Defense-in-Depth Engineering

John Poulin

@forced_request

CLOUD SECURITY PARTNERS

Talk Outline

- Introduction
- Technical Content
  - *Designing for Incidents*
  - *Preventing Regression*
  - *Designing for Extensibility*
  - *Understand your Libraries*
- Wrap Up


@forced_request


john@cloudsecuritypartners.com

John Poulin
@forced-request

- CTO, Cloud Security Partners
- Previously: Manager, Product Security Engineering @ GitHub
  - *I managed teams who hacks things*
- Consultant
  - *I hacked things*
- Software engineer
  - *I made things that were hacked*

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

A04:2021 – Insecure Design

- Focuses on risks related to design and architectural flaws, with a call for more use of threat modeling, secure design patterns, and reference architectures.”

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

Defense in Depth?

"Also known as layered defense, defense in depth is a security principle where single points of complete compromise are eliminated or mitigated by the incorporation of a **series or multiple layers of security** safeguards and risk-mitigation countermeasures

Have diverse defensive strategies, so that if one layer of defense turns out to be inadequate, another layer of defense will hopefully prevent a full breach."

- Controls that may be overlooked
- Easier to solve during design phase

@forced_request

john@cloudsecuritypartners.com

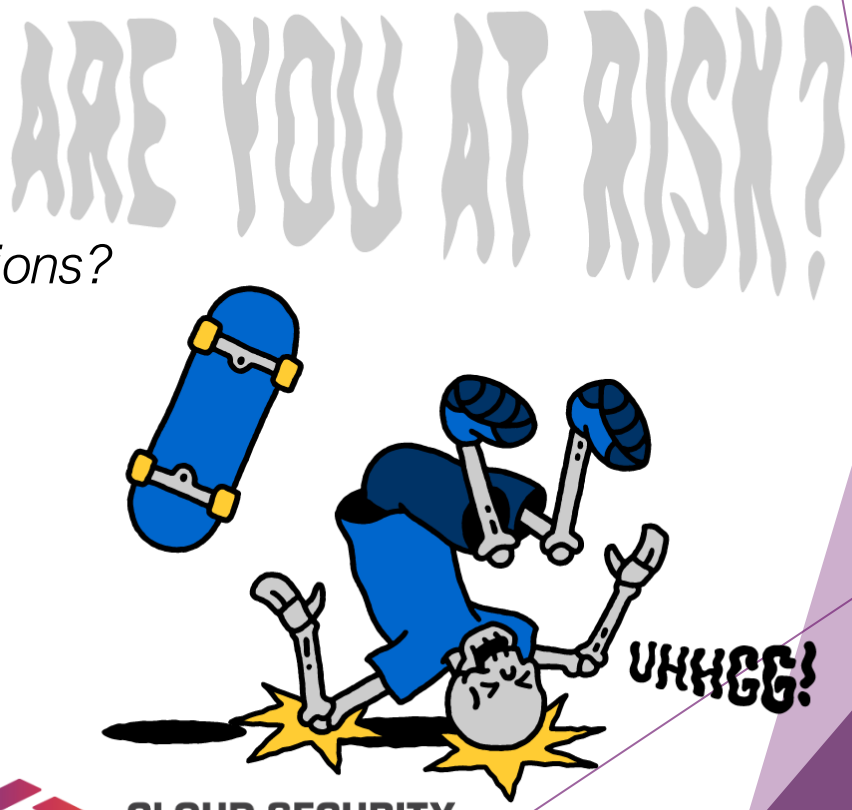
CLOUD SECURITY PARTNERS

Designing for Incidents

Becoming friends with your incident response team.

Understand risk of the system

- What is the riskiest data in your system?
  - *Billing Info, PII, etc.*
- Is all sensitive data documented?
  - *Can IR teams discover these classifications?*
- Are there data flow diagrams?



@forced_request
john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

Build a plan of action

- Work with the team to understand logging desires
  - *What data should be logged?*
  - *What data shouldn't be logged?*
  - *How long should logs live?*
- Understand their expectations from engineering during IR
  - *Will you be in the war room?*
  - *Will there be public communication?*
- Engineering will need to take ownership
- Build playbooks to understand logging infrastructure


@forced_request


john@cloudsecuritypartners.com

Be prepared to own the outcome!

On March 8, we shared that, out of an abundance of caution, we logged all users out of GitHub.com due to a rare security vulnerability. We believe that transparency is key in earning and keeping the trust of our users and want to share more about this bug. In this post we will share the technical details of this vulnerability and how it happened, what we did to respond to it, and the steps we are taking to ensure this does not happen again.

https://github.blog/2021-03-18-how-we-found-and-fixed-a-rare-race-condition-in-our-session-handling/


@forced_request


john@cloudsecuritypartners.com

Investigations will happen

Use the plan of action!

- Build employee trust by focusing on post-mortem improvement.
- Take a breath – this is a team effort
- Blameless IR process

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

SUPPORT CUSTOMERS

Customers have incidents too.

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

Supporting Customers

Customers want to know:

- What / When was data accessed?
- Gaps in coverage means support inquiries and/or unhappy customers.

Providing this data:

- Self-Service Logs
  - *Audit Logging*
  - *Sanitization*
- Dedicated playbooks for customer support team

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

Audit all the things

- Ensure every state-changing request is audited

```ruby
class ApplicationController < ActionController::Base
    after_action :require_audit_trail

    private
    def require_audit_trail
        unless ["GET","HEAD"].include?(request.request_method)
            raise "RequestNotAudited" unless audited?
        end
    end

    def audited?
        Rails.cache.read('audited', raw: true)
    end
end
```



@forced_request


john@cloudsecuritypartners.com

Improve the Things

```ruby
def destroy
    AuditLogEntry.event(
        event: event_for_auditing("Carts#Destroy"),
        actor_id: actor_for_auditing,
        target: target_for_auditing(@cart),
        ip_address: ip_for_auditing
    )

    @cart.product_info = JSON.dump([])
    @cart.save
end
```

```ruby
def require_audit_trail
    unless ["GET","HEAD"].include?(request.
    request_method)
        unless audited?
            if Rails.env.production?
                AuditLogEntry.event(
                    event: event_for_auditing
                    ("AuditTrail#Missing"),
                    actor_id: actor_for_auditing,
                    target: request.path,
                    ip_address: ip_for_auditing
                )
            else
                raise "RequestNotAudited"
            end
    end
end
```


@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

Audit Logs Wall of Shame

A list of vendors that don't prioritize high-quality, widely-available audit logs for security and operations teams.

https://audit-logs.tax/

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

Preventing Regression

Never waste a good incident!

How can we prevent regression?

1

Root Cause Analysis (RCA) during IR process to drive solutions

2

Perform thorough variant analysis

3

Utilize tests to prevent regression

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

READY FOR A TEST?

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

GOAL?

```
13      scenario "attack                                                    , js: true do
14        login(normal_user)
15
16        legit_file = File.join(Rails.root, "public", "data", "legit.txt")
17        File.open(legit_file, "w") { |f| f.puts "totes legit" }
18
19        visit "/users/#{normal_user.id}/benefit_forms"
20        Dir.mktmpdir do |dir|
21          hackety_file = File.join(dir, "test; cd public && cd data && rm -f * ;")
22          File.open(hackety_file, "w") { |f| f.print "mwahaha" }
23          within(".new_benefits") do
24            attach_file "benefits_upload", hackety_file
25            find(:xpath, "//input[@id='benefits_backup']", visible: false).set "true"
26          end
27          click_on "Start Upload"
28        end
29
30        expect(File.exist?(legit_file)).to be_truthy
31      end
32    end
```

https://github.com/OWASP/railsgoat

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

Potential Improvements

- Utilize **unit testing** when possible, to validate underlying implementation
- Consider alternative payloads
  - *Sleep*
- Consider alternative encodings
  - *URL Encoded, Unicode*
- Happy Path?
- False Negatives?
  - *Race Condition*

```
13    scenario "attack                                                    , js: true do
14      login(normal_user)
15
16      legit_file = File.join(Rails.root, "public", "data", "legit.txt")
17      File.open(legit_file, "w") { |f| f.puts "totes legit" }
18
19      visit "/users/#{normal_user.id}/benefit_forms"
20      Dir.mktmpdir do |dir|
21        hackety_file = File.join(dir, "test; cd public && cd data && rm -f * ;")
22        File.open(hackety_file, "w") { |f| f.print "mwahaha" }
23        within(".new_benefits") do
24          attach_file "benefits_upload", hackety_file
25          find(:xpath, "//input[@id='benefits_backup']", visible: false).set "true"
26        end
27        click_on "Start Upload"
28      end
29
30      expect(File.exist?(legit_file)).to be_truthy
31    end
32  end
```

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

ONE LAST TEST

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

GOAL?

```
27          context 'given text containing script tags' do
28            let(:text) { '<script>alert("Hello")</script>' }
29
30            it 'strips the scripts' do
31              is_expected.to_not include '<script>alert("Hello")</script>'
32            end
33          end
34
35          context 'given text containing malicious classes' do
36            let(:text) { '<span class="mention  status__content__spoiler-link">Show more</span>' }
37
38            it 'strips the malicious classes' do
39              is_expected.to_not include 'status__content__spoiler-link'
40            end
41          end
```

https://github.com/mastodon/mastodon

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

Prevent rendering of script tags

- What if input is lowercased, normalized, etc?
- What if comparison is for equality, and no substring/match?
- Are there tests for other attributes?
- What about malicious ID's?
- Which are the legitimate classes?

```
27          context 'given text containing script tags' do
28            let(:text) { '<script>alert("Hello")</script>' }
29
30            it 'strips the scripts' do
31              is_expected.to_not include '<script>alert("Hello")</script>'
32            end
33          end
34
35          context 'given text containing malicious classes' do
36            let(:text) { '<span class="mention  status__content__spoiler-link">Show more</span>' }
37
38            it 'strips the malicious classes' do
39              is_expected.to_not include 'status__content__spoiler-link'
40            end
41          end
```

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

Advice for designing tests

Ensure tests are utilized, and a part of the development and deployment process.

Focus test on a specific issue

Integration test XSS on user profile

Determine method to make test reusable

Can we instrument test helpers with common XSS payloads that can be reused across the app?

Establish tests as close to the logic as possible.

Unit test for validating comparison mechanisms.

Focus on testing for *permutations*.


john@cloudsecuritypartners.com

Permutations: An Easy Win

- Reusable solution to test for encoding concerns
- Reduce risk of input-transformation causing false negatives.
- Should be used as both input and output tests

```ruby
# Find permutations of data
def permutations_of(input)
  permutations = [input]
  permutations << Base64.encode64(input)
  permutations << URI.encode(input)
  permutations << URI.decode(input)
  permutations << input.upcase
  permutations << input.downcase
end

def assert_no_permutation(needle, haystack)
  permutations_of(needle).each do |p|
    assert_no_match(Regexp.new(p), haystack)
  end
end
```


@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

Designing for Extensibility

Product will change, prepare for exceptions.

SELECT password FROM users WHERE `id=$id`

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

SELECT createdAt FROM `$tablename` WHERE `country='de'`

- What's the solution here?

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

Parameterized Queries Won't Mitigate Risk

- Table name cannot be parameterized.
- An alternative: **allow-listing** known table names.


@forced_request


john@cloudsecuritypartners.com

Solution

- Utilize framework controls or middleware to introspect parameters.
- Better: Additionally filter at the method level to provide better traceability.

```
2   before_filter :set_table
3   VALID_TABLES = %w(analytics, reports)
4
5   def index
6       query = "SELECT createdAt FROM #{@curent_table} WHERE country='DE'"
7       results = ActiveRecord::Base.connection.execute(sql)
8   end
9
10  private
11  def set_table
12      raise "InvalidTable" unless VALID_TABLES.include?(params[:table])
13
14      @curent_table = params[:table]
15  end
        [:table])
    13  end
    14
```

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

Accept the reality: insecure code patterns will be used.

- Provide a supported exception process
- Annotate insecure code
  - *Link to discussion, exception, etc.*
  - *Train AI models to ignore insecure code?*
- Regularly review exceptions and prioritize based on risk.

Key is being able to **document** and **audit** for these decisions.

```ruby
# INSECURE: Potential Mass assignment
def create
    product = Product.find(params[:review][:product])

    review = Review.create(
        content: params[:review][:content],
        product: product,
        user: current_user || User.first
    )

    redirect_to product
end
```

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

Annotation for CI


```
#!/bin/bash
EXPECTED_COUNT=3

OUT=`rails notes --annotations TODO|wc -l`

if [ $OUT -ne $EXPECTED_COUNT ]; then
  echo "Expected $EXPECTED_COUNT TODOs, found $OUT"
  exit 1 # https://www.cyberciti.biz/faq/linux-bash-exit-status
fi      You, 4 days ago • Added todo annotations script
```

```
class CartsController < ApplicationController
    before_action :set_cart_for_user, only: [:show, :
    def index
        # INSECURE: Used for debugging
        @carts = Cart.all
        render json: @carts
    end
```

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

BUILD PAVED PATHS FOR EXTENSIBILITY

CLOUD SECURITY
PARTNERS

Security headers are the new bank-grade encryption

- Content-Security-Policy
- Strict-Transport-Security
- X-Frame-Options
- X-Content-Type-Options
- Referrer-Policy
- Permissions-Policy

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

Content-Security-Policy

Huge help in *reducing the risk* of Cross-Site Scripting

- Easiest to roll out early in app. Lifecycle.
- Start with `default-src 'self'` and iterate adding exceptions.
  - *Script-src*
  - *Connect-src*

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

BUT WHAT IF...?

We want to run A/B tests utilizing
a new vendor?

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY
PARTNERS

CSP can be set per-page

- Utilize one site-wide, default policy
- Add exceptions at a per-page basis
  - *Exceptions should be reviewed by security.*
- Tighten the policy while providing developers the ability to seek exception

```
class ApplicationController < ActionController::Base
  SecureHeaders::Configuration.default do |config|
    config.csp = {
      default_src: %w('self'),
      script_src: %w(example.org)
    }
  end

  # override default configuration
  SecureHeaders::Configuration.override(:script_from_otherdomain_com) do |config|
    config.csp[:script_src] << "otherdomain.com"
  end
end

class MyController < ApplicationController
  def index
    # Produces default-src 'self'; script-src example.org otherdomain.com
    use_secure_headers_override(:script_from_otherdomain_com)
  end

  def show
    # Produces default-src 'self'; script-src example.org otherdomain.org evenanot
    use_secure_headers_override(:another_config)
  end
end
```

https://github.com/github/secure_headers

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

A clear example

GitHub.com/readme

content-security-policy: default-src 'none'; base-uri 'self'; block-all-mixed-content; child-src github.com/assets-cdn/worker/ gist.github.com/assets-cdn/worker/; connect-src 'self' uploads.github.com objects-origin.githubusercontent.com www.githubstatus.com collector.github.com raw.githubusercontent.com api.github.com github-cloud.s3.amazonaws.com github-production-repository-file-5c1aeb.s3.amazonaws.com github-production-upload-manifest-file-7fdce7.s3.amazonaws.com github-production-user-asset-6210df.s3.amazonaws.com cdn.optimizely.com logx.optimizely.com/v1/events *.actions.githubusercontent.com wss://*.actions.githubusercontent.com online.visualstudio.com/api/v1/locations github-production-repository-image-32fea6.s3.amazonaws.com github-production-release-asset-2e65be.s3.amazonaws.com insights.github.com insights-api-staging.service.iad.github.net wss://alive.github.com; font-src github.githubassets.com; form-action 'self' github.com gist.github.com objects-origin.githubusercontent.com; frame-ancestors 'none'; frame-src viewscreen.githubusercontent.com notebooks.githubusercontent.com; img-src 'self' data: github.githubassets.com identicons.github.com github-cloud.s3.amazonaws.com secured-user-images.githubusercontent.com/ github-production-user-asset-6210df.s3.amazonaws.com customer-stories-feed.github.com spotlights-feed.github.com *.githubusercontent.com **images.ctfassets.net/s5uo95nf6njh**/; manifest-src 'self'; media-src github.com user-images.githubusercontent.com/ secured-user-images.githubusercontent.com/ **github.githubassets.com assets.ctfassets.net/s5uo95nf6njh/ downloads.ctfassets.net/s5uo95nf6njh/**; script-src github.githubassets.com; style-src 'unsafe-inline' github.githubassets.com; worker-src github.com/assets-cdn/worker/ gist.github.com/assets-cdn/worker/; report-uri https://api.github.com/_private/browser/errors

GitHub.com/forced-request

content-security-policy: default-src 'none'; base-uri 'self'; block-all-mixed-content; child-src github.com/assets-cdn/worker/ gist.github.com/assets-cdn/worker/; connect-src 'self' uploads.github.com objects-origin.githubusercontent.com www.githubstatus.com collector.github.com raw.githubusercontent.com api.github.com github-cloud.s3.amazonaws.com github-production-repository-file-5c1aeb.s3.amazonaws.com github-production-upload-manifest-file-7fdce7.s3.amazonaws.com github-production-user-asset-6210df.s3.amazonaws.com cdn.optimizely.com logx.optimizely.com/v1/events *.actions.githubusercontent.com wss://*.actions.githubusercontent.com online.visualstudio.com/api/v1/locations github-production-repository-image-32fea6.s3.amazonaws.com github-production-release-asset-2e65be.s3.amazonaws.com insights.github.com insights-api-staging.service.iad.github.net wss://alive.github.com; font-src github.githubassets.com; form-action 'self' github.com gist.github.com objects-origin.githubusercontent.com; frame-ancestors 'none'; frame-src viewscreen.githubusercontent.com notebooks.githubusercontent.com; img-src 'self' data: github.githubassets.com identicons.github.com github-cloud.s3.amazonaws.com secured-user-images.githubusercontent.com/ github-production-user-asset-6210df.s3.amazonaws.com customer-stories-feed.github.com spotlights-feed.github.com *.githubusercontent.com; manifest-src 'self'; media-src github.com user-images.githubusercontent.com/ secured-user-images.githubusercontent.com/; script-src github.githubassets.com; style-src 'unsafe-inline' github.githubassets.com; worker-src github.com/assets-cdn/worker/ gist.github.com/assets-cdn/worker/; report-uri https://api.github.com/_private/browser/errors

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

Implementation

before_action :csp_override

def csp_override
  use_secure_headers_override(:stripe)
end

```
106   SecureHeaders::Configuration.override(:dompurify) do |config|
107     config.csp[:script_src] << "https://cdnjs.cloudflare.com"
108   end
109
110   SecureHeaders::Configuration.override(:stripe) do |config|
111     config.csp[:script_src] << "https://js.stripe.com"
112   end
```

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

Edge cases in XSS

Output encoding is not a silver bullet.

Use CSP, *don't* rely on it.

"Content Security Policy (CSP) is **an added layer of security** that helps to detect and mitigate certain types of attacks, including Cross-Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft, to site defacement, to malware distribution."

https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP


@forced_request


john@cloudsecuritypartners.com

Output encoding is still key

- Frameworks generally perform output encoding by default
  - *Largest concern is with exemption (I.e., Rails' .html_safe)*
  - *Most likely not context specific*
- Attribute injection is relatively common
  - *Many WAF's look for <brackets>*
  - *Unquoted attributes are dangerous*
    - <p data-id=<%= params[:id] %>>
- Input validation is helpful, but is defense-in-depth

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY
PARTNERS

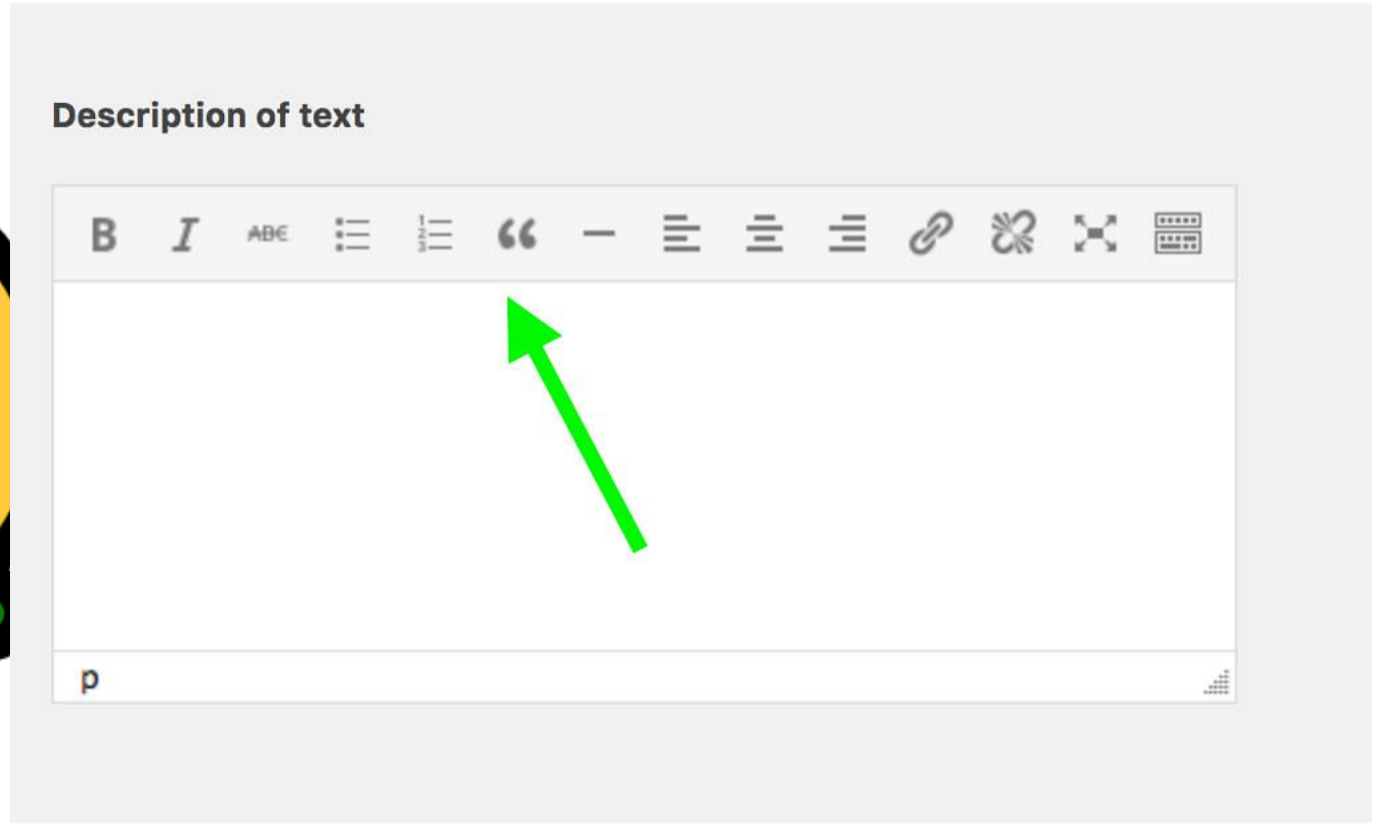
BUT WHAT IF…?

We want to allow users to use
some HTML tags?

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY
PARTNERS



@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

Goes back to exceptions

Problem:

- Output encoding is likely insufficient now.
  - *Our secure coding policies won't work*
- Developers will look to the framework's way of disabling output encoding.

Solution:

- Have a plan
  - *Identify list of tags that need to be supported*
  - *Identify list of attributes that need to be supported*
  - *Understand the context in which the tags will be rendered*
- Utilize a trusted sanitizer
  - *Spend the time to understand the library*


@forced_request


john@cloudsecuritypartners.com

DOMPURIFY IS BAE

@forced_request

john@cloudsecuritypartners.com

How do I use it?

It's easy. Just include DOMPurify on your website.

Using the unminified development version

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

Using the minified and tested production version (source-map available)

```
<script type="text/javascript" src="dist/purify.min.js"></script>
```

Afterwards you can sanitize strings by executing the following code:

```
let clean = DOMPurify.sanitize(dirty);
```

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

Know your libraries

"*Note that by default, we permit HTML, SVG and MathML*"

The resulting HTML can be written into a DOM element using `innerHTML` or the DOM using `document.write()`. That is fully up to you. Note that by default, we permit HTML, SVG **and** MathML. If you only need HTML, which might be a very common use-case, you can easily set that up as well:

```
let clean = DOMPurify.sanitize(dirty, { USE_PROFILES: { html: true } });
```



@forced_request


john@cloudsecuritypartners.com

What *is* default behavior?

- Will HTML tags be rendered as HTML, removed, or encoded??
  - *Roughly 110 tags will be evaluated as HTML. https://github.com/cure53/DOMPurify/blob/main/src/tags.js*
- Are attributes supported?
  - *Yes, over 100. https://github.com/cure53/DOMPurify/blob/main/src/attrs.js*
- Does it utilize an AllowList or Denylist approach?
  - *AllowList, with ability to explicitly deny tags*


@forced_request


john@cloudsecuritypartners.com

What's the expectation?

```
$(".eventTrigger").on("click", () => {
  let payload = '<a href="https://site.com" class="link" data-url="https://malicious.com">Test</a>';
  let clean = DOMPurify.sanitize(payload);
  document.getElementById("sandbox").innerHTML  = clean;
})
```

```
<div id="banner-message">…</div>
<div id="sandbox">
  <a data-url="https://malicious.com" class="link" href="https://site.com">Test</a>
</div>
```

https://jsfiddle.net/bwLokh75/17/

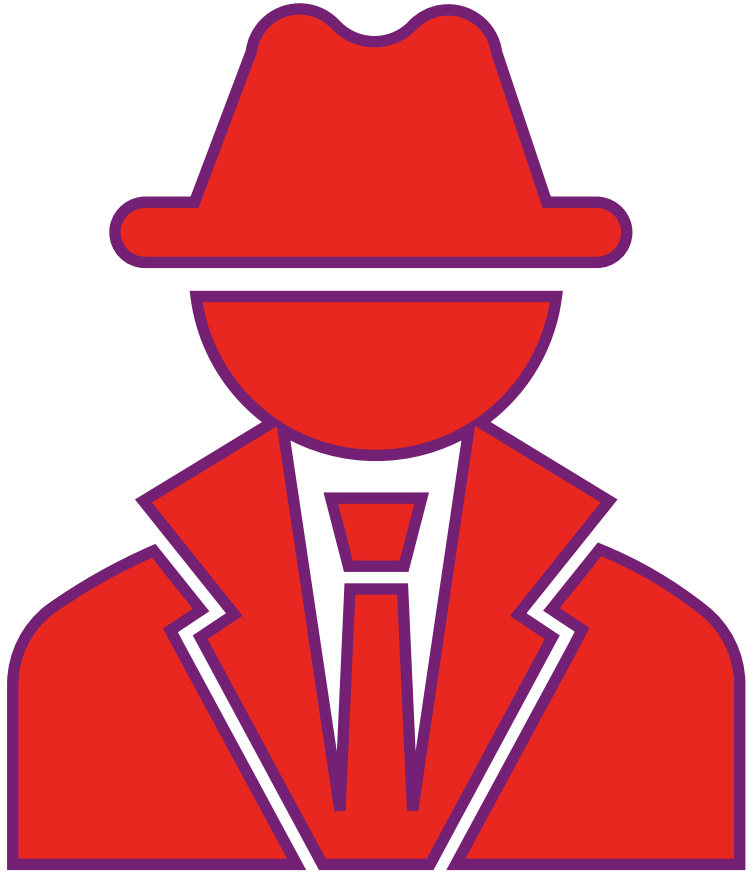
@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

Major Takeaway

- We can inject:
  - *Classes*
  - *ID*
  - *Data Attributes*
  - *More..*
- Event Handlers?



@forced_request
john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

1. Button clicked, triggers .eventTrigger
2. Event handler sanitizes payload, injects to the DOM.
3. DOM element contains findReviews class, which when clicked triggers another callback.
4. Invokes showReviews function
5. Performs HTTP request. Response is injected directly into the DOM

```
<div id="banner-message">
  <button class="eventTrigger">Click here to execute</button>
  Payload: <input type="text" id="payload">
</div>

<div id="sandbox"></div>
<div id="reviews"></div>
```

```
$(".eventTrigger").on("click", () => {
  var payload = $("#payload").val();
  document.getElementById("sandbox").innerHTML = DOMPurify.sanitize(clean
})

$(document).on("click", ".findReviews", function () {
  var itemId = $(this).data("item-id");
  showReviews(itemId);
});

function showReviews(itemId) {
  $.get(apiEndpoint, function(data) {
    document.getElementById("reviews").innerHTML = data;
  });
}

// <div class="findReviews" data-item-id="22"">asdfasdf</div>
```

https://jsfiddle.net/1ntcuvLx/53/

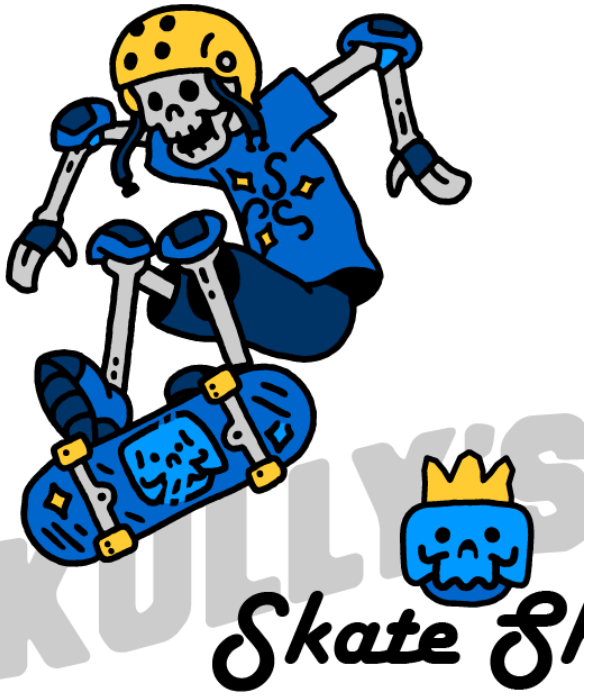
@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS

DOM XSS IS NEARLY IMPOSSIBLE TO INVESTIGATE

Example

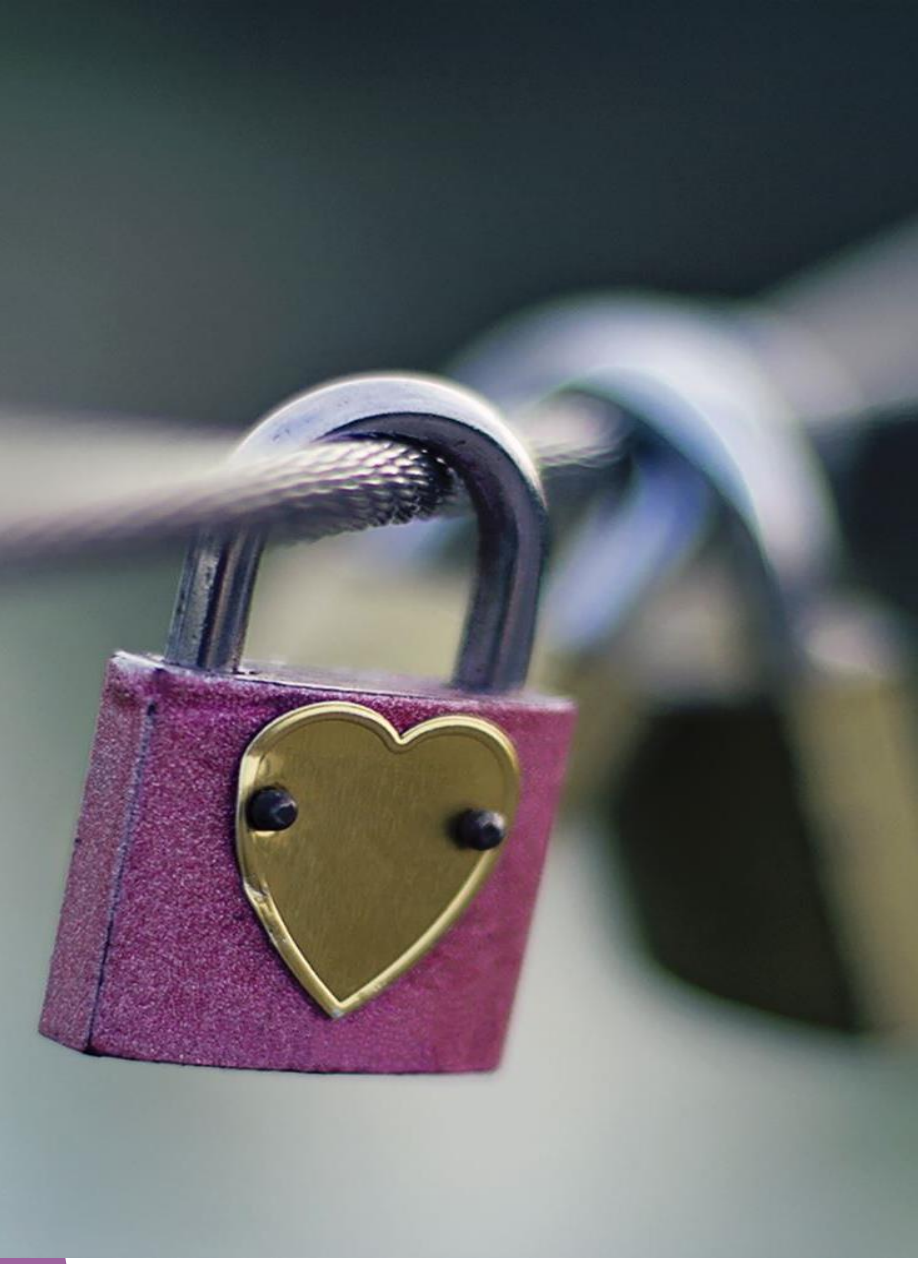



```
<script>
    function fetchData() {
        console.log("Fetching updated data.");
        const productId = document.getElementById("reviews").dataset.productId;
        const xhr = new XMLHttpRequest();
        xhr.open("GET", "/reviews/" + productId, true);
        xhr.onload = () => {
            cleanData = DOMPurify.sanitize(xhr.responseText,{
                USE_PROFILES: {html: true},
                FORBID_ATTR: ['style', 'class','id']
            });

            document.getElementById("reviews").innerHTML = cleanData;
        };
        xhr.send();
    }

    fetchData()
</script>
```

[redacted]@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS



Key Takeaways

- Incidents Happen – Design to improve incident response.
- Prevent Regression – Never waste a good incident. Write tests!
- Support Extension and Exemption – Exceptions will be needed, design for them.
- Understand your libraries and compensating controls

@forced_request

john@cloudsecuritypartners.com

CLOUD SECURITY PARTNERS