

We will use Postman for API penetration testing.

→ just play around with Postman to get familiar with.

* Basic authentication

You can authenticate your self through header request

Authorization: Basic username:password

But you will get bad request 400, because you did not encode your credentials with base64, so you have to encode them to be authorized. After that send your request and it will login successfully.

Use apimatic.io to transfer API?

How to search?

Google → Site:target.com api: postman

Site:com /swagger/index.html

Site:com /v2/api-docs.json

Site:com /v2/api-docs

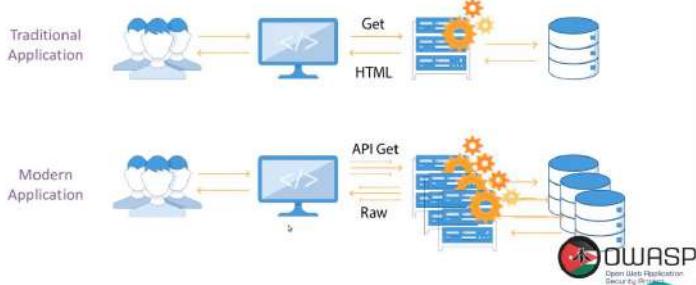
websites

- * jsonplaceholder.typicode.com/users
- * postman-echo.com/basic-auth
- * apidocs.imgur.com
- * rest.vulnweb.com/v1/api/users/
- * webscantest.com

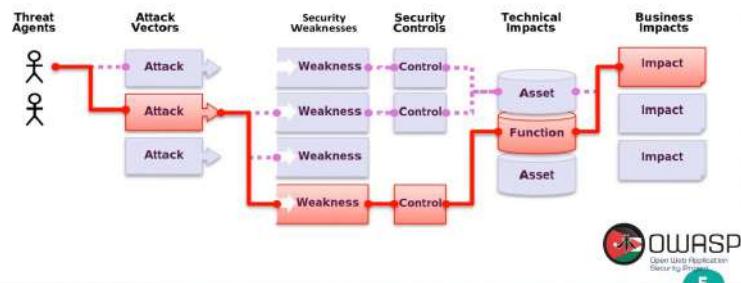
Lab Setup

1. <https://github.com/tkisason/vulnapi>
2. <https://github.com/snoopysecurity/dwes-node>
3. <https://github.com/rev0s/VAmPI/>
4. <https://github.com/jorritfolmer/vulnerable-api/>
5. <https://github.com/InsiderPhD/Generic-University>

Web application security vs API security



What Are Application Risks?

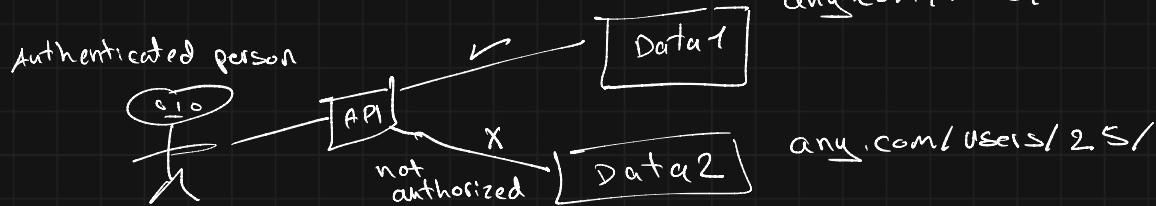


OWASP API Top 10 2019

- A1 : Broken object level authorization
- A2 : Broken authentication
- A3 : Excessive data exposure
- A4 : Lack of resources and rate limiting
- A5 : Broken function level authorization
- A6 : Mass assignment
- A7 : Security misconfiguration
- A8 : Injection
- A9 : Improper assets management
- A10 : Insufficient logging and monitoring

OWASP API Top 10

1- Broken Object level Authorization



If user access something he is not allow to access, then it is vulnerability.

How to find?

- ① Create two accounts
- ② Capture request with identifier
- ③ Replace the authentication token account1 with account2
- ④ If you are able to make request, then there is broken object level authorization.

A1 : Broken object level authorization



- APIs consume a lot of object IDs by design:
 - URL params (/api/users/717) / Query Params (/download_file? id=111)
 - Body params / HTTP Headers (user-id:717)
- Known also as:
 - IDOR
 - Forceful Browsing
 - Parameter Tampering
 - Broken Authorization



8

T-Mobile API Breach (2017)

The screenshot shows a JSON response from a T-Mobile API endpoint. The 'user_id' field is highlighted with a blue arrow. The JSON structure includes fields such as 'id', 'user_id', 'name', 'phone', and others. A link to a YouTube video is provided at the bottom: https://www.youtube.com/watch?v=3_gd3a077RU.

Prevention

- Implement authorization checks with user policies and hierarchy.
- Do not rely on IDs that the client sends. Use IDs stored in the session object instead.
- Check authorization for each client request to access database.
- Use random IDs that cannot be guessed (UUIDs).

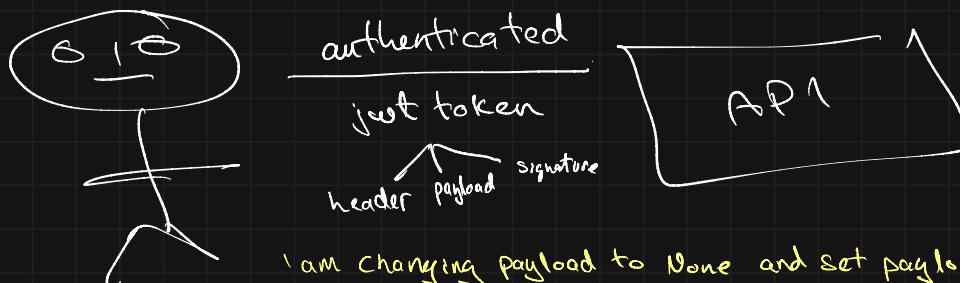


10

This is very dangerous. Attacker can write a program in (for loop to get all possible numbers from 0---- to 9----). All users has been exposed.

2- Broken Authentication

- Authentication mechanism for clients and end-user are sometimes not properly specified and implemented.
- Attacker can use that flawed implementations to compromise authentication token and assume identity of another user.
- if the system ability to identify the client is compromised the overall api security is compromised.



I am changing payload to None and set payload to another user (user id), then send jwt and if I am able to authorized, it is vulnerability.

How to find?

- if there is no brute-force protection, report it
- if there is no account lockout mechanism
- weak password → /api/v/register and they accepting 1234, then there is weak password.
- sensitive data/token in url
- Does not validate token.
- Accept (unsigned token or jwt non attack)
- use of plaintext, weak hashes password
- use of weak encryption keys / API keys.

These checklist what I will look for: impact: attacker can take over another accounts, sensitive data.

A2 : Broken authentication



- Unprotected APIs that are considered "internal"
- Weak authentication that does not follow industry best practices
- Weak API keys that are not rotated
- Passwords that are weak, plain text, encrypted, poorly hashed, shared, or default passwords
- Authentication susceptible to brute force attacks and credential stuffing
- Credentials and keys included in URLs
- Lack of access token validation (including JWT validation)
- Unsigned or weakly signed non-expiring JWTs



Balboa hot tubs (2018) example



<https://www.youtube.com/watch?v=SyL3zG1FtKg>

<https://www.pentestpartners.com/security-blog/hackers-in-hot-water-pwning-smart-hot-tubs-yes-really/>



- The APIs are "protected" with a shared hardcoded password.
- The mobile app controlling the hot tub is invoking the hot tub's API over the internet.
- Each hot tub comes with an unprotected WiFi hotspot.
- The APIs use the WiFi hotspot ID as the device identifier.
- Attackers can use WiFi hotspot directories to locate all Balboa hot tubs. This gives them both the ID and the location of a hot tub. After that, they can invoke the APIs of the hot tub, know when it is in use, and take over control.



13

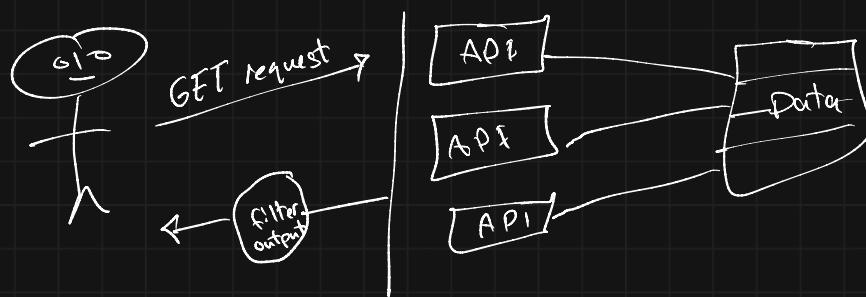
Prevention

- Check all possible ways to authenticate to all APIs.
- APIs for password reset and one-time links also allow users to authenticate, and should be protected just as rigorously.
- Use standard authentication, token generation, password storage, and multi-factor authentication (MFA).
- Use short-lived access tokens.
- Authenticate your apps (so you know who is talking to you).
- Use stricter rate-limiting for authentication, and implement lockout policies and weak password checks.



14

3 - Excessive Data Exposure



Basically api returns full data object as they are stored in background database.

The client application filter the data response and only show that data that need to see.
Attacker can call directly api and bypass the client application and he will get easily the full information. That called **excessive data exposure**.

The problem is that api is relying on client application to do data filtering.

How to hunt's

- ① Does your api call return either sensitive data or full data object stored in db
- ② is returned content filtered on the client side before presented to the user?
↳ if yes, there is a strong chance to be vulnerable.

A3 : Excessive data exposure



- The API returns full data objects as they are stored in the backend database.
- The client application filters the responses and only shows the data that the users really need to see.
- Attackers call the API directly and get also the sensitive data that the UI would filter out.



15

Uber account takeover (2019) example



- Error message leaking user UUID

Request

```
POST /v3/fleet-manager/_rpc?rpc=addDriverV2 HTTP/1.1
Host: partners.uber.com
{"nationalPhoneNumber": "99999xxxxx", "countryCode": "1"}
```

Response

```
{
  "status": "failure",
  "data": {
    "code": 1009,
    "message": "Driver '47d063f8-0xx5e-xxxx-b01a-xxxx' not found"
  }
}
```

- Make a request with the UUID

Request

```
POST /marketplace/_rpc?rpc=getConsentScreenDetails HTTP/1.1
Host: bonjour.uber.com
Connection: close
Content-Length: 67
Accept: application/json
Origin: [https://bonjour.uber.com](https://bonjour.uber.com)
x-csrf-token: xxxx
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36
DNT: 1
Content-Type: application/json
Accept-Encoding: gzip, deflate
Accept-Language: en-US;q=0.9
Cookie: xxxx
{"language": "en", "userUuid": "xxxx-776-4xxx1bd-B61a-837xx604ce"}
```

- Receive tons of user data including mobile app authentication token

Response
(373 lines!!!)

```
Response (373 lines!!!)
{
  "id": "xxxx-776-4xxx1bd-B61a-837xx604ce",
  "name": "xxxx-776-4xxx1bd-B61a-837xx604ce",
  "email": "xxxx-776-4xxx1bd-B61a-837xx604ce@uber.com",
  "phone": "+1 (555) 123-4567",
  "address": "123 Main St, Anytown, USA",
  "city": "Anytown",
  "state": "CA",
  "zip": "90210",
  "lat": 34.052235,
  "lon": -118.243683,
  "last_login": "2020-01-01T12:00:00Z",
  "last_logout": "2020-01-01T12:00:00Z",
  "last_ip": "192.168.1.100",
  "last_device": "iPhone X (iOS 13.3.1)",
  "last_browser": "Safari (Version 13.1.2)",
```

Prevention

- Never rely on the client to filter data!
- Review all API responses and adapt them to match what the API consumers really need.
- Carefully define schemas for all the API responses.
- Do not forget about error responses, define proper schemas as well.
- Identify all the sensitive data or Personally Identifiable Information (PII), and justify its use.
- Enforce response checks to prevent accidental leaks of data or exceptions.



19

4- lack of resources and rate limiting:

leak of resource scenario :

what if attacker upload a large image by using a post request to api . /api/d/images , the available memory become exhausted / unresponsive -

Scenario 2 :

large number of resource request

We have an application that can send a list of users

/api/user?page=1 & size = 100

so attacker change size parameter to 20000. This will cause a performance issue on database, so api may get unresponsive . → DDOS attack / buffer overflow error.

rate limit scenario (Brute force attack) :

For authentication user + password and two factor authentication that require OTP. on OTP there is no limit request from the user, so you can send request multiple times like 100 time and more.

What is the use case ?

1. Attacker overload the api by sending more request than it can handle .
2. Attacker send request at a rate exceeding the API's processing speed, clogging it up.
3. The size of request or some fields in them exceed what the API's can process
4. Zip bombs archive that have been designed so that unpacking them takes excessive amount of resources and overloads the API .

A4: Lack of Resources & Rate Limiting



• Attackers overload the API by sending more requests than it can handle.
• Attackers send requests at a rate exceeding the API's processing speed, clogging it up.
• The size of the requests or some fields in them exceed what the API can process.
• "Zip bombs", archive files that have been designed so that unpacking them takes excessive amount of resources and overloads the API.

OWASP
Open Web Application Security Project
20

Instagram account takeover (2019)

Used password reset API

Passcode expires in 10 minutes

Rate limiting 200 / minute / IP

5000 cloud VMs can enumerate all 6-digit codes

AWS/GCP cost ~ \$150

<https://www.youtube.com/watch?v=4O9FjTMiHUM>

Requesting passcode
POST /api/v1/users/lookup/
Host: i.instagram.com
q=mobile_number&
device_id=android-device-id

Verify passcode
POST /api/v1/accounts/
/account_recovery_code_verify/
Host: i.instagram.com
recover_code=123456&
device_id=android-device-id



21

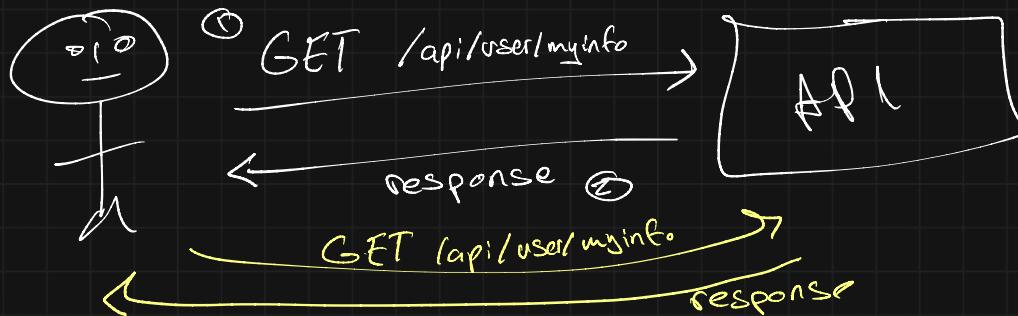
Prevention

- Define proper rate limiting.
- Limit payload sizes.
- Tailor the rate limiting to be match what API methods, clients, or addresses need or should be allowed to get.
- Add checks on compression ratios.
- Define limits for container resources.



22

5-Broken Function level Authorization



what if user request `GET /api/admin/allinfo` and get response
then it is vulnerable. just play with the url.

or

you are making post request `/api/user/register` and there is a parameter `is_admin=false`, so change it to `is_admin=true` and send request.

How to test?

you need to understand deeply the authorization hierarchy, so ask the following questions

1. Can a regular user access administrative endpoints?
2. Can a user perform sensitive actions. (creation, modification... by change http method
for example from get to delete)
3. Can a user from group X access a function that should be exposed only to users
from group Y by simply guessing the endpoint URL and parameter.

A5: Broken Function Level Authorization

- The API relies on the client to use user level or admin level APIs as appropriate. Attackers figure out the "hidden" admin API methods and invoke them directly.
 - Some administrative functions are exposed as APIs.
 - Non-privileged users can access these functions without authorization if they know how.
 - Can be a matter of knowing the URL, or using a different verb or a parameter:
 - `/api/users/v1/user/myinfo`
 - `/api/admins/v1/users/all`

23

Gator kids smartwatches (2019)

- Simple request of `User[Grade]` value from 1 to 0 enabled management of all smartwatches

<https://www.pentestpartners.com/security-blog/gps-watch-issues-again/>

Type	Name	Value
URL		seconduser/profile
Cookie	_cf	e432ab101199a43690ca7329145a9fe444c4b6
Cookie	PHPSESSID	d3wqj0leu68t0jaq1cpusd
Body	_cf	HTTP2/2a2chBcTh02aUA09kQAFdp9fR56eBN
Body	User[cfid]	7a37eb1fb5-11e9-aefc-0af0-0a01080
Body	User[Grade]	1
Body	User[companyID]	0070A0BE-7168-43C3-6A41-C629C3F4DCF
Body	User[nickName]	egn2
Body	User[business]	05CD69A2-4C02-42EB-8351-40190301
Body	User[zipAddress]	
Body	User[zipCity]	
Body	User[zip]	
Body	User[Email]	
Body	User[dateFormat]	
Body	User[dateformat]	

24

Prevention

- Do not rely on the client to enforce admin access.
- Deny all access by default.
- Only allow operations to users belonging to the appropriate group or role.
- Properly design and test authorization.



F - Security Misconfiguration

Try to look for CORS | HTTP VERB issue | Errors | unprotected files + dir's
unpatched hosts (Apache Struts) | common endpoints

Scenario :-

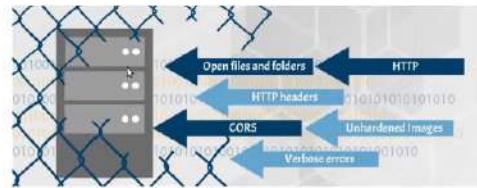
let say you found a bash history file contains the following command:
`curl -X GET 'endpoint url' -H 'authorization token'`

What to look for :-

- unpatched system
- unprotected files and directories
- unhardened images
- missing outdated or misconfigured TLS
- exposed storage or server management panel
- Missing CORS policy or security headers
- Error messages with stack traces
- unnecessary features enabled

A7: Security Misconfiguration

- Poor configuration of the API servers allows attackers to exploit them.
 - Unpatched systems
 - Unprotected files and directories
 - Unhardened images
 - Missing, outdated, or misconfigured TLS
 - Exposed storage or server management panels
 - Missing CORS policy or security headers
 - Error messages with stack traces
 - Unnecessary features enabled



29

Equifax



- In 2017, Equifax was breached and private information of 147 million people got exposed. The breach started with unpatched Apache Struts exploit. The system was not enforcing formats on incoming API calls and specifically crafted Content-Type header allowed attackers to get in.

<https://republicans-oversight.house.gov/wp-content/uploads/2018/12/Equifax-Report.pdf>



30

Prevention

- Establish repeatable hardening and patching processes.
- Automate locating configuration flaws.
- Disable unnecessary features.
- Restrict administrative access.
- Define and enforce all outputs, including errors.



31

B - injection

→ GET /api/users/username/info

→ POST /api/accounts/recovery
username = Ibrahim'

→ /api/config/restore
app_id = 123

↳ I will try to inject commands or SQL injection

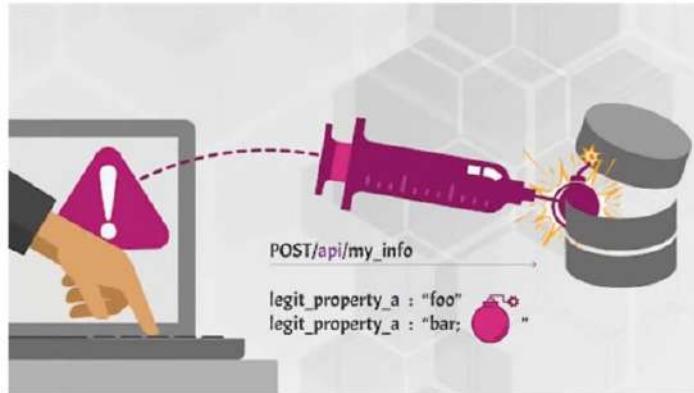
inject query (SQL injection)



A8: Injection



- Attackers construct API calls that include SQL, NoSQL, LDAP, OS, or other commands that the API or the backend behind it blindly executes.



32

Samsung SmartThings Hub (2018)



- SmartThings Hub is able to communicate with cameras
- credentials API can be used to set credentials for remote server authentication
- Data is saved in SQLite database
- JSON keys with ; allowed to execute arbitrary queries

```
# using curl from inside the hub, but the same request could be sent using a SmartApp
$ sInj='";_id=0 where 1=2;insert into camera values (123,replace(substr(quote(zeroblob((10000 + 1) / 2)), 3, 10000), \'\"A\\\"'),1,1,1,1,1,1,1,1,1,1,1,1,1,1);--;"'
$ curl -X POST 'http://127.0.0.1:3000/credentials' -d
"{'s3:{'accessKey':'','secretKey':'','directory':'','region':'','bucket':'','sessionToken':'${sInj}'},'
videoHostUrl:'127.0.0.1/'}"
$ curl -X DELETE "http://127.0.0.1:3000/cameras/123"
```

https://talosintelligence.com/vulnerability_reports/TALOS-2018-0556/



33

Prevention

- Never trust your API consumers, even if internal
- Strictly define all input data: schemas, types, string patterns -and enforce them at runtime
- Validate, filter, sanitize all incoming data
- Define, limit, and enforce API outputs to prevent data leaks



34

g- Improper assets management

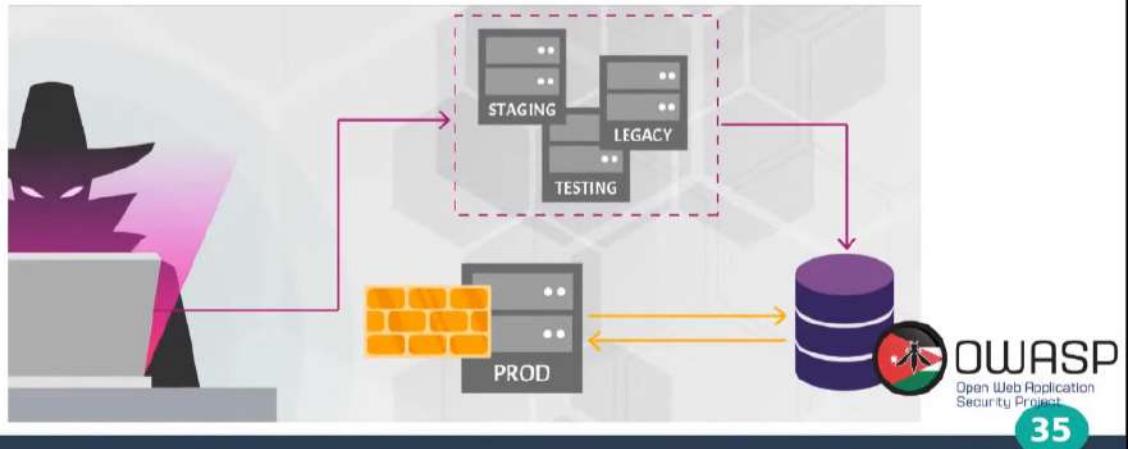
- purpose of an API host is unclear and there are no explicit answers to the following:

- Documentation is not updated.
 - No retirement plan for api version.
 - Host inventory is missing or outdated.
 - old or previous api versions are running unpatched.

A9: Improper Assets Management



- Attackers find non-production versions of the API (for example, staging, testing, beta, or earlier versions) that are not as well protected as the production API, and use those to launch their attacks.



JustDial(2019)



Example

* India's largest local search service

- Had old unused unprotected API
 - It was connected to live database
 - Was leaking 100 mln users' data

```
- data:  
    education: "",  
    name: "Kashishan",  
    name_1: "Kashishan",  
    name_2: "Kashishan",  
    gender: "Male",  
    birthday: "1998-01-01",  
    city: "",  
    mobile: "+919876543210",  
    login: "test@kashishan@gmail.com",  
    image: "https://www.profilebutton.com/  
    privacy: "",  
    company: "",  
    occupation: "Businessman",  
    language: "",  
    CMobile: "+919876543210",  
    talktime: "00:00:00",  
    businessname: [""],  
    Pmobile: "+919876543210"  
},  
    MRating: 0,  
    Joe: 0,  
    planid: 1,  
    planname: "J",  
    meterstatus: "On Status"
```

Tchap



Prevention

- Keep an up-to-date inventory all API hosts.
 - Limit access to anything that should not be public.
 - Limit access to production data, and segregate access to production and non-production data.
 - Implement additional external controls, such as API firewalls.
 - Properly retire old versions of APIs or backport security fixes to them.
 - Implement strict authentication, redirects, CORS, and so forth.

- Tchap is a messaging app that the French government released for internal use. It was hailed as a more secure replacement for Telegram and WhatsApp.
 - The sign-up API had an email address parameter that didn't validate the input format.
 - A security researcher, Elliot Alderson, submitted `fs0c131y@protonmail.com@elysee.fr` as the address.
 - The code simply verified that the address ended with `@elysee.fr`, which is a government email domain.
 - The check was successful, and Tchap sent a verification email that got delivered to `fs0c131y@protonmail.com` belonging to the attacker.
 - The attacker could click the confirmation link, get in, and be able to get into government chat rooms.

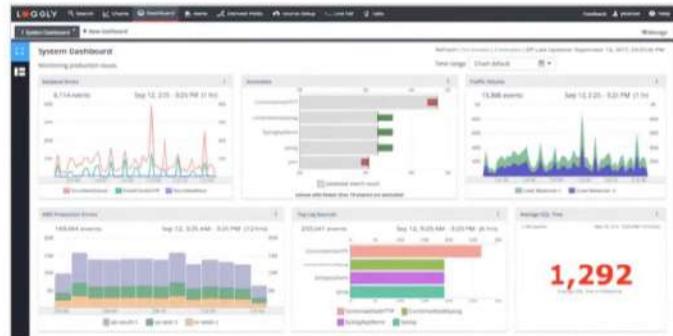


10 - Insufficient Logging & Monitoring

A10: Insufficient Logging & Monitoring



- You can't react to attacks that you don't know about.
- Logs are important for:
 - Detecting incidents
 - Understanding what happened
 - Proving who did something



39

7-Eleven Japan: 7Pay

- Payment app
- Password reset allowed email change if personal details were supplied
- Attackers used API to try combinations of info for various Japanese citizens
- \$510K was stolen from 900 customers
- The company only noticed after too many users complained

if they had monitoring system,
this will not happen.

Prevention

- Log failed attempts, denied access, input validation failures, any failures in security policy checks
- Ensure that logs are formatted to be consumable by other tools
- Protect logs as highly sensitive
- Include enough detail to identify attackers
- Avoid having sensitive data in logs -If you need the information for debugging purposes, redact it partially.
- Integrate with SIEMs and other dashboards, monitoring, alerting tools



41

