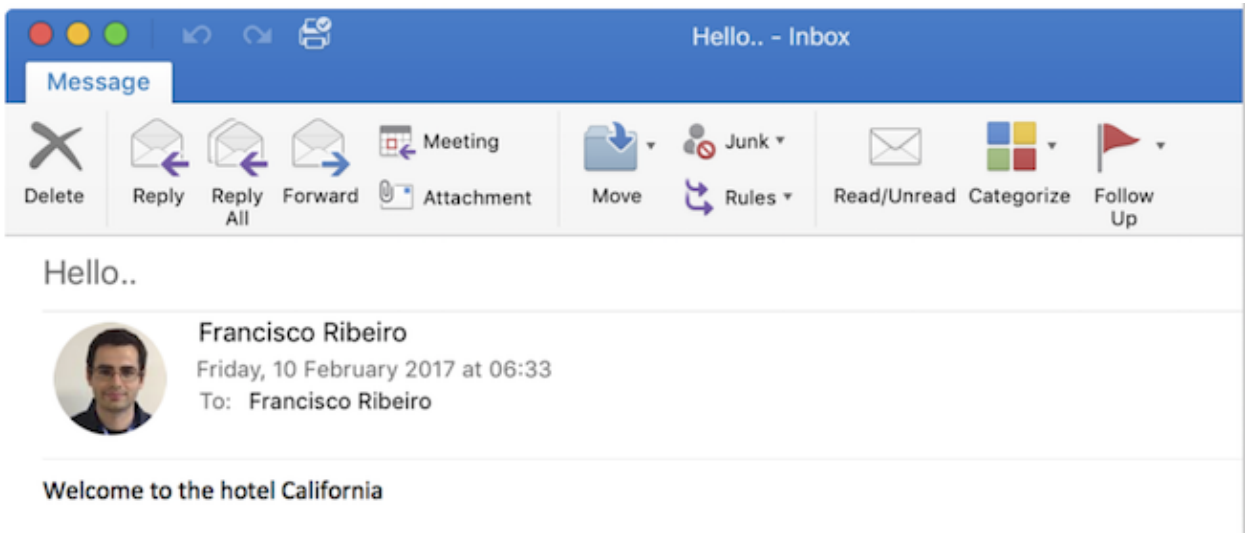


Figure 9: Fixed message display in Microsoft Outlook using the Matrix technique

1.2.1.1 When the victim is Offline

When using the *Matrix* technique, another suspicious side effect takes place if the victim opens the email while offline and sees a bizarre long list of meaningless characters.

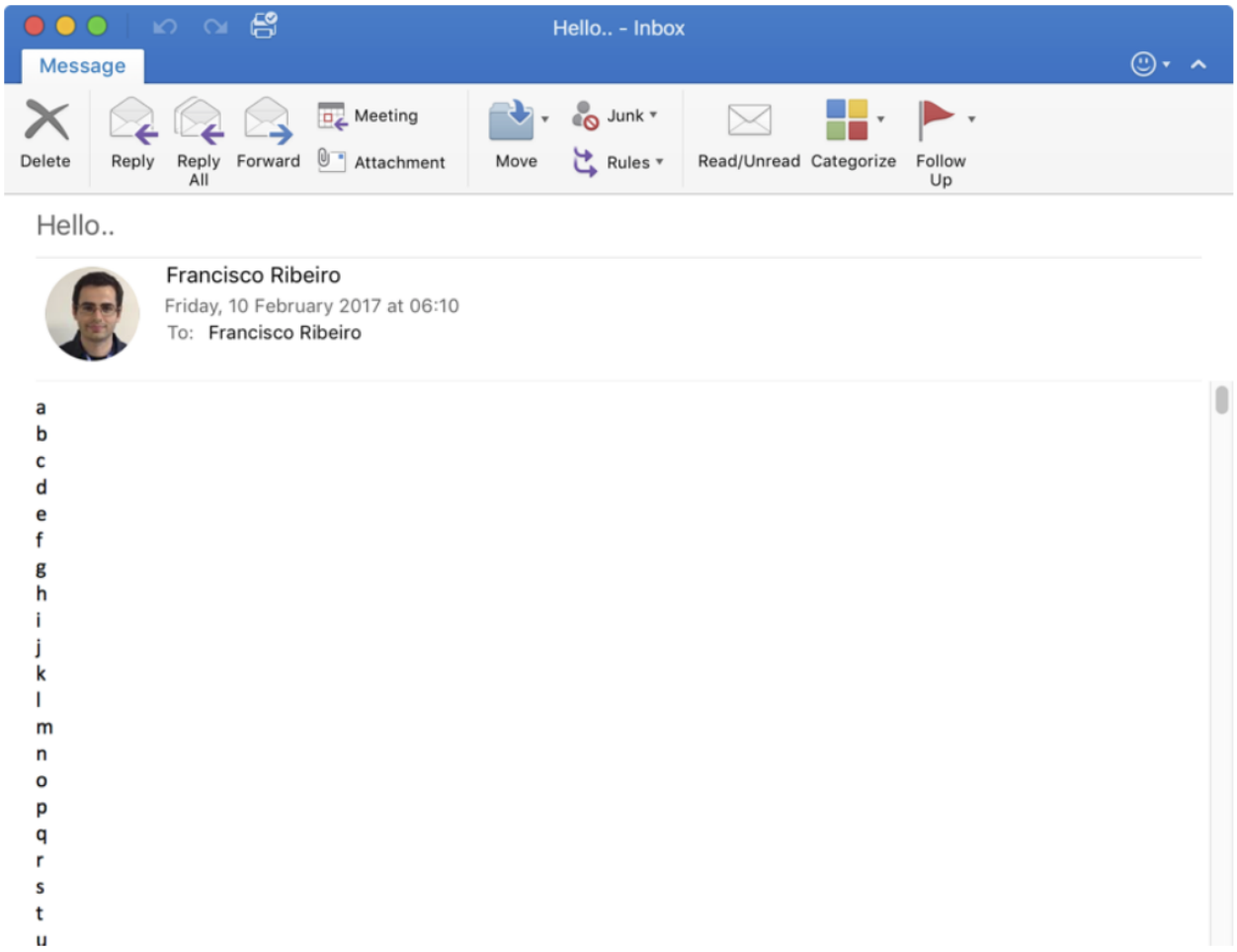


Figure 10: Matrix side-effect when victim is offline

We can get around it by adding a *safe* version of our text within a tag which is only visible until the remote CSS loads and overrides its display property to hide it. Also, we may want to include a tall empty section before the matrix itself which shall push it under the visible area of the Email. If the remote CSS fails to load either because the email client is not vulnerable or the victim is offline, then the user will see the *safe* version as a fallback.

With the makeup techniques described above one could already create credible messages that look just like common emails and without actually carrying the text message in it. However, this is not to say they would provide plausible deniability that could stand against any forensic investigation which would easily outline the underlying trickery.

1.3 Content Property: Towards Plausible Deniability

In CSS, the *content property* can also be used to define text which can then be added (and modified) before or after any given HTML tag by using the *::before* or *::after* selectors, respectively. Note that in this technique, the CSS to be applied sets no requirements in the original HTML email (other than being linked against it), as follows:

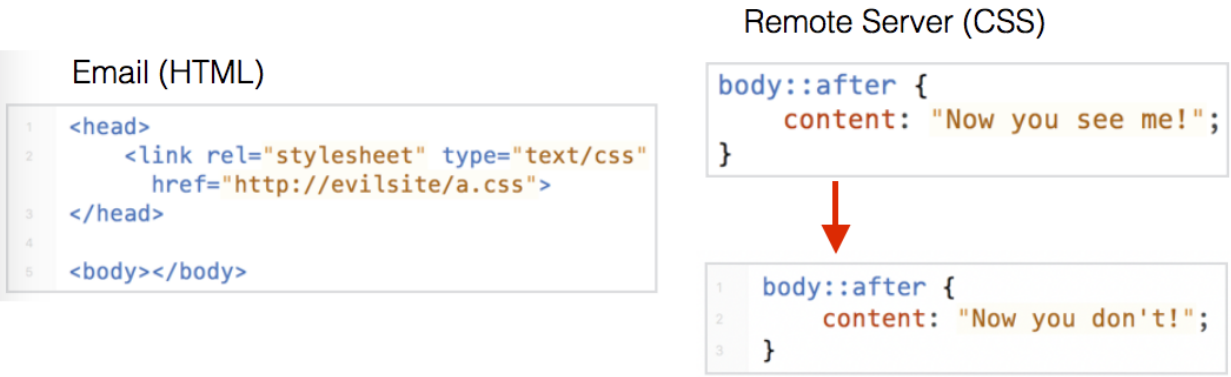


Figure 11: Content Property with ::after selector

This technique is quite elegant since it does not have any predefined character limit and does not require one to send fat emails.

Whilst here, URL patterns sent are not clickable as we saw in the *Matrix* approach, this technique has fewer constraints and such patterns would not be picked up by anti-spam tools in general since most of these do not retrieve remotely hosted contents, let alone parsing CSS. In fact, the email above has a completely empty body which makes it easy to place something else in there to be displayed in case the victim is offline or uses an email client that is not vulnerable.

This is also interesting for plausible deniability. All one needs is a plausible justification to have a remote CSS linked into his/her emails. This remote CSS could define the layout of an email footer that is always sent along with all emails (including those where there is no malicious behaviour) so that all messages are indifferentiable from each other in this regard. Then, that malicious actor could just send a common email with a *safe* version of the text which would be hidden once the CSS kicks in and a new textual content would be set through the content property just as explained above. If the malicious actor then updates the CSS to the known clean version, the email used for malicious purposes would look just as any other even from a forensics perspective.

1.4 Alphamix: Leaving CSS behind

So far we have discussed techniques that depend on remote CSS files which may lead one to believe that blocking them is sufficient. This is not the case as one can set custom fonts that can also be externally hosted and loaded into an email. One way to think about a font is to think of a mapping between the ASCII / Unicode bytes and the glyphs it can represent (fonts can be quite more complex than that but let us keep it simple for this exercise). Now, let us abandon that idea and think of a dynamically generated remote font where non-required characters are used as slots that point to the desired character representation for any given position. In this case, we will explore this with a TrueType font[1]:

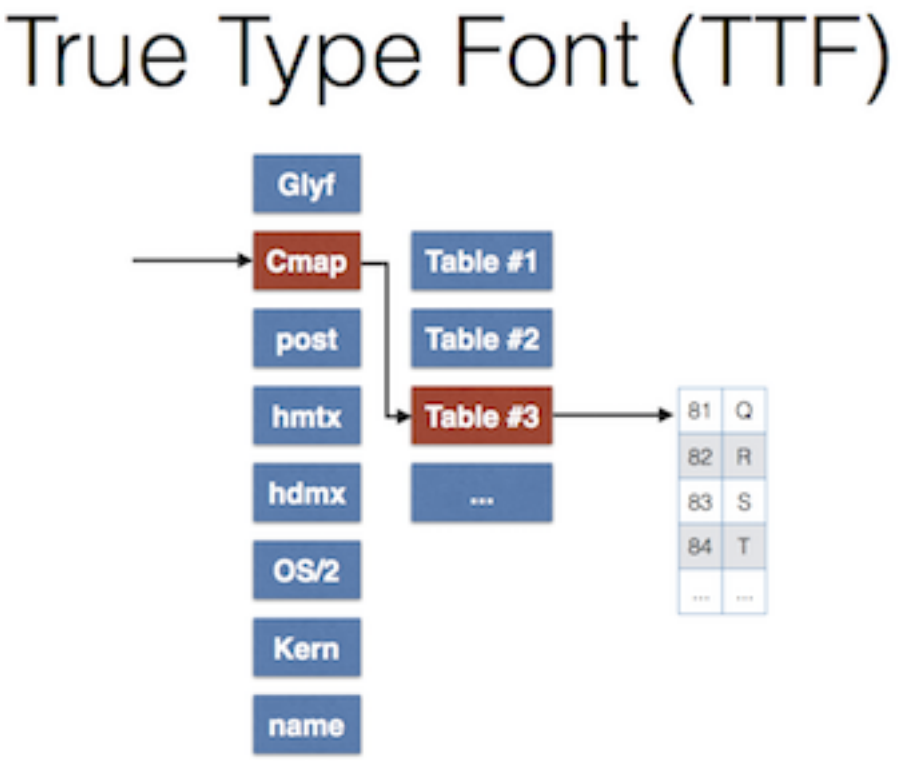


Figure 12: TrueType Font - manipulated structure

Basically, every time the email is sent with a sequence of non-required characters that should be different from each other (as they represent different positions in the text), an external font will then wire each byte with the representation of the desired character for that given position, as follows:

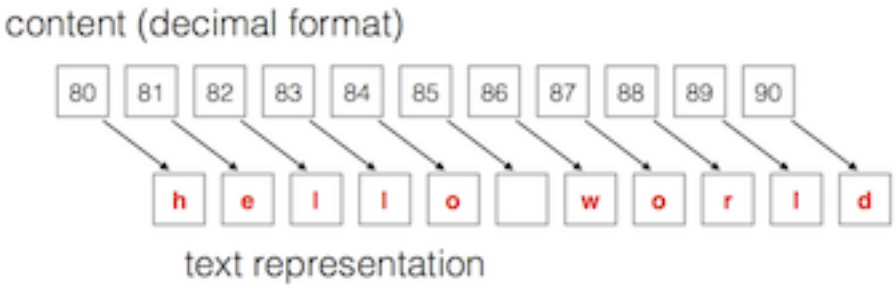


Figure 13: Alphamix

This is also a really nice trick for clipboard attacks because now what one (apparently) copies is not necessarily what he or she pastes and it can also bring text that is wired into non printable characters and therefore hidden. In my next example, we can see a proof of concept where the current time that gets updated every time a user opens the email. If we were to copy that timestamp we would get something completely different once pasted, as can be seen in the source code shown:

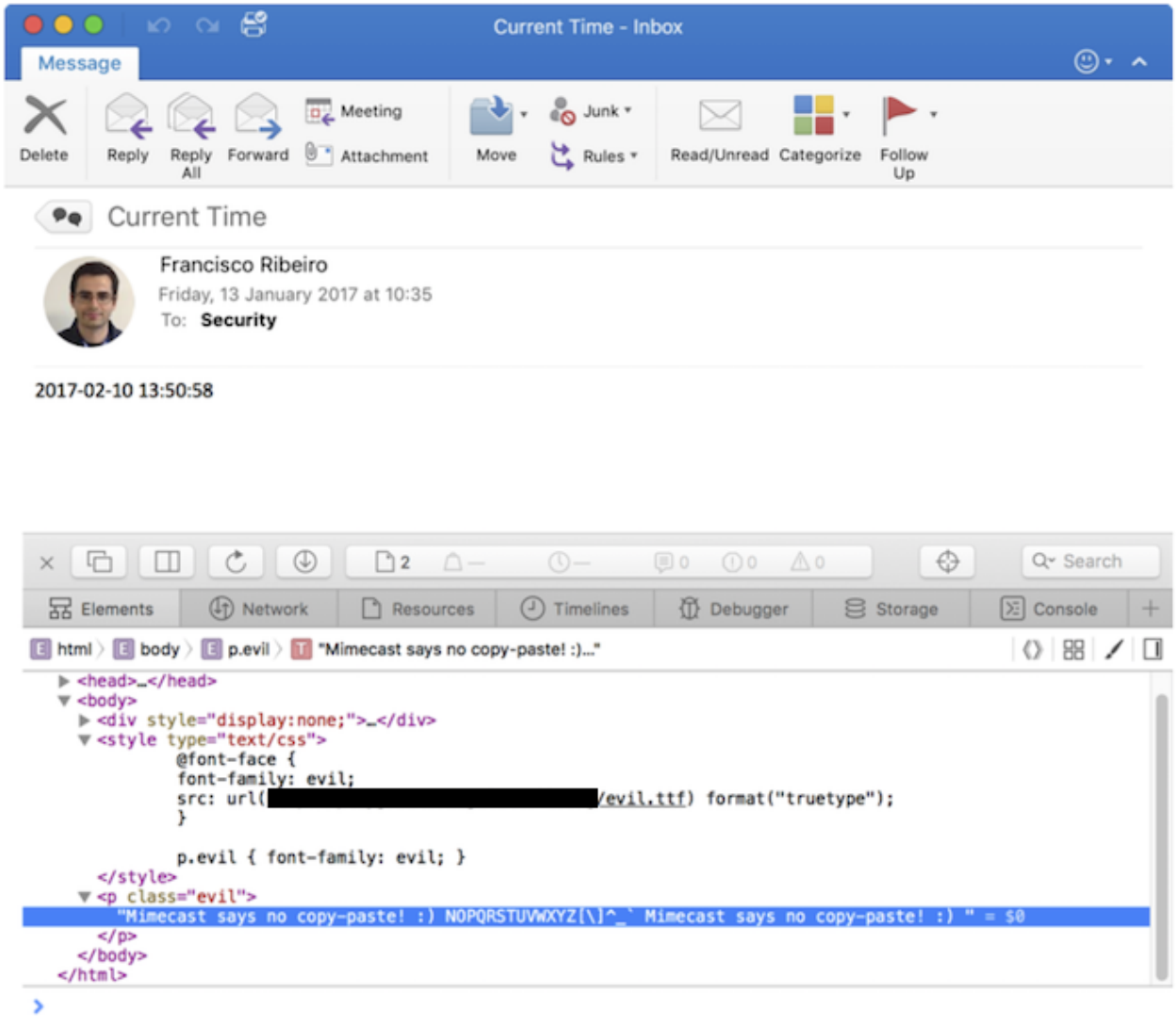


Figure 14: What we copy is not what we paste

If we highlight and copy the timestamp we would get:

`Mimecast says no copy-paste! :) NPQRSTUVWYZ[\\]^_` Mimecast says no copy-paste! :)‘

when pasting. Note that this content actually has a greater length than the timestamp “copied” and that is because we are also wiring several visible characters into the carriage return line feed ASCII code so they are not visible.

1.5 Other Vectors

There are several other vectors to exploit this. For example with CSS one could also have achieved this by playing with text positioning and vertical overlapping (`z-index`) - overriding characters in the right order to define a desired message.

Other approaches of this attack are possible without using CSS such as SVG (which can also carry text and links) and other HTML tags such as `<embed>`, `<iframe>`, `<object>`, `<element>` (only works on Apple Mail).

2 Other Threats

2.1 Man-in-the-Middle attacks

When we consider the general assumption that CSS is just used for style and should not carry any sensitive data in it, it is not that unusual to see applications with these resources linked over HTTP. From an offensive standpoint here we do not get to set the content of the original email which now is sent by the legitimate user but if we do get to control CSS we can still manipulate it.

First we need to wipe the existing text to then include our own. This can be trivially done using concepts already described such as hiding a pre-existing tag (such as `<body>`) or setting a font with 0px size against its text. Then we add some text after the body using the content property as shown before. This would work regardless of the use of OpenPGP or S/MIME to digitally sign emails.

2.1.1 Breaking Confidentiality

It is also not completely inconceivable the idea of using this vector to actually read the content of the email through the intercepted CSS by exploring ideas such as Mario Heiderich's *et al* on "Scriptless attacks - Stealing the Pie Without Touching the Sill"[2].

2.1.2 Server Compromise

It is frequent for organisations to set predefined signatures to be included in all emails across a e.g. company. If an email administrator uses a remote server (maybe because he or she dragged and dropped some fancy logo from the web by reference) the Integrity of all these emails now depends on the trustworthiness of that IP / hostname so these threats may not necessarily come from the sender.

2.2 Other things one can try

There are also other attacks one can try, such as sending an Email to a group to be perceived differently per reader leveraging differential responses against IP, request time, user agent that could hint on who is who. We can also try to send an email to a thread group and modify the perceived content of other participant's replies since the quoted responses are just HTML. Security solutions that do not work in *Proxy* mode and request these resources to the malicious server can often be fooled since they may end up analysing content that is not necessarily the same as the one delivered to the victim.

3 Digital Signatures (OpenPGP, S/MIME)

To better protect Email, different encryption standards were created throughout time namely OpenPGP (Pretty Good Privacy) and S/MIME (Secure/Multipurpose Internet Mail Extensions). These two standards may differ in things like their key distribution model but both can be used for data encryption and digital signatures by leveraging public key cryptography.

From chapter 1 of the document *Introduction to Cryptography* in the PGP 6.5.1[3] documentation:

> Digital signatures enable the recipient of information to verify the authenticity of the information's origin, and also verify that the information is intact. Thus, public key digital signatures provide authentication and data integrity. A digital signature also provides non-repudiation, which means that it prevents the sender from claiming that he or she did not actually send the information. These features are every bit as fundamental to cryptography as privacy, if not more.

That said, these digital signatures may cover the different parts of the actual email (and file attachments, optionally) but not remote resources that may be linked to it. Therefore, in spite of the increased assurance provided by these standards, the ROPEMAKER techniques described earlier will work regarless of the use of OpenPGP or S/MIME.

The following example illustrates this. We can see a signed email holding two different different messages at two different points in time yet always presenting a valid signature.

HTML email sent:

```
<html>
  <head>
    <link rel="stylesheet" type="text/css"
      href="http://evilsite/file.css">
    <meta http-equiv="content-type" content="text/html; charset=utf=-8">
  </head>
  <body>
    We will review your salary when the time is right.
  </body>
</html>
```

Remote CSS:

```
body { font-size: 0px; }

body::after {
   font-family: Helvetica;
   font-size: 12px;
   content: "Hey, keep working hard and I shall triple your salary by the end of the month!";
}
```

Once the malicious actor removes the CSS file (or if the victim goes offline), the content displayed will fallback to the version in the original email.

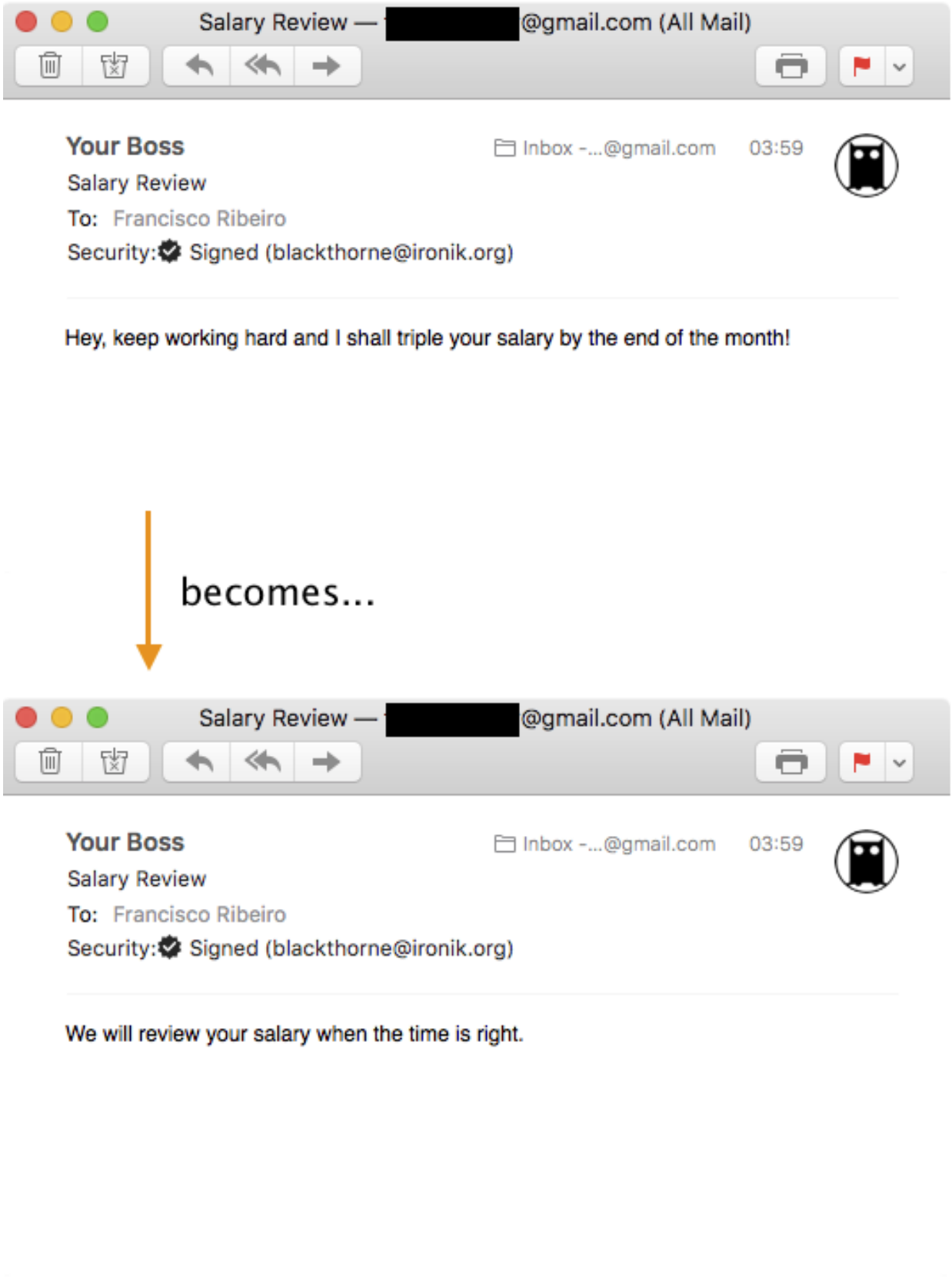


Figure 15: Different messages displayed for the same OpenPGP signed email

4 Software Affected

	Microsoft Outlook Desktop (Mac)	Microsoft Outlook Mobile	Apple Mail Desktop	Apple Mail Mobile	Thunderbird	Android
Switch	X	X	X	X	√	√
Matrix	X	X	X	X	√	√
Content property	X	X	X	X	X	√
Font Alphamix	X	X	X	X	√	√
SVG	X	X	X	X	X	√
embed / frame	√	√	X	X	√	√

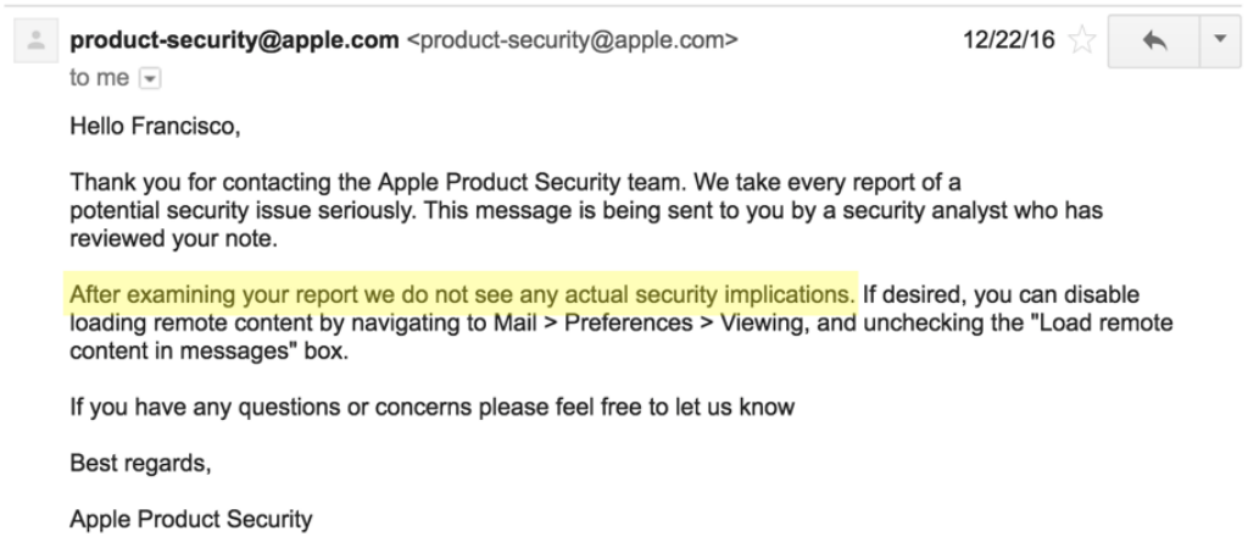
Figure 16: Software tested - X means vulnerable, V means safe

Mozilla Thunderbird may be exploitable in more techniques but my results were inconclusive so this table may not be accurate. Also, testing in Android was done against a Samsung device so I am not certain about its behavior for other flavours.

Finally, most web clients tested were found to be safe including Microsoft Outlook Web Access (OWA), Outlook.com, iCloud, Gmail, Yandex, Yahoo web Mail. That is not to say that if we have an account in e.g. Yandex service and we pull our email messages with Apple Mail that we will be protected, most likely not.

5 Responsible Disclosure

This issue was reported to both Apple and Microsoft in early November, 2016 and we can see their replies below:



product-security@apple.com <product-security@apple.com> 12/22/16

to me

Hello Francisco,

Thank you for contacting the Apple Product Security team. We take every report of a potential security issue seriously. This message is being sent to you by a security analyst who has reviewed your note.

After examining your report we do not see any actual security implications. If desired, you can disable loading remote content by navigating to Mail > Preferences > Viewing, and unchecking the "Load remote content in messages" box.

If you have any questions or concerns please feel free to let us know

Best regards,

Apple Product Security

Figure 17: Apple's response

This was later brought up to a wider group of organisations at the 39^{th} edition of the M^3AAWG[4] event in San Francisco, US. Since then, selected organizations where also reached on this matter which expressed serious concern including but not only national CERTs, national security departments, banks and email security vendors. The Rspamd anti-spam solution has included some level of detection against these patterns.

Finally, a CVE was also requested and rejected. It is worth noting that the CVE assignment process is directly handled by both Microsoft and Apple who are responsible to issue CVEs on their own products as CVE Numbering Authorities (CNA[5]).

Figure 18: Microsoft's response

6 Recommendations

I consider the ideal solution to be for email clients to stop loading external resources. This does not imply that they can not use HTML nor CSS but that these should be self-contained without loading functionality from an untrusted control sphere.

Exceptions to this should only be granted upon user approval and risk acceptance. Note that the current warning banner presented in Microsoft Outlook in its desktop version does not qualify to this because it refers to a different risk (Privacy) applied to a different type of resources (Images) so users have been "accepting" a different risk.

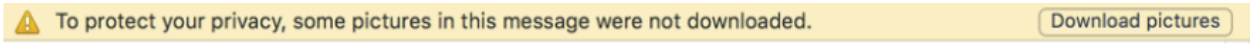


Figure 19: Warning banner displayed in Microsoft Outlook desktop version

Moreover, delegating security decisions to users (and on something that they have not received yet to be able to analyse) is highly prone to go wrong so exceptions of any kind are suboptimal if it is a real solution that we are aiming to achieve.

For now, there are a few workarounds that can be applied. For Outlook, in its desktop version we can make sure that we do not accept any remote content by never accepting the warning banner and we can take similar action on Mozilla Thunderbird. On Apple Mail for Desktop there is a setting (not enabled by default) to disable this behavior as we see on Apple's response message above.

Finally, for Apple Mail in its mobile version as well Microsoft Outlook for iPhone, I am not aware of any control to mitigate this. We can however use different clients that may not be susceptible to this. There are safe native clients such as the one from Mimecast (although this will only work with their email service) as well as web clients (if available for our email service) which in general are safe against this. It is worth noting that corporate environments often maintain their own email servers and predominantly use native clients like the ones affected by this vulnerability so most of these web clients may not be an option as they are often tied with their own service (Microsoft's OWA is safe though).

7 Discussion

Email was never designed to support all the properties and features that we expect from it today. There was not a pre-conceived architecture or stack in which its technologies were pushed into throughout time. In fact, its development was quite organic and chaotic and things like styling were imported from the web without sufficient consideration with the implications of bringing something from a dynamic environment where it is normal and desirable for contents to change at any point in time. This interception is, in my view, the root cause of this and other problems.

While organisations intentionally push relevant events into Email as an audit trail, it is important to realise that loading external resources into these is more than a privacy issue. Email clients should also protect the integrity of (perceived) content[1] at all times since this is actually what users rely on *i.e.* the presentation layer.

At the time of writing this document, most Security technologies including anti spam / malware solutions will not load these remote resources, let alone interpreting CSS and fonts to understand the final outcome of a rendered email. Even when they do, they merely look after malware which is not always what this is about. Manipulating messages post delivery can also be used to break past commitments and to remove textual evidence that humans rely on and it is very hard for these tools to grasp the meaning of text and infer anything malicious about it. Breaking *non repudiation* is actually more likely to happen from contacts with whom the victim already holds some level of Trust which also makes users much more likely to accept any warning message displayed (and specially a message that only mentions Privacy concerns of remote pictures).

As we keep on bringing more features and capabilities to email (e.g.: HTML5, CSS3, BIMI[6]) these problems are likely to get worse and new ones are likely to emerge. At the same time, we still have issues like ROPEMAKER or the fact that encryption is often weak or optional so this may be a good opportunity to deal with the huge technical debt existing in this area, and rethink Email Architectural Design using proven building blocks.

8 F.A.Q.

What does ROPEMAKER acronym stand for and why is it so contrived?

ROPEMAKER stands for Remotely Originated Post-delivery Email Manipulation Attacks (obviously!). I did not come up with it but there is a reason on why it is so contrived. Ropemaker is the street name of Mimecast[7] HQ in Europe (London). Its meaning also has something to do with the concept of connecting things which is what we are doing to manipulate contents post-delivery.

Is this a vulnerability or a feature?

HTML with remote CSS support was intentionally added into Email clients and is working as expected given its implementation. However, the concept of Feature is not mutually exclusive from Vulnerability and we consider this feature applied to the Email's use case to be a dangerous addition that introduces a design flaw *i.e.* a Vulnerability[8]. As an analogy, if a banking web application features the HTTP protocol by default, can't it be vulnerable to Man-in-the-Middle attacks even in the absence of known bugs in its HTTP protocol implementation? Perhaps this confusion relates to the fact that this is not a *bug*.

Why is this a design flaw?

By loading remote resources into an email, we break a core property / expected behaviour which is that the emails once delivered are immutable and consequently we lose *non repudiation*. Introducing features always needs to be done in respect to the purpose of the application as a whole including the security properties that must be met. If a particular functionality voids them, then it introduces a *weakness*.

Is disabling remote CSS sufficient to address this problem?

[1] The actual data stored on the device may not even be reachable (e.g. non-jailbroken iPhone).

No, because as we described there are other ways to manipulate delivered email without relying on CSS.

How can we classify this vulnerability?

Inclusion of Functionality from Untrusted Control Sphere - CWE-829[9].

Is this a problem and these techniques only meaningful to Email?

No. The concepts involved actually relate more to the Web so depending on the threat model, having e.g. CSS injection could also be more than a Privacy issue.

Why are webmail clients not affected?

Even before we think about Security, the idea of loading any foreign HTML or CSS within emails is likely to interfere and to break the appearance of a Web application as a whole. Concepts like HTML or CSS injection and some of its implications are also better understood from a Security perspective in this space so it is not surprising that the techniques here presented are less likely to work on the Web. That said, we see no reason why the security requirements provided by native email clients should be any lower than their Web counterpart.

Is this a new risk?

No, the inclusion of remote CSS is a functionality dating back (at least) until 2007 so the risk was always there.

What if I save the email onto a backup file and open it from there?

If we open it using the same email client, we will be exposed to the same risks regardless.

Is this a complex exploit?

No, I'm almost embarrassed to call it "exploit". Note that the fact that something is trivial to exploit only makes it more likely to be explored *i.e.* more risky.

Is this a "Cyber Virus"?

Hell, no.

Are there any previous references to any of these techniques?

Not that we are aware of but we are keen to include it if there is any. However, we found a Security Consideration within the RFC 2318[10] which mentions the potential of CSS to hide content which already touches the problem:

> Security considerations:
>
> ```
> Applying a style sheet to a document may hide information
> otherwise visible. For example, a very small font size may be
> specified, or the display of certain document elements may be
> turned off.
> ```

Is plain text a solution here?

Usually email clients allow us to compose messages in plain text but keep in mind that the victim is the one receiving email so what we really need is a way to enforce that received messages are parsed as plain text only. We did not find such a capacity in Apple Mail for both Desktop and Mobile versions but of course we always use good old Mutt and Emacs Rmail.

Do the risks highlighted also apply to remote Images by embedding text in it?

Yes. That is actually well known but remote images carry significant differences from pure text such as the fact that text embedded in images does not adapt to the screen orientation of the device nor to default font attributes (face, size and color) and it often includes little artefacts that give it away. Another key difference is that text embedded in Images can not be highlighted / copied as pure text.

What happens if victims are using different Email clients?

Different email clients work differently and are affected by different things. For very targeted attacks (which is probably the biggest concern here) that might not make much of a difference since the attacker might know the client used by its victim. However, if the target is a larger group of people as opposed to an individual, it may happen that one uses a e.g. Apple Mail and someone else uses Microsoft Outlook or even a client that is not affected and we can not necessarily pull the same trick against all. The attacker can however explore the fact that each client identifies itself upon fetching remote files (*User-Agent* header or equivalent) and play with differential responses from his/her web server. The *content property* technique is the best suited for that purpose. Differential responses can also often be used to bypass many security solutions that give themselves away when triaging web resources, although this is out of the scope for this document.

Disclaimer

The views expressed here are mine alone and not those of my employer.

References

[1] Inc., A., "TrueType reference manual," <https://developer.apple.com/fonts/TrueType-Reference-Manual/>.

[2] al, M. H. et., "Scriptless attacks - stealing the pie without touching the sill," <https://www.nds.rub.de/media/emma/veroeffentlichungen/2012/08/16/scriptlessAttacks-ccs2012.pdf>.

[3], <https://users.ece.cmu.edu/~adrian/630-f04/PGP-intro.html>.

[4] "Messaging malware mobile - anti-abuse working group.", <https://www.m3aawg.org/>.

[5] "Common vulnerabilities and exposures.", <https://cve.mitre.org/cve/request_id.html>.

[6] al., T. L. et., "Brand indicators for message identification (bimi)," <https://authindicators.github.io/rfc-brand-indicators-for-message-identification/>.

[7] Mimecast., "Mimecast," <https://www.mimecast.com>.

[8] Wikipedia., "Vulnerability (computing)," <https://en.wikipedia.org/wiki/Vulnerability_(computing)>.

[9] "CWE-829: Inclusion of functionality from untrusted control sphere.", <https://cwe.mitre.org/data/definitions/829.html>.

[10] Lie, e. a., "The text/css media type," <https://tools.ietf.org/html/rfc2318>.