Single Sign-On Watering Hole

Four Horsemen of Apocalypse by Viktor Vasnetsov

This Presentation

- Authors: Justin Barron, Rodney Beede, Michael Hunter, Roan Moore
- Intro (1 min).
  - Who the speaker is, etc.
- Background (7 min).
  - What is SSO?
  - Anatomy of a cookie.
  - Anatomy of an enterprise network.
  - Anatomy of a security watering hole.
- The vulnerability (2 min).
- The fixes (7 min).
- Prevalence of the flaw (1 min).
- Vendor checklist (1 min).
- Questions.

What is SSO?

Photo Credit: Sherrie Thai

What is Single Sign-On? (Definitions)

- Single Sign-On Definition
  - User gives username and password once
  - User accesses multiple applications without logging in again
- Enterprise Web SSO: Cookie-based authentication for “internal” web applications (can include cloud-based applications via SAML).
  - Often implemented with agents, gateway or proxies between the browser and the web application that translate SSO cookie into authentication and authorization native to the application

Misc. Photo Credit: Open Clipart

Traditional Enterprise Web SSO

Photo credit: Jamie Dubs

Browser

internal-app1.example.com

internal-app2.example.com

single-sign-on-server.example.com

Traditional Enterprise Web SSO

Photo credit: Jamie Dubs

Browser

1. app1 initial request

internal-app1.example.com

internal-app2.example.com

single-sign-on-server.example.com

Traditional Enterprise Web SSO

Photo credit: Jamie Dubs

Browser

1. app1 initial request

2. Browser redirected to get SSO cookie

internal-app1.example.com

internal-app2.example.com

single-sign-on-server.example.com

Traditional Enterprise Web SSO

Photo credit: Jamie Dubs

Browser

1. app1 initial request

2. Browser redirected to get SSO cookie

3. User provides credentials (pw)

internal-app1.example.com

internal-app2.example.com

single-sign-on-server.example.com

Traditional Enterprise Web SSO

Photo credit: Jamie Dubs

Browser

1. app1 initial request

2. Browser redirected to get SSO cookie

internal-app1.example.com

internal-app2.example.com

3. User provides credentials (pw)

4. Browser receives SSO cookie (←🍪)

single-sign-on-server.example.com

Traditional Enterprise Web SSO

Photo credit: Jamie Dubs

Browser

1. app1 initial request

2. Browser redirected to get SSO cookie

5. app1 req. w/ SSO cookie (🍪)

3. User provides credentials (pw)

4. Browser receives SSO cookie (←🍪)

internal-app1.example.com

internal-app2.example.com

single-sign-on-server.example.com

Traditional Enterprise Web SSO

Photo credit: Jamie Dubs

Browser

1. app1 initial request

2. Browser redirected to get SSO cookie

5. app1 req. w/ SSO cookie (🍪)

6. Request accepted unimpeded!

internal-app1.example.com

internal-app2.example.com

3. User provides credentials (pw)

4. Browser receives SSO cookie (←🍪)

single-sign-on-server.example.com

Traditional Enterprise Web SSO

Photo credit: Jamie Dubs

Browser

1. app1 initial request
2. Browser redirected to get SSO cookie
5. app1 req. w/ SSO cookie (🍪)
6. Request accepted unimpeded!

internal-app1.example.com

7. app2 initial request w/ SSO cookie (🍪)

internal-app2.example.com

3. User provides credentials (pw)
4. Browser receives SSO cookie (←🍪)

single-sign-on-server.example.com

Traditional Enterprise Web SSO

Photo credit: Jamie Dubs

Browser

1. app1 initial request
2. Browser redirected to get SSO cookie
5. app1 req. w/ SSO cookie (🍪)
6. Request accepted unimpeded!

internal-app1.example.com

7. app2 initial request w/ SSO cookie (🍪)
8. Request accepted unimpeded!

internal-app2.example.com

3. User provides credentials (pw)
4. Browser receives SSO cookie (←🍪)

single-sign-on-server.example.com

Anatomy of a Cookie

Photo Credit: Sherrie Thai

Anatomy of a Cookie

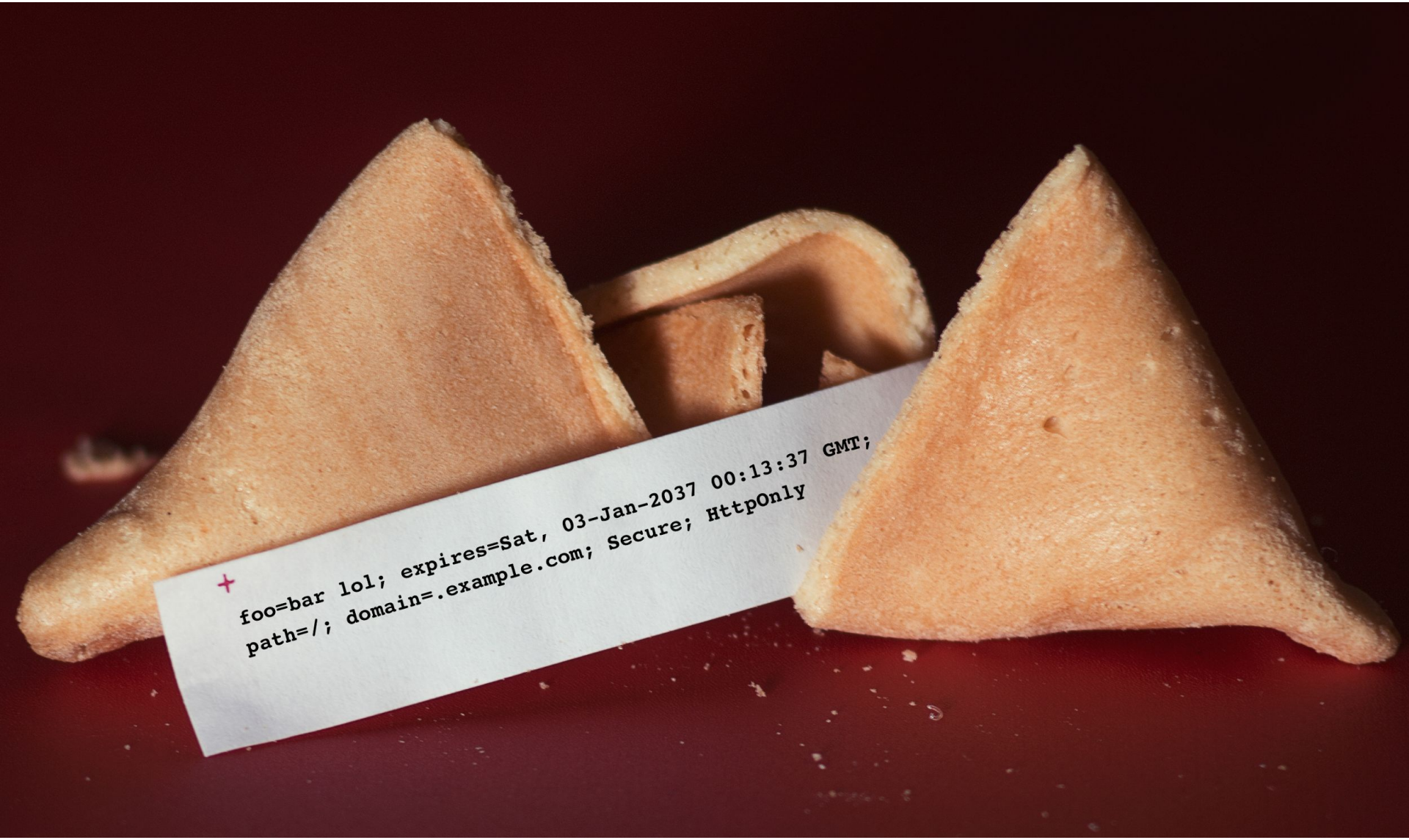


Photo Credit: MattysFlicks

Anatomy of a Cookie

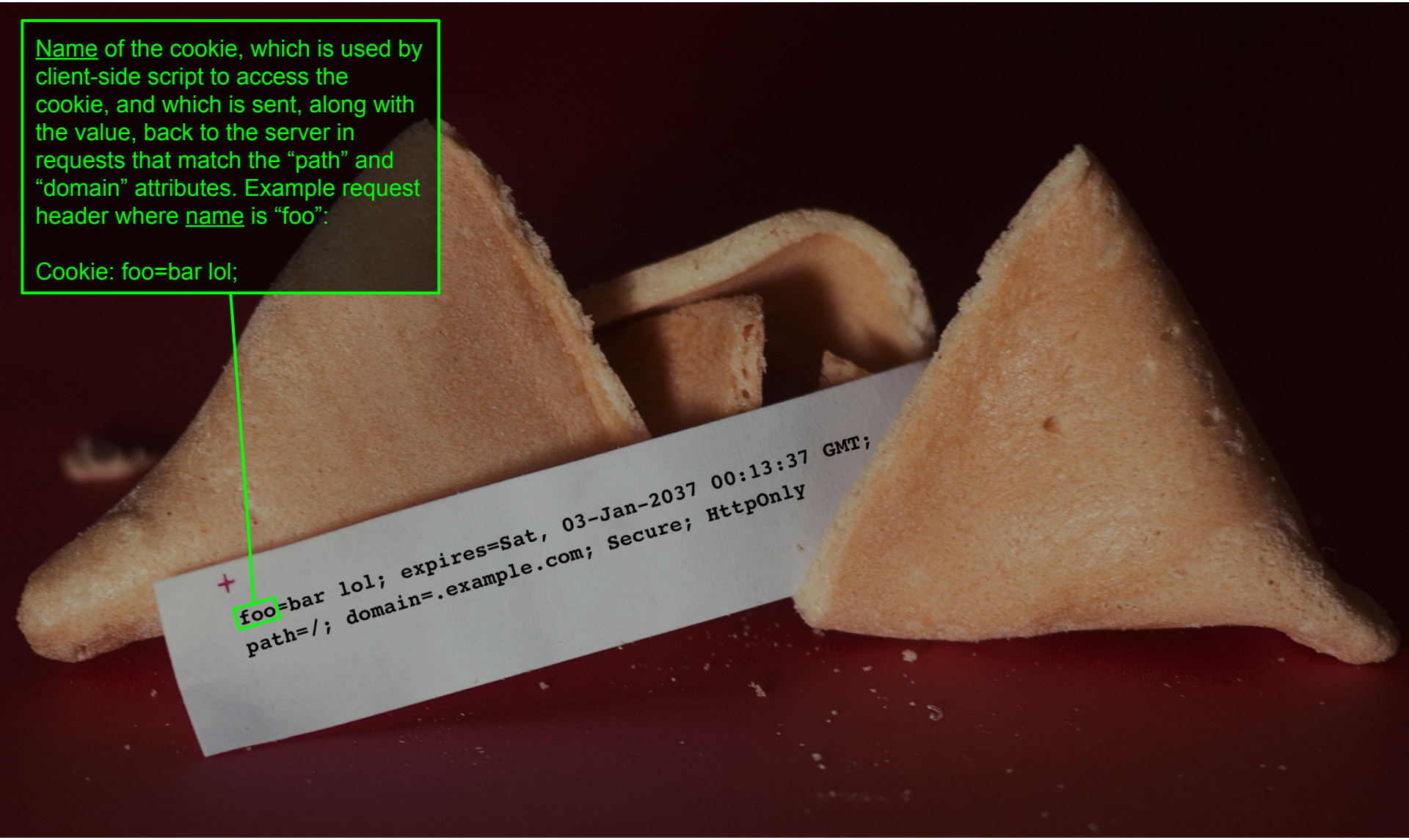


Photo Credit: MattysFlicks

Anatomy of a Cookie

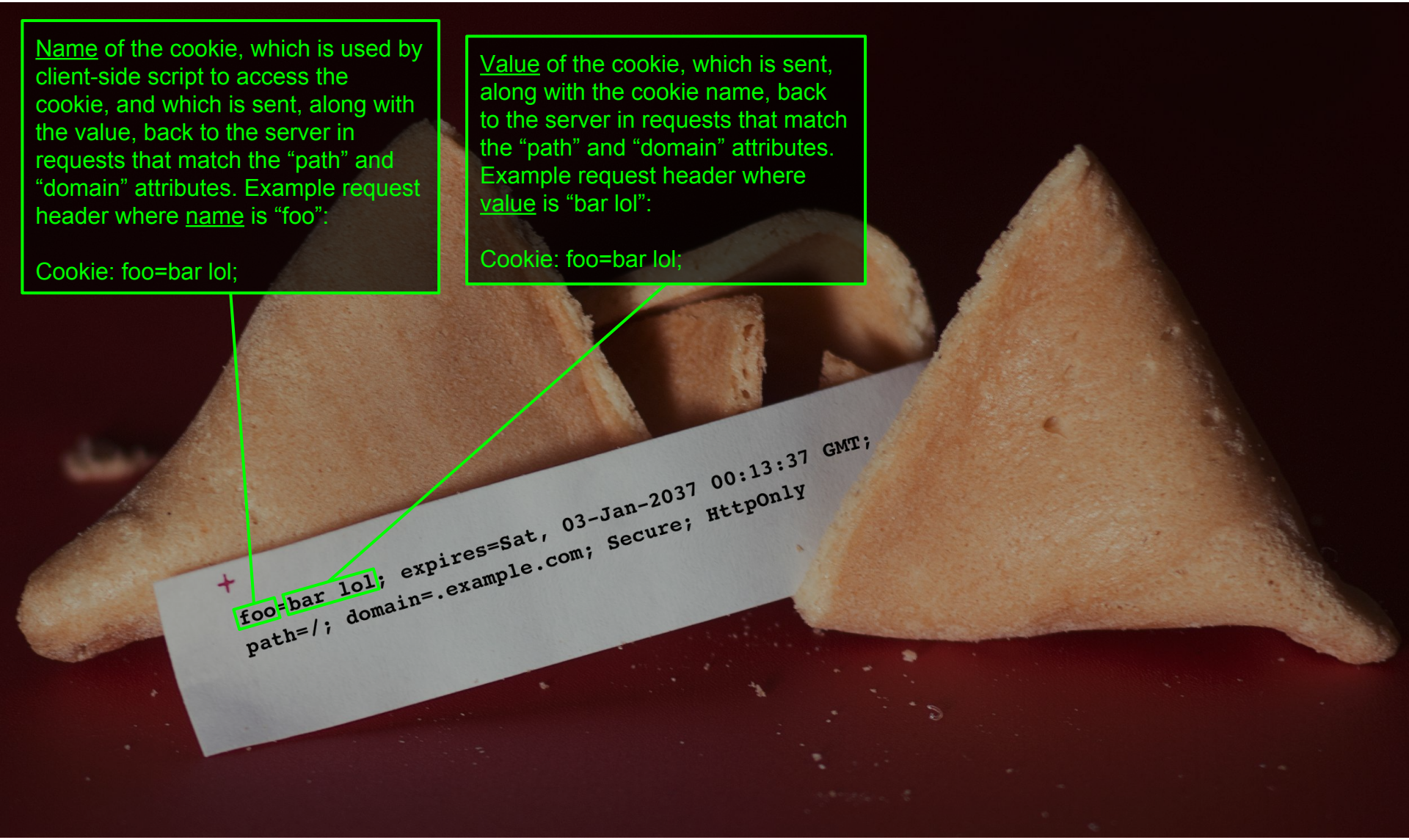


Photo Credit: MattysFlicks

Anatomy of a Cookie

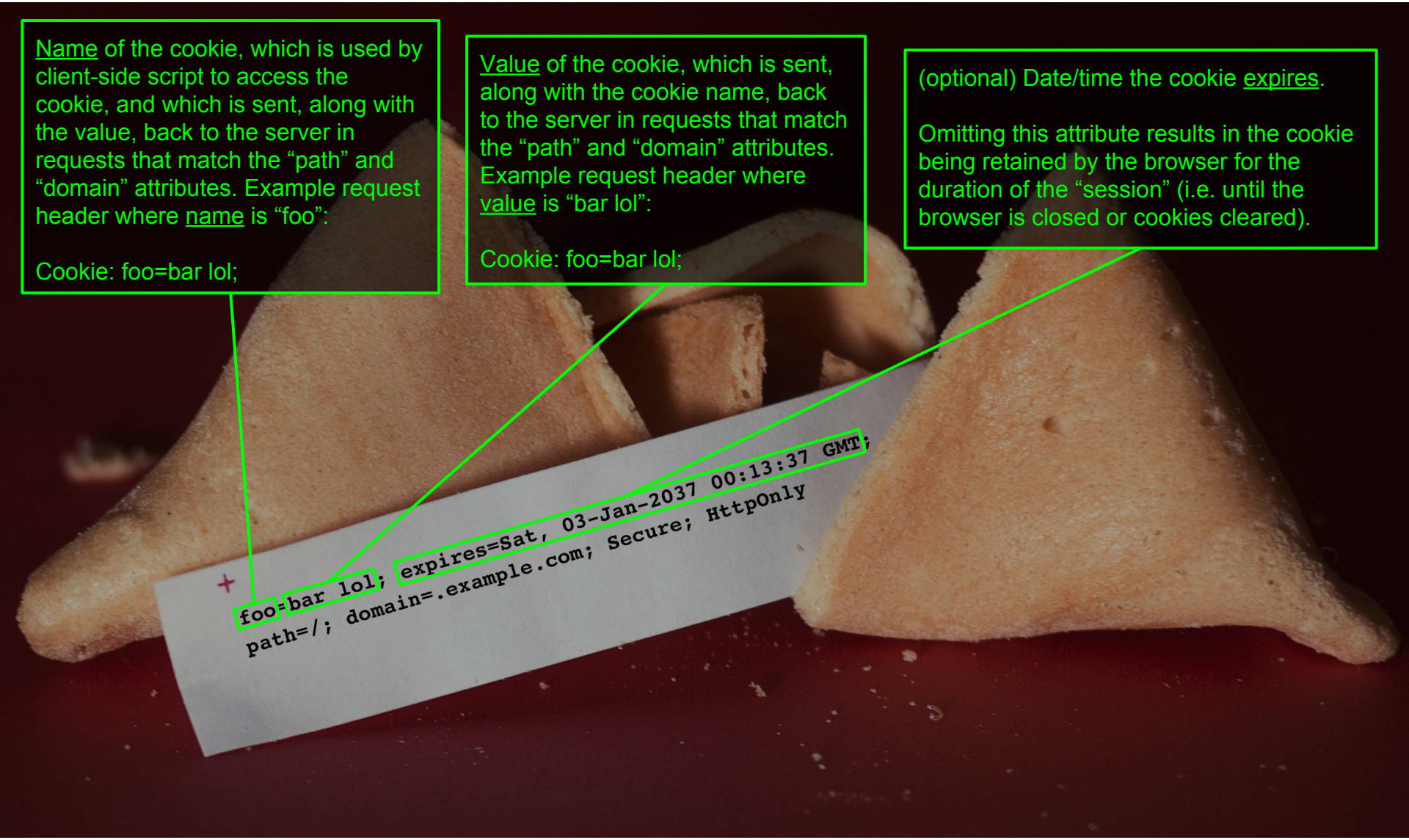


Photo Credit: MattysFlicks

Anatomy of a Cookie

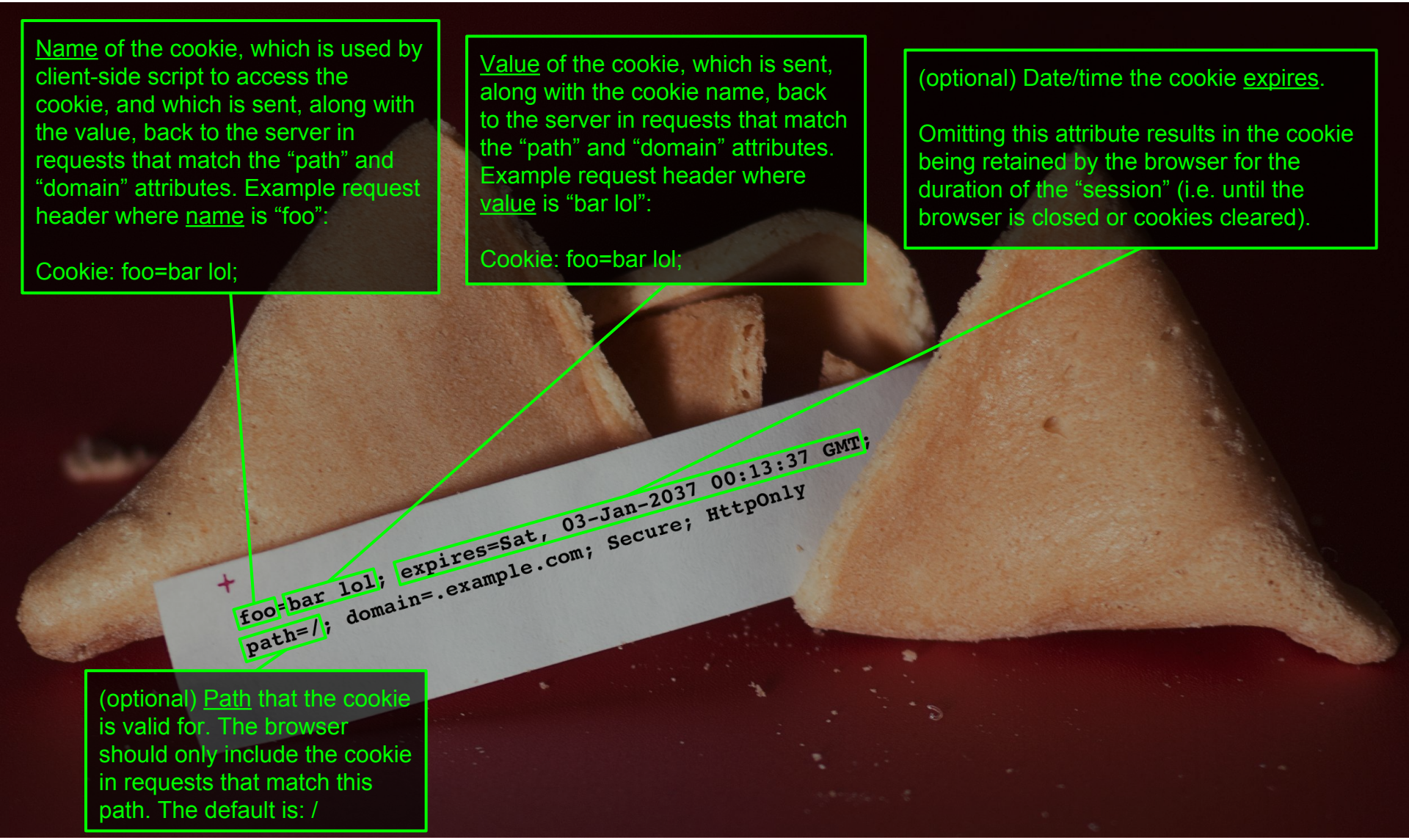
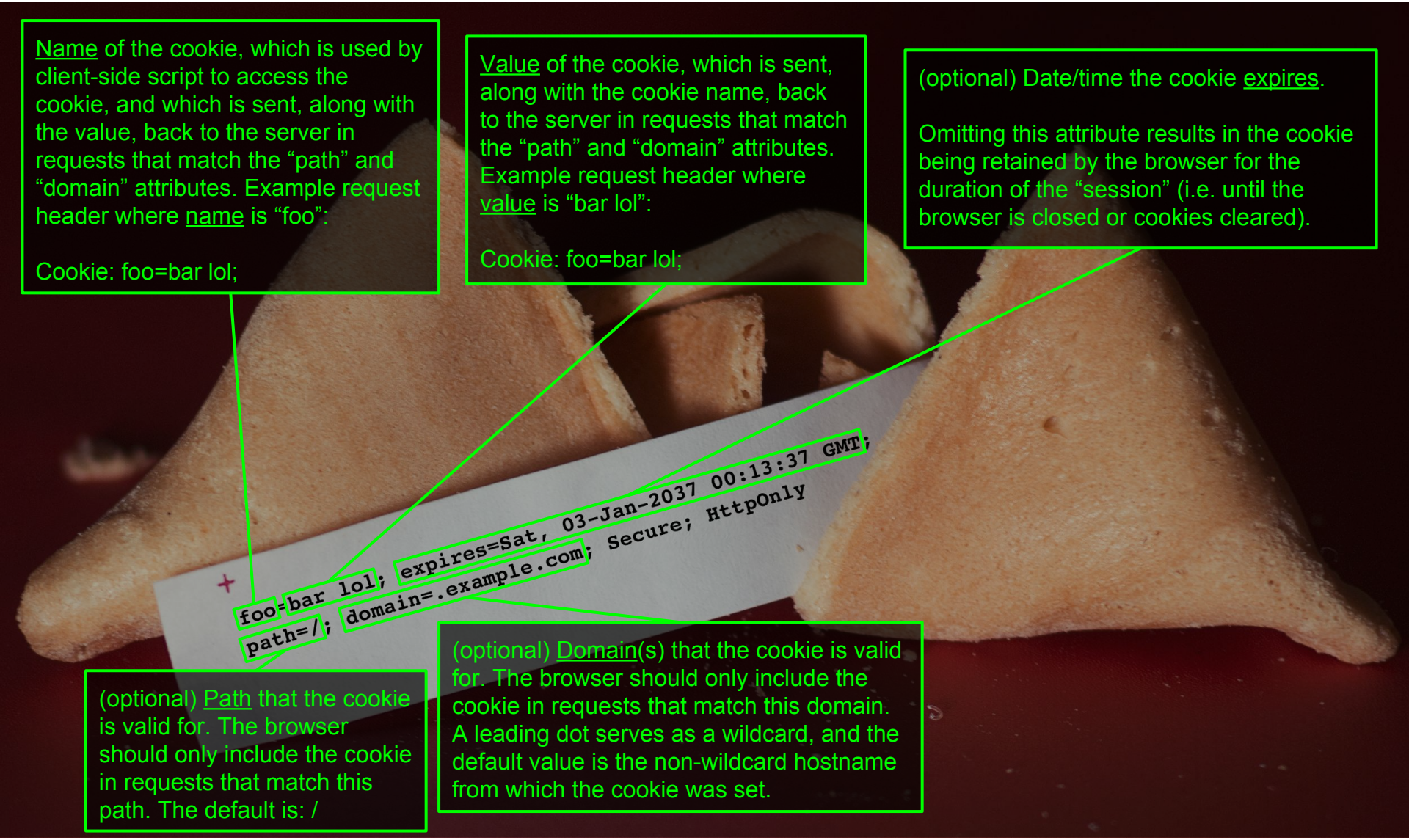


Photo Credit: MattysFlicks

Anatomy of a Cookie



Name of the cookie, which is used by client-side script to access the cookie, and which is sent, along with the value, back to the server in requests that match the “path” and “domain” attributes. Example request header where name is “foo”:

Cookie: foo=bar lol;

Value of the cookie, which is sent, along with the cookie name, back to the server in requests that match the “path” and “domain” attributes. Example request header where value is “bar lol”:

Cookie: foo=bar lol;

(optional) Date/time the cookie expires.

Omitting this attribute results in the cookie being retained by the browser for the duration of the “session” (i.e. until the browser is closed or cookies cleared).

(optional) Path that the cookie is valid for. The browser should only include the cookie in requests that match this path. The default is: /

(optional) Domain(s) that the cookie is valid for. The browser should only include the cookie in requests that match this domain. A leading dot serves as a wildcard, and the default value is the non-wildcard hostname from which the cookie was set.

Photo Credit: MattysFlicks

Anatomy of a Cookie

Name of the cookie, which is used by client-side script to access the cookie, and which is sent, along with the value, back to the server in requests that match the "path" and "domain" attributes. Example request header where name is "foo":

Cookie: foo=bar lol;

Value of the cookie, which is sent, along with the cookie name, back to the server in requests that match the "path" and "domain" attributes. Example request header where value is "bar lol":

Cookie: foo=bar lol;

(optional) Date/time the cookie expires.

Omitting this attribute results in the cookie being retained by the browser for the duration of the "session" (i.e. until the browser is closed or cookies cleared).

(optional) Path that the cookie is valid for. The browser should only include the cookie in requests that match this path. The default is: /

(optional) Domain(s) that the cookie is valid for. The browser should only include the cookie in requests that match this domain. A leading dot serves as a wildcard, and the default value is the non-wildcard hostname from which the cookie was set.

(optional) Secure flag that, when present, instructs the browser to send the cookie over HTTPS connections only. Default behavior, (flag omitted) is to send the cookie over both HTTPS and HTTP connections.

Photo Credit: MattysFlicks

Anatomy of a Cookie

Name of the cookie, which is used by client-side script to access the cookie, and which is sent, along with the value, back to the server in requests that match the "path" and "domain" attributes. Example request header where name is "foo":

Cookie: foo=bar lol;

Value of the cookie, which is sent, along with the cookie name, back to the server in requests that match the "path" and "domain" attributes. Example request header where value is "bar lol":

Cookie: foo=bar lol;

(optional) Date/time the cookie expires.

Omitting this attribute results in the cookie being retained by the browser for the duration of the "session" (i.e. until the browser is closed or cookies cleared).

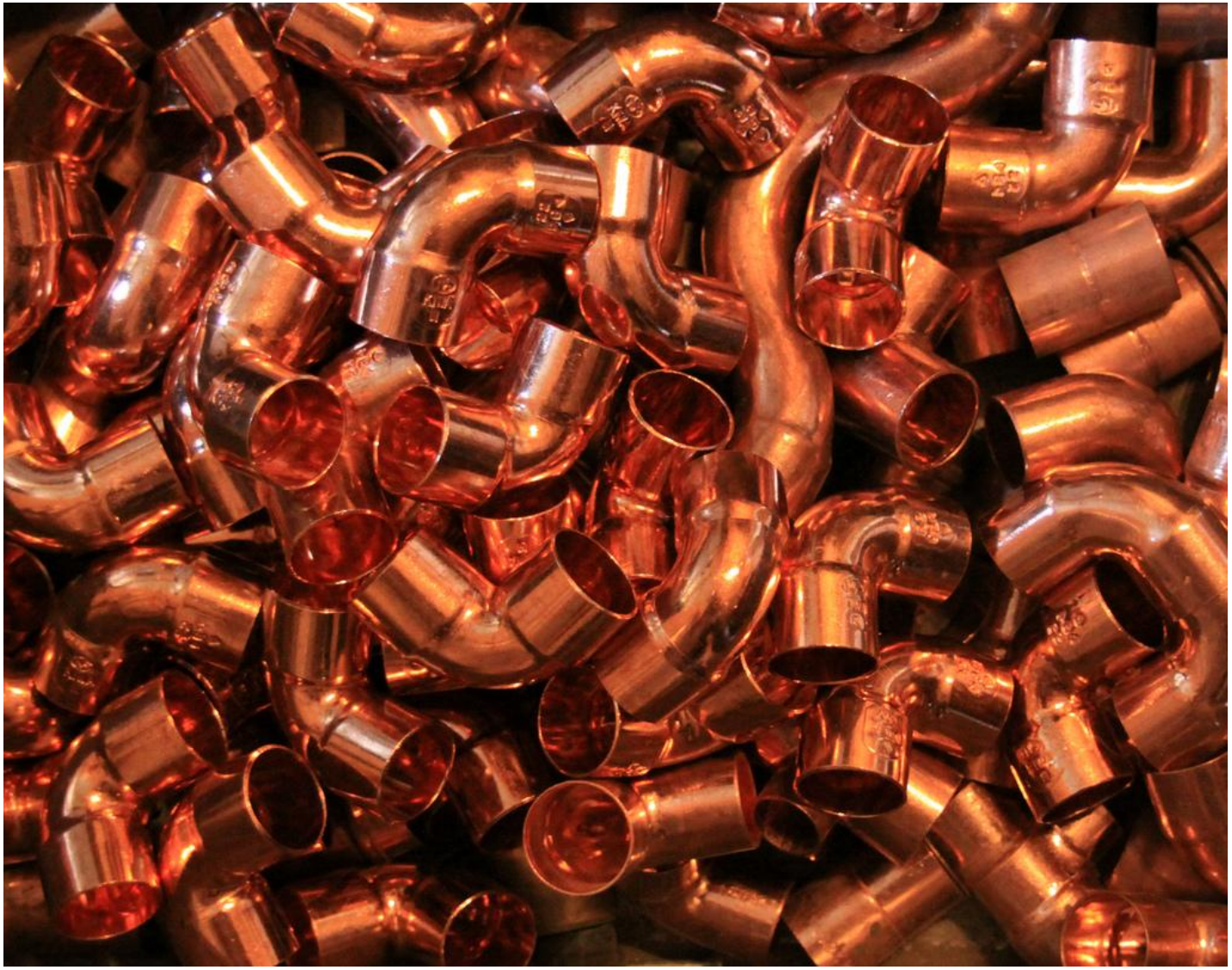
(optional) HttpOnly flag that, when present, instructs the browser to prevent access to the cookie from client-side script. Default behavior (flag omitted) is to allow access from client-side script.

(optional) Path that the cookie is valid for. The browser should only include the cookie in requests that match this path. The default is: /

(optional) Domain(s) that the cookie is valid for. The browser should only include the cookie in requests that match this domain. A leading dot serves as a wildcard, and the default value is the non-wildcard hostname from which the cookie was set.

(optional) Secure flag that, when present, instructs the browser to send the cookie over HTTPS connections only. Default behavior, (flag omitted) is to send the cookie over both HTTPS and HTTP connections.

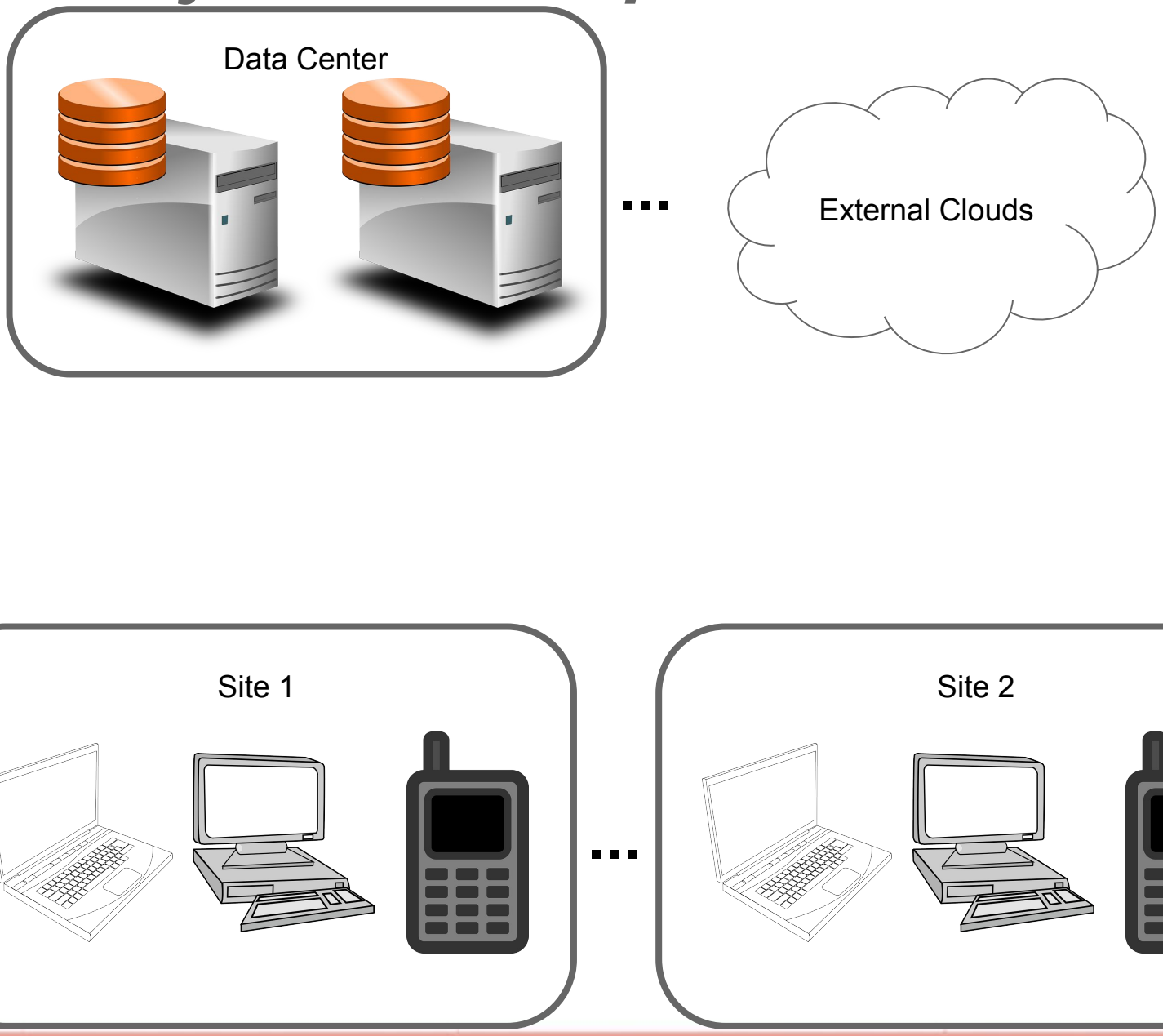
Photo Credit: MattysFlicks



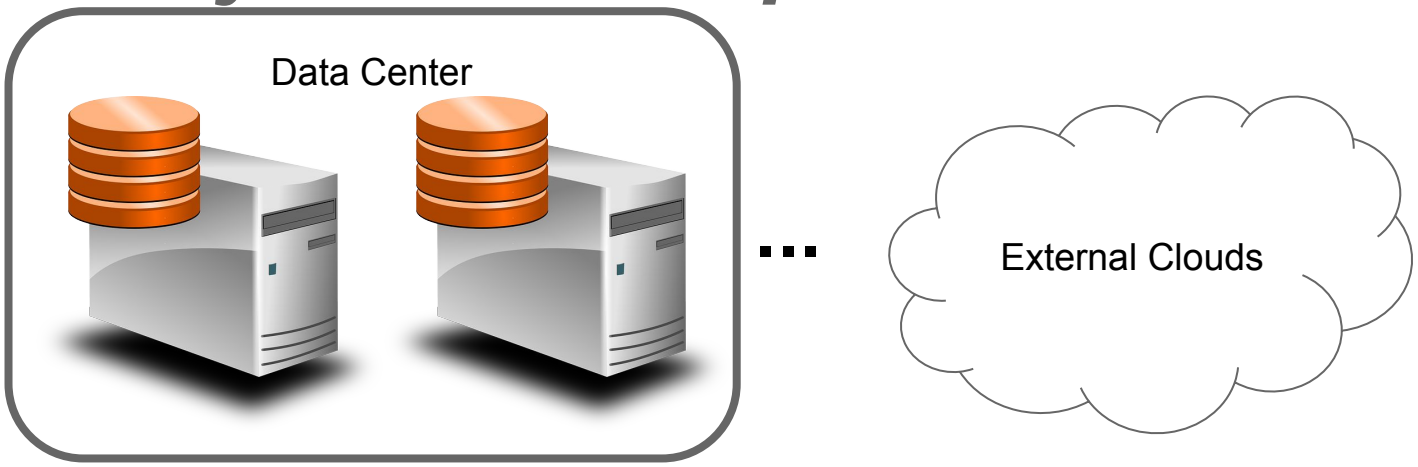
Anatomy of an Enterprise Network

Photo Credit: Tony Hisgett

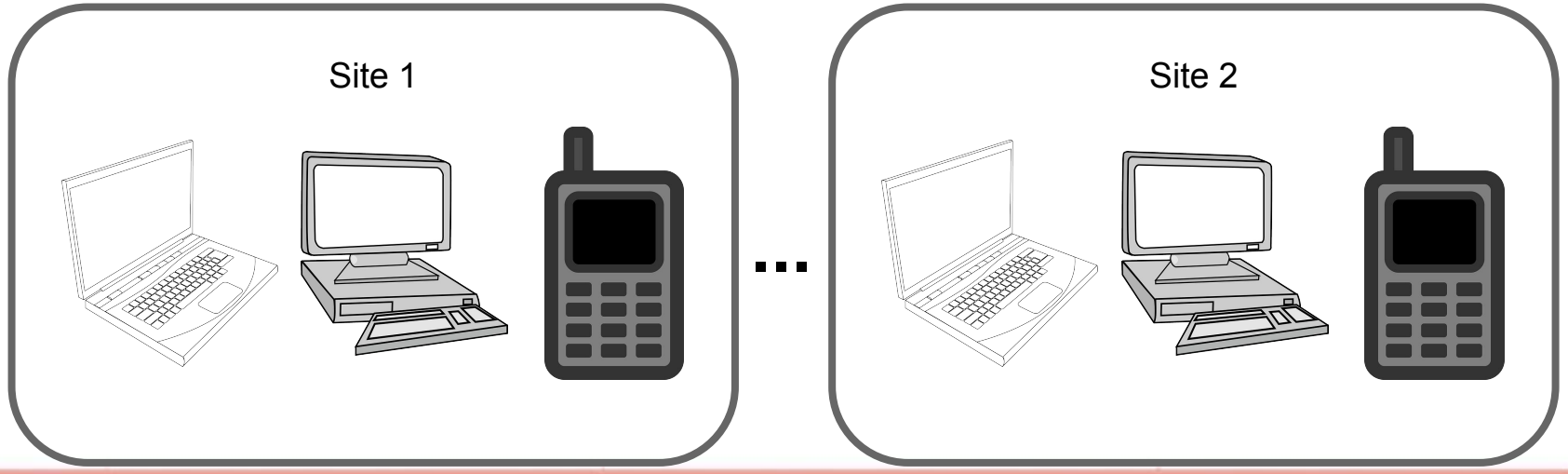
Anatomy of an Enterprise Network



Anatomy of an Enterprise Network

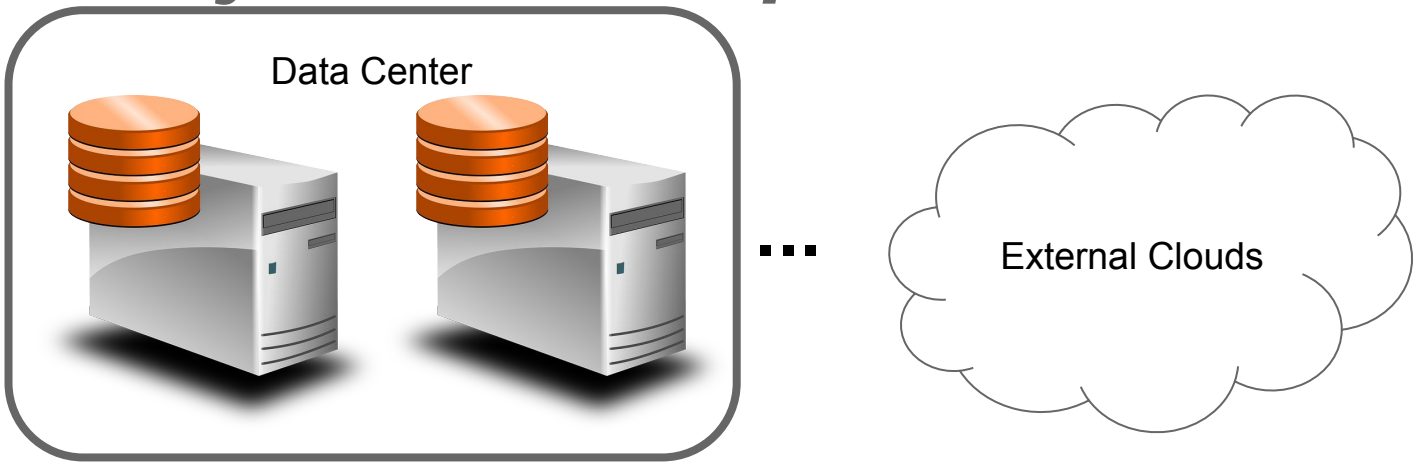


↓Clients↓

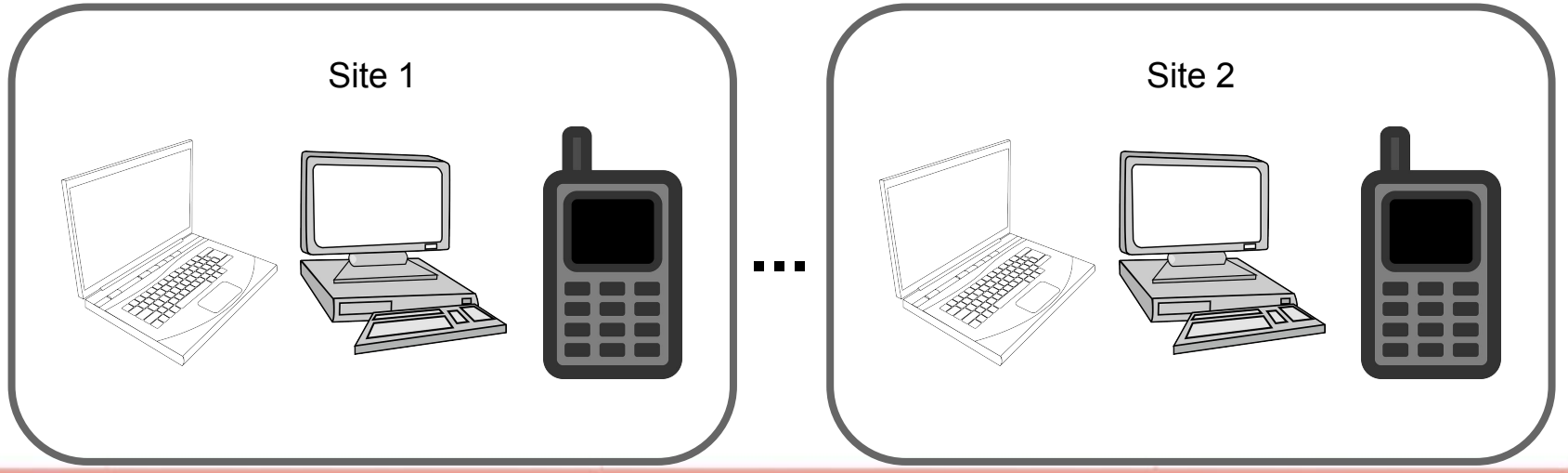


Misc. Photo Credit: Open Clipart

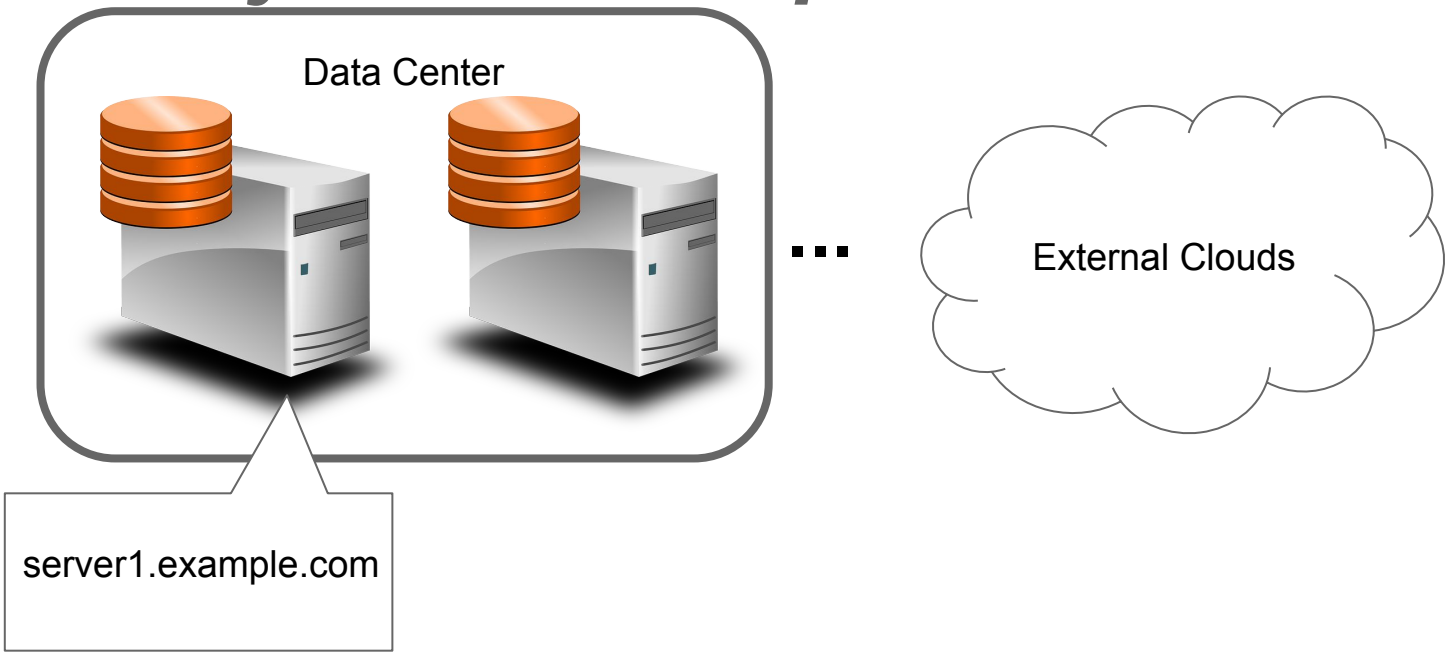
Anatomy of an Enterprise Network



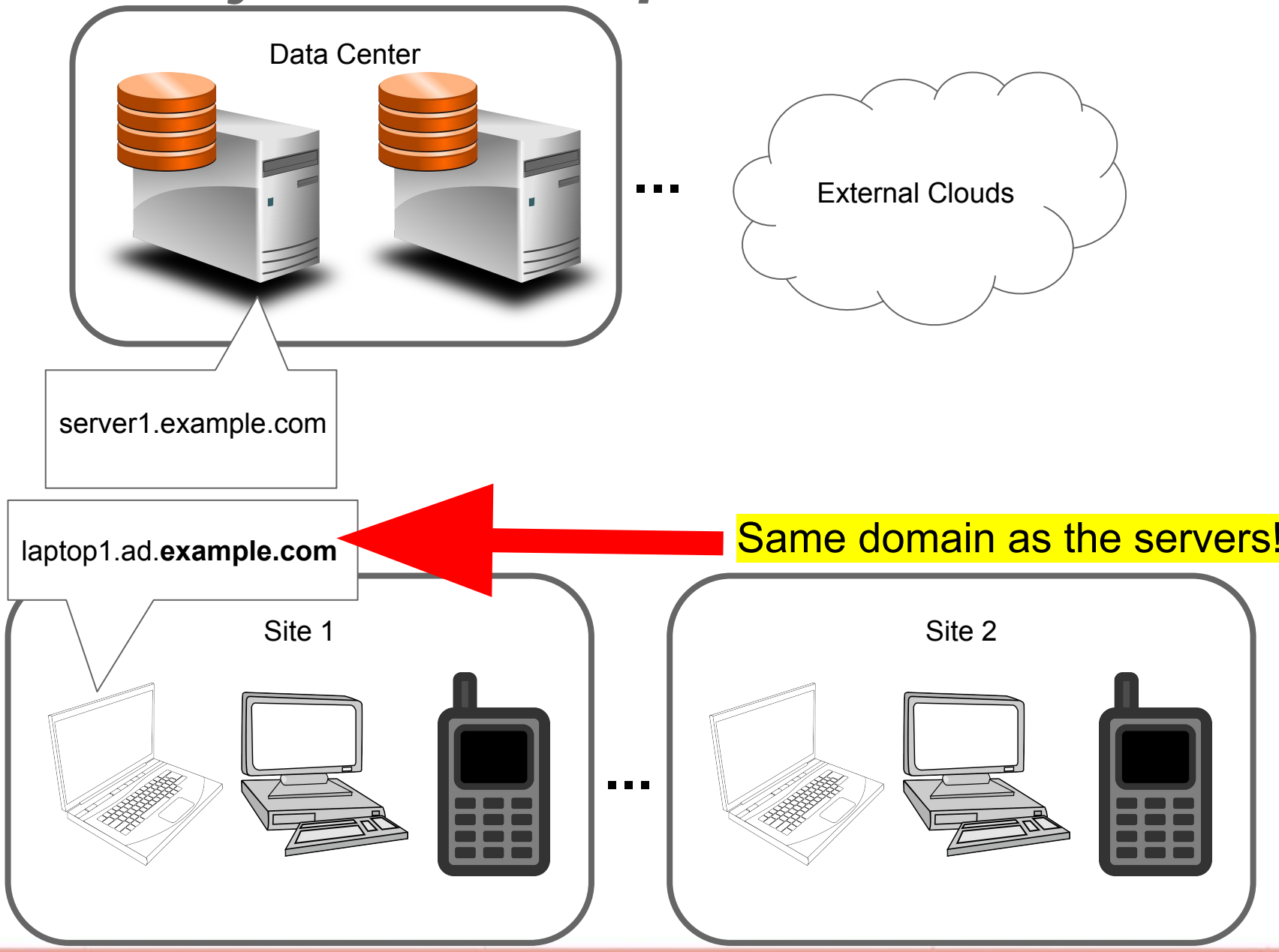
↑Servers, etc.↑



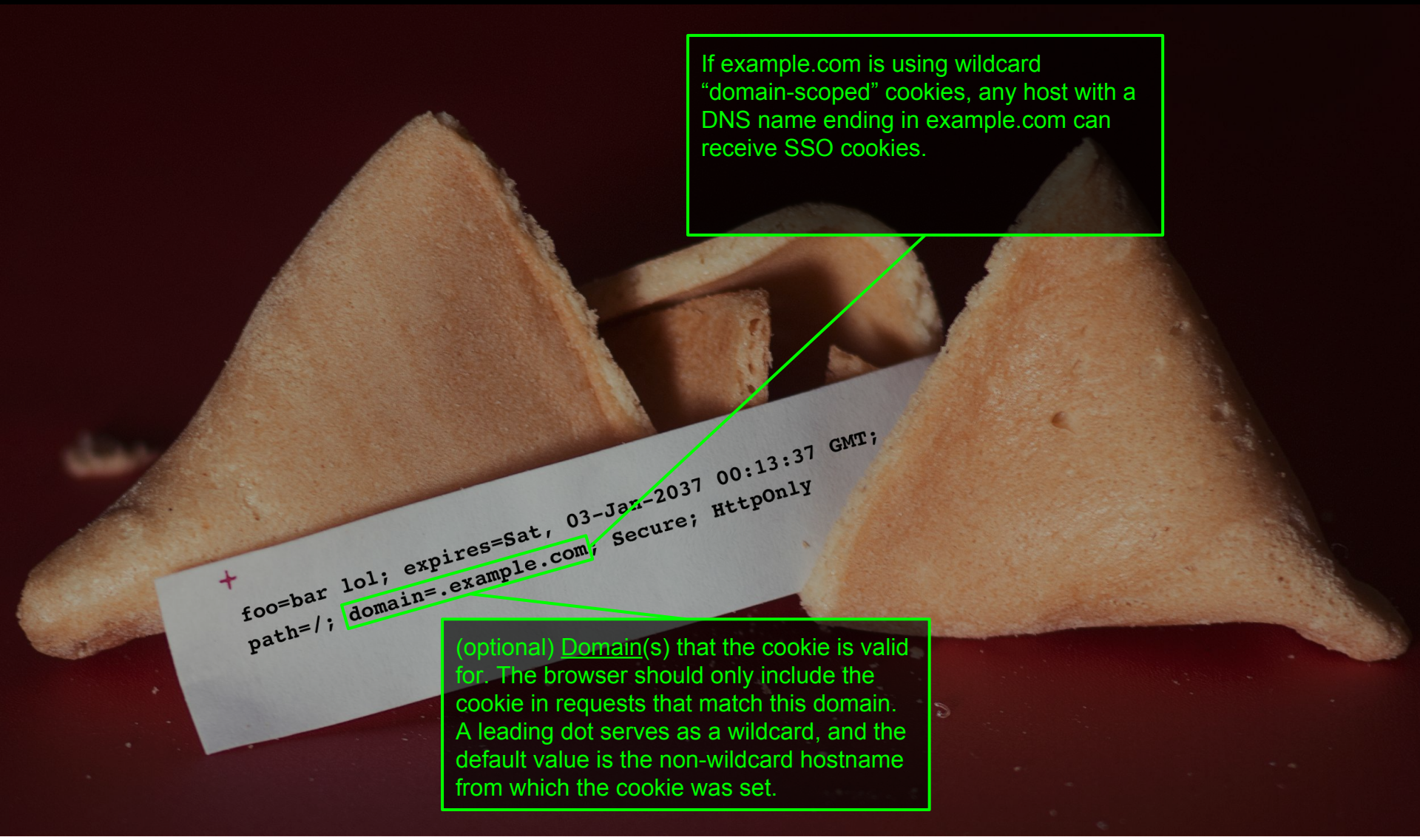
Anatomy of an Enterprise Network



Anatomy of an Enterprise Network



Recall the “Domain” Component!

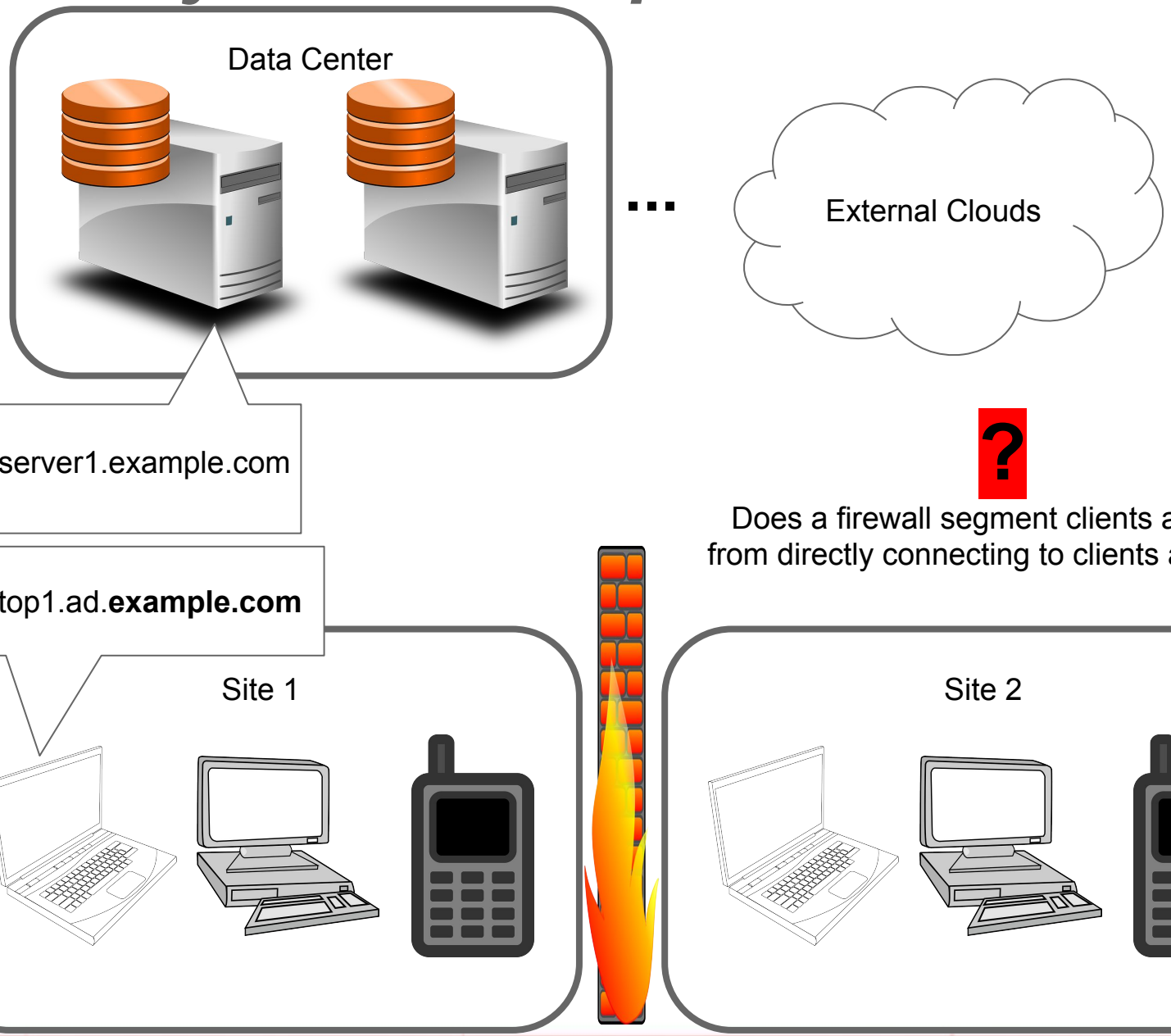


If example.com is using wildcard “domain-scoped” cookies, any host with a DNS name ending in example.com can receive SSO cookies.

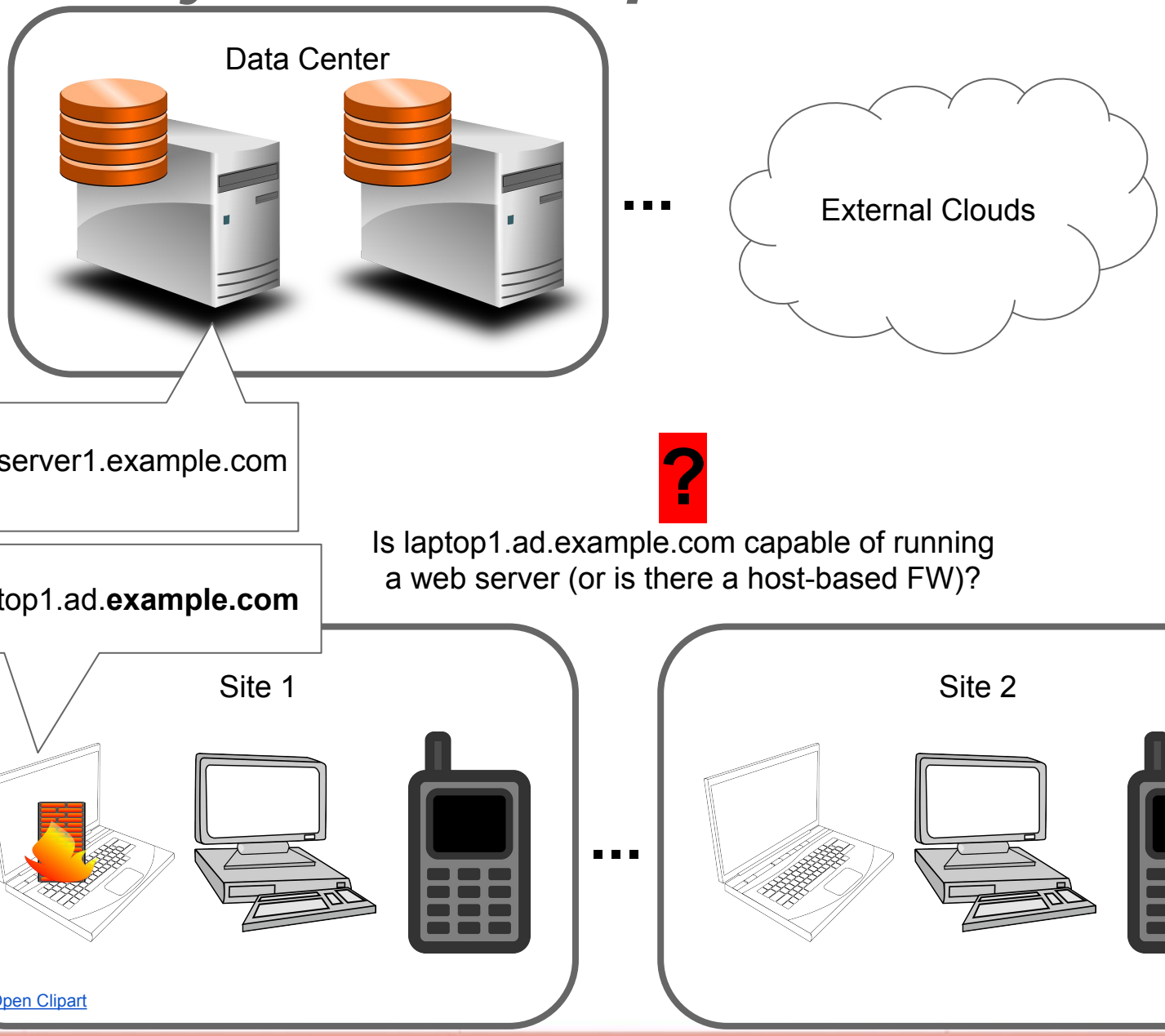
(optional) Domain(s) that the cookie is valid for. The browser should only include the cookie in requests that match this domain. A leading dot serves as a wildcard, and the default value is the non-wildcard hostname from which the cookie was set.

Photo Credit: MattysFlicks

Anatomy of an Enterprise Network



Anatomy of an Enterprise Network



Anatomy of a Security Watering Hole

Photo Credit: Binarysequence

Define "Watering Hole"

Merriam-Webster:

- "a place where water may be obtained; especially: one where animals and especially livestock come to drink"

- "a place where people gather socially"

What does this have to do with computer security?

Description from invincea.com *[emphasis added]*:

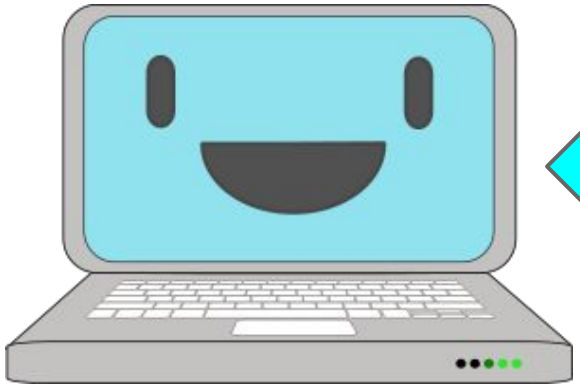
> "Watering hole attacks – or the hijacking of ***legitimate*** websites to push malware…"
>
> "The attacks are two pronged. First, the adversary ***injects malware*** into a ***legitimate website*** without the property owner knowing. Then the malware lays in wait, until ***unsuspecting users*** [...] ***browse*** to that site."

What's the threat?

Hackers Target Popular iOS Forum and Compromise Facebook and Apple Developers! (2013)

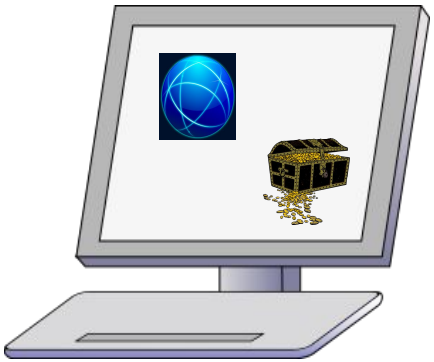
Flash Vulnerability Used In Watering Hole Attack, Attackers Gain Foothold at Aerospace Company! (2015)

Outline of a Watering Hole Attack



Happy users (victims) who are surfing the world wide web to their favorite websites (social media, blogs, news, etc.)

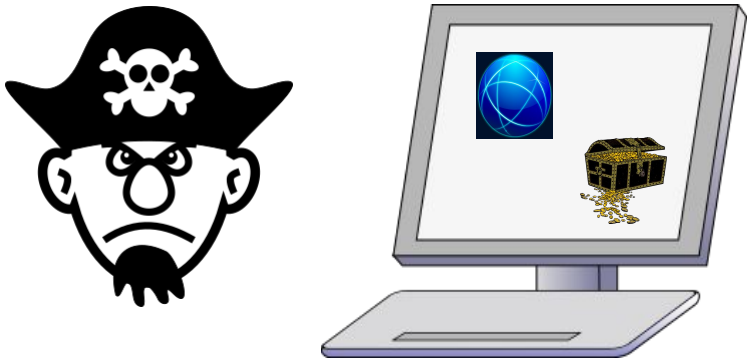
Web servers (legit) visited by users

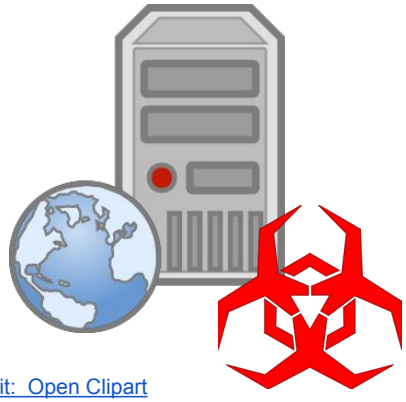


A malicious attacker profiles their victims and the types of websites they visit

Misc. Photo Credit: Open Clipart

Attacker Infects Website



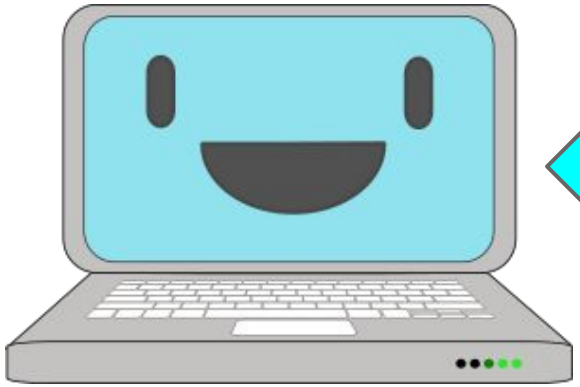


- Website has vulnerability giving attacker ability to inject code
- Website has third party content (like advertisements) where attacker can inject malicious code
- Attacker controls victims network connection and substitutes malicious website but with trusted name
- Users still trust (legit) www.example.com

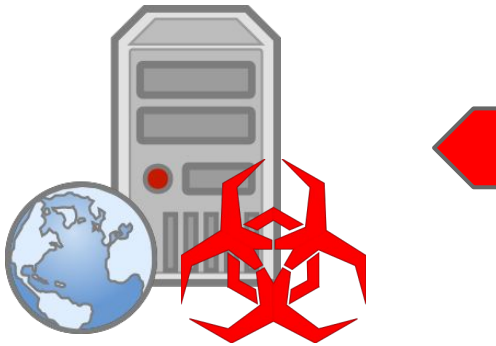
Website (legit) now infected with malware awaiting visitors

Misc. Photo Credit: Open Clipart

Users (victims) Infected



User visits their normal website but unknown to them...



Web site delivers malware to user's computer.

User's computer is now under control of the attacker

The SSO Watering Hole

Photo Credit: milan.boers

Misc. Photo Credit: Open Clipart

Single Sign-on Watering Hole

SNEAK ATTACK!

Attacker can steal user data **WITHOUT** deploying malware on a legit popular website **OR** the target victim's computer!

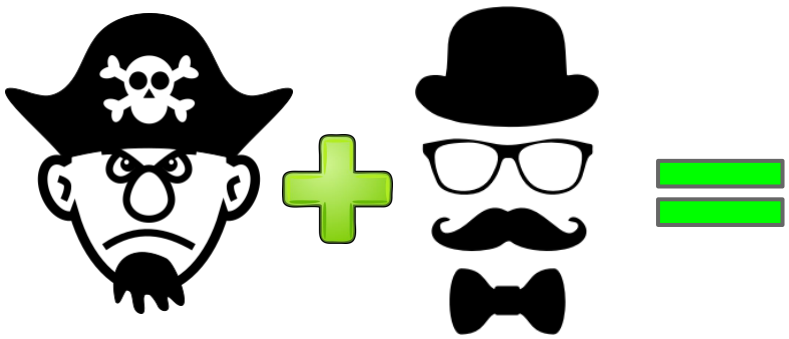


Photo Credit: Edal Anton Lefterov

Misc. Photo Credit: Open Clipart

Simplified Diagram of Keys to Attack

Malicious Insider

1) Plants hidden <img> in blog comment that sources from the insider's computer.

Legit Website
blog.example.com

2) Do normal daily visit to read blogs

3) Tells users to load <img> tag in comments

Happy Users (victims)

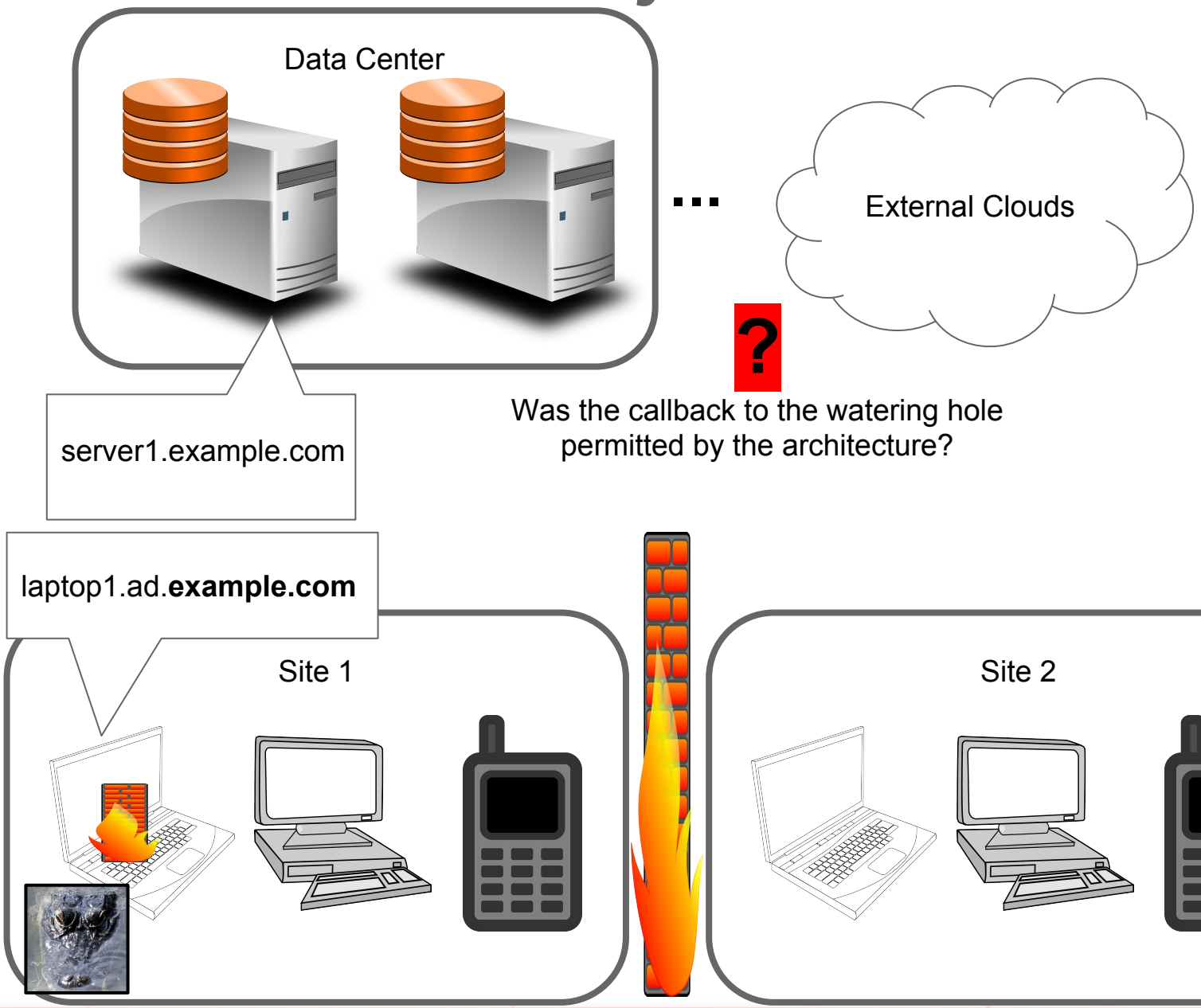
4) Users' web browsers send SSO_Cookie for .example.com to Insider's computer when loading <img> content

Insider's Computer
dev-a234324.ad.example.com

Attacker can now access victim's sensitive data

payroll.int.example.com

Recall the "Anatomy" Slide!



The Fixes

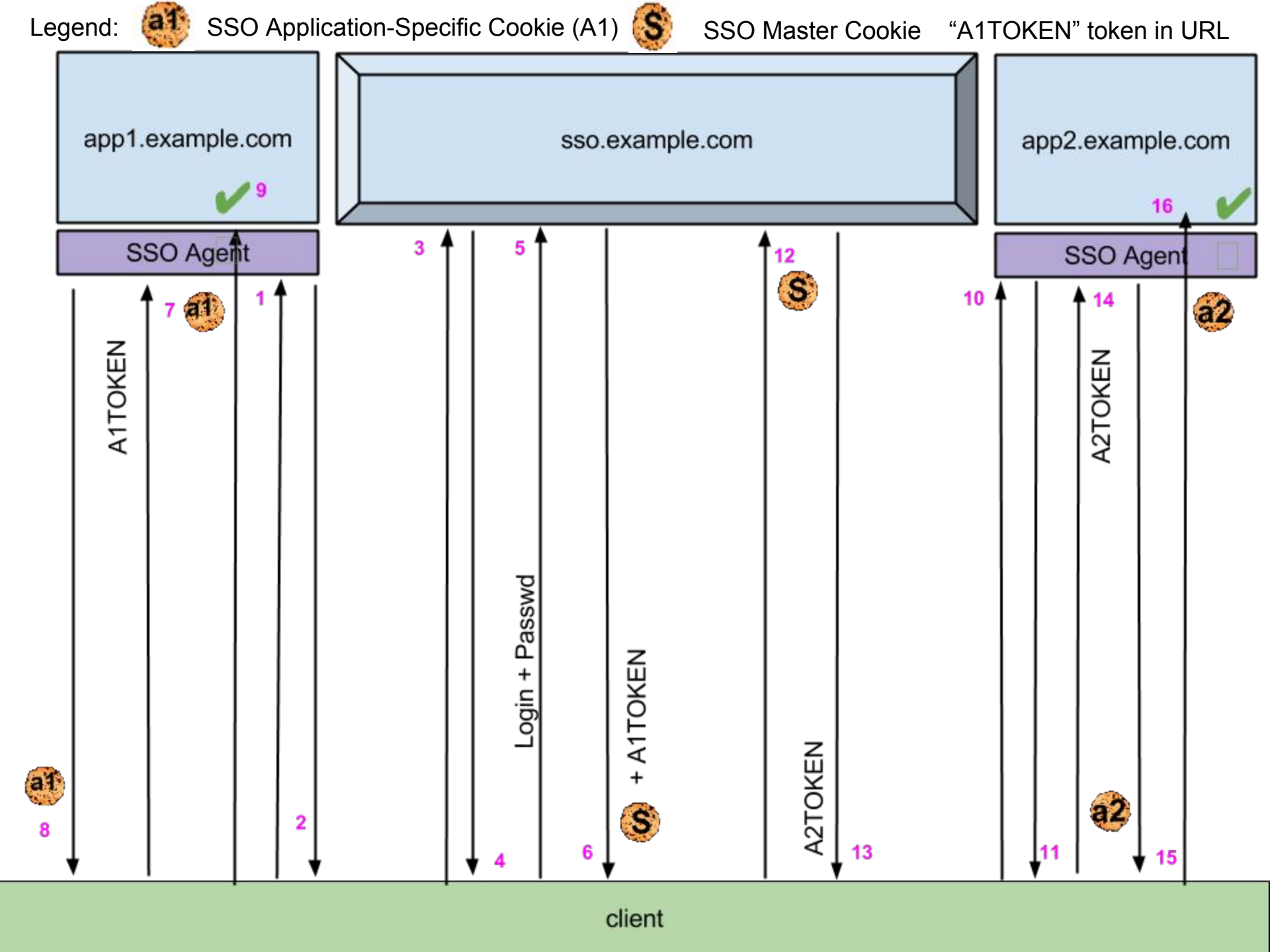
Photo Credit: Tim Parkinson

Implement Multiple Layers

- Drop use of wildcard root domain
  - SSO cookie should not be bound to .example.com
  - Bind it to something like
    - sso-authority.example.com
  - Each web site/app that uses SSO must get its own SSO unique and random cookie value
    - Requires users first visit to redirect back to sso-authority.example.com to get established
      - Typically done with SAML
      - Architecture depends on your SSO provider

Deep Dive: Cookie Provider Model

- Example
  - User comes to work, tries to access app1.example.com
  - Redirected to sso.example.com
  - Establishes session with sso.example.com via login + password
  - sso.example.com redirects user back to app1.example.com with a token value (e.g., GET parameter)
  - User uses token to establish sign-on with app1
  - Similar workflow for app2
- See graphic on next page (resembles SAML)


Legend:
a1
SSO Application-Specific Cookie (A1)
S
SSO Master Cookie
“A1TOKEN” token in URL
app1.example.com
sso.example.com
app2.example.com
SSO Agent
SSO Agent
client
A1TOKEN
Login + Passwd
+ A1TOKEN
A2TOKEN
A2TOKEN
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

Implement Multiple Layers (continued)

- Separate domains for same-origin policy
  - Office network: exa-internal-office.example
  - Intranet data center & sites: exa-intranet.example
  - Public Internet: example.com
  - Every-day internal employees can't host webservers with an example.com name.
- SSO Cookie
  - Use Secure flag
    - Implies all SSO sites must use TLS for https
    - Would make the watering hole scenario more difficult since the attacker would need a valid cert (or the user would have to click through a security warning).
  - Use HttpOnly flag
    - Would not affect the watering hole scenario, but is a good way to enhance security of the SSO environment.

Implement Multiple Layers (continued)

- Leverage firewall for office users
  - John on office floor 2 cannot access http://jane-pc.ad.exa-internal-office.example
  - Jane has no reason to be running a web server for other office workers to access
- What about those pesky developers?
  - Sometimes developer Bob needs to test something on Alice's system
  - Could do firewall rule exceptions
  - Or perhaps encourage a separate dev system on a non-office network such as devs.exa-int-cld.example

Implement Multiple Layers (continued)

- Monitor for hijacked sessions if you can
  - Automate review of all internal web server logs
  - Look for same session or user from multiple IPs
    - False positives occur though from
      - Users roaming from wired to wireless
      - Load balancers masking source IP in the log data
    - Attacker could spoof an IP address too
  - Don’t rely on NAC or device fingerprinting
    - Attacker can spoof these with a little more effort

Implement Multiple Layers

RULE: Don't trust

- The network
- DNS
- Intranet or internal web servers
  - Compromise of serverX should not grant access to SSO session for everything else

RULE: Use multiple layers of defence

- Blocking comments on that one blog isn't enough
- Just because it is behind the firewall isn't enough
  - Attacker could drop hidden link on some popular Internet site that links back to a hidden system on your network.
  - In other words the attack can work around firewalls with enough effort.

What's the Prevalence of this Problem?

Photo Credit: M J Richardson

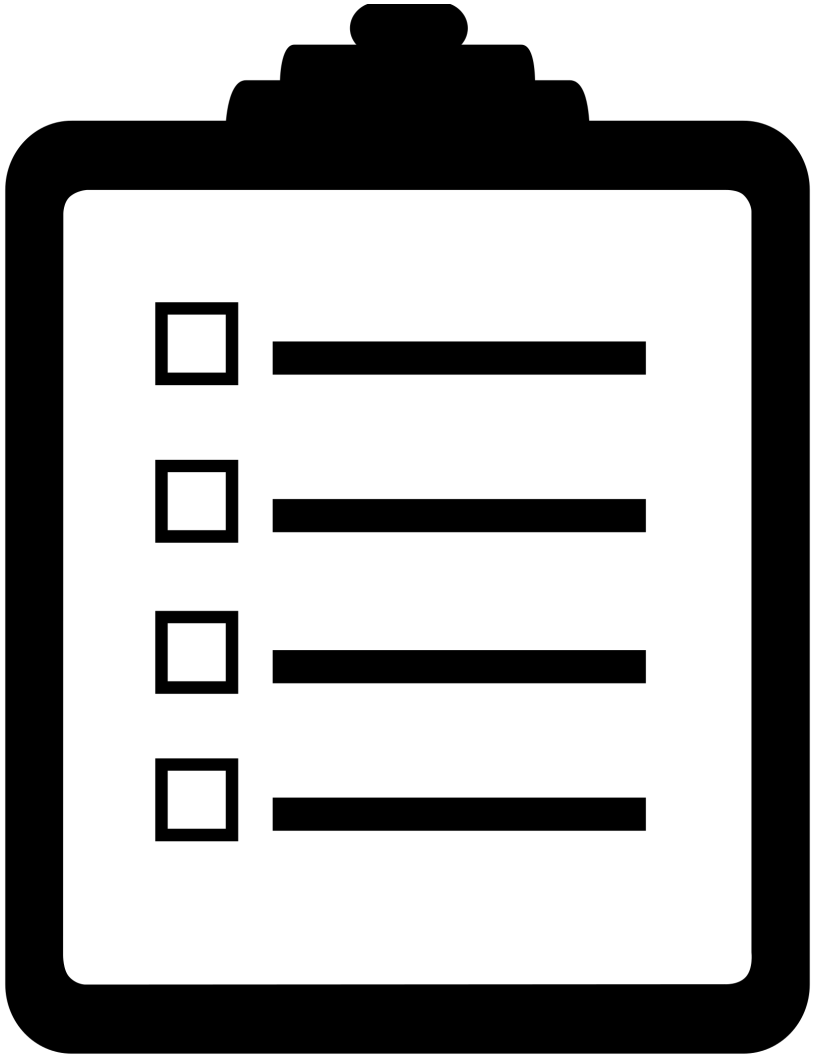
How Prevalent Is This?

Products

- Known to be the default configuration in at least 1 enterprise SSO solution
- Known to be the default configuration of an older version of another enterprise SSO solution

Enterprises

- Investigating prevalence for fortune 500 companies



Vendor Checklist

What Do I Look For?

- Host-Scoped Cookies
  - SAML-based systems support this out of the box
  - “Application zero-trust”: Cookie value stolen from compromised app cannot be used w/ other apps
- Secure and HttpOnly Cookie Flags: Any SSO cookies should have the secure and HttpOnly flags set (RFC6265)
  - This implies the required use of https and TLS, which is necessary to protect SSO cookies from interception.
- Proper Logout: Ensure SSO sessions are invalidated on the back-end
- Replay: Make sure cookies and things that create cookies (e.g., nonces in URLs) either can’t be replayed or have an appropriately short lifespan
- 2FA or step-up challenge

Questions?

Photo Credit: hucky