GLITCHSECURE

Generated on 02 August 2023
Valid until 14 May 2024

In-depth Pentest Report

Prepared for Acme Co

Prepared by

- Jade Null
- Dirk Nyhof
- Brad Bahls

Table of Contents

Introduction

Executive Summary

Attack Narrative

- Reconnaissance & Information Gathering
- Identifying Technologies Used
- Port Scanning & Service Identification
- Subdomain Enumeration
- Directory & File Enumeration
- Automated Scanning
- Manual Testing

Findings

- Asset List
- Vulnerability & Findings List
- Vulnerability & Findings Details
  - #1041_1 - Account Compromise Through Password Reuse
  - #1041_2 - Email MFA bypass
  - #1041_3 - Username Enumeration
  - #1041_4 - Web Application Firewall Bypass
  - #1041_5 - Privilege Escalation

Conclusion & General Comments

- Document Change Log
- Completed Test Cases

Introduction

Acme Co contracted GlitchSecure to conduct a In-depth Pentest of 3 externally facing web assets. The evaluation, which formally began on 01 May 2023, and concluded on 15 May 2023, aimed to thoroughly assess the security posture of the organisation's online presence.

Please note that the following report has been specifically prepared for demonstration and illustrative purposes. Consequently, it may not provide an exhaustive account of the technical particulars typically found within an authentic security assessment report.

Executive Summary

The primary objective of the evaluation was to identify potential security vulnerabilities and issues within a specific subset of the Acme Co infrastructure, with the aim of safeguarding the security and privacy of its users and overall system. To achieve this, a black box penetration test was performed, simulating the actions and strategies of a real-world adversary. By adopting this approach, the GlitchSecure team sought to gain comprehensive insights into defence mechanisms and identify any exploitable weaknesses.

All assessment activities were conducted in a manner that simulated an external malicious actor engaged in a targeted attack. The ultimate goal was to identify and exploit any existing security weaknesses that could allow a remote attacker to gain unauthorised access to organisational and customer data and systems. This assessment adhered to the recommendations and industry best practices outlined by The Open Web Application Security Project (OWASP). OWASP's framework covers various aspects, including but not limited to: input validation, session management, encryption, error handling, and secure coding practices.

Attack Narrative

Reconnaissance & Information Gathering

As with typical black box assessments, Acme Co provided minimal information regarding the existing infrastructure and technologies employed. This approach aimed to closely replicate a real-world attack where external actors lack internal knowledge. Information gathered about the targets came from a variety of sources and focused on identifying the software utilised, discovering open ports and services, and conducting file and directory enumeration.

Identifying Technologies Used

Fingerprinting was performed using a combination of manual source code review and Open Source tools.

Port Scanning & Service Identification

Port scanning was performed on all hosts within the scope with scanning covering a port range of 1-65535 across TCP and UDP.

The follow targets showed open ports:

- app.acme.tld port 443

Subdomain Enumeration

Throughout the course of the engagement, one additional subdomain was discovered. The following subdomains were found to be active, and within scope for this assessment:

- app.acme.tld
- app-old.acme.tld

Directory & File Enumeration

Directory enumeration was performed on all assets using predefined and customised word lists to help expand the scope and identify potentially sensitive information and additional targets.

Automated Scanning

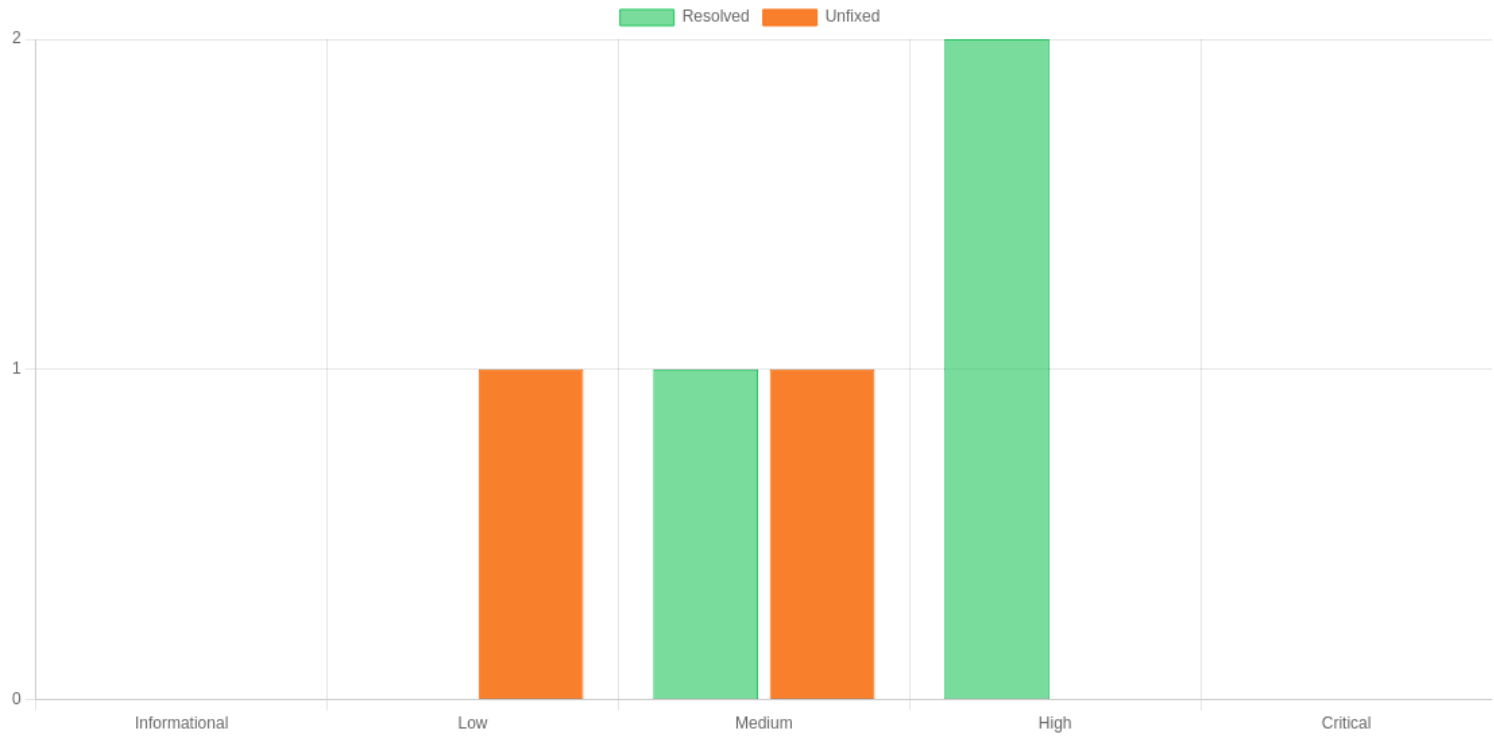
Several automated scanning and testing tools were deployed on the previously found hosts to help ensure any potential findings were not missed. Results from automated tools were then processed and manually reviewed and tested to confirm the accuracy of the findings.

Manual Testing

Manual testing was carefully conducted using various methodologies including those outlined in the OWASP Application Security Verification Standard (ASVS).

Findings

During the assessment, a total of 5 issues were identified. Of the findings, 2 are of high severity, 2 are of medium severity, 1 is of low severity, .



Asset List

This table presents an overview of the assets that were targeted during this assessment, along with the number of corresponding findings.

Asset #	Asset Location	Asset Type	Environment	Findings
ym3sai_1	app.acme.tld	DOMAIN	Production	5
ym3sai_2	13.55.55.37	IP ADDRESS	Production	1
ym3sai_4	10.13.37.1	IP ADDRESS	Development	0

Vulnerability & Findings List

The following lists contains summary information of vulnerabilities and findings identified during the assessment. Corresponding technical details can be found in the Vulnerability & Findings Details section.

Affected Asset	Finding	Severity
app.acme.tld	Account Compromise Through Password Reuse	HIGH
app.acme.tld	Email MFA bypass	HIGH
app.acme.tld	Username Enumeration	LOW
13.55.55.37, app.acme.tld	Web Application Firewall Bypass	MEDIUM
app.acme.tld	Privilege Escalation	MEDIUM

Vulnerability & Findings Details

Account Compromise Through Password Reuse

#1041_1 Reported by Jade Null

high resolved

Category:

Sensitive Data Exposure -> Disclosure of Secrets

CWE(s):

CWE-654: Reliance on a Single Factor in a Security Decision CWE-308: Use of Single-factor Authentication

CVSS 3.1 Base Score:

8.2 **(High) -** CVSS3.1/AV:C/AC:V/PR:S/UI:S/S:3/C:1/I:A/A:V

Affected Assets

`app.acme.tld`

Overview

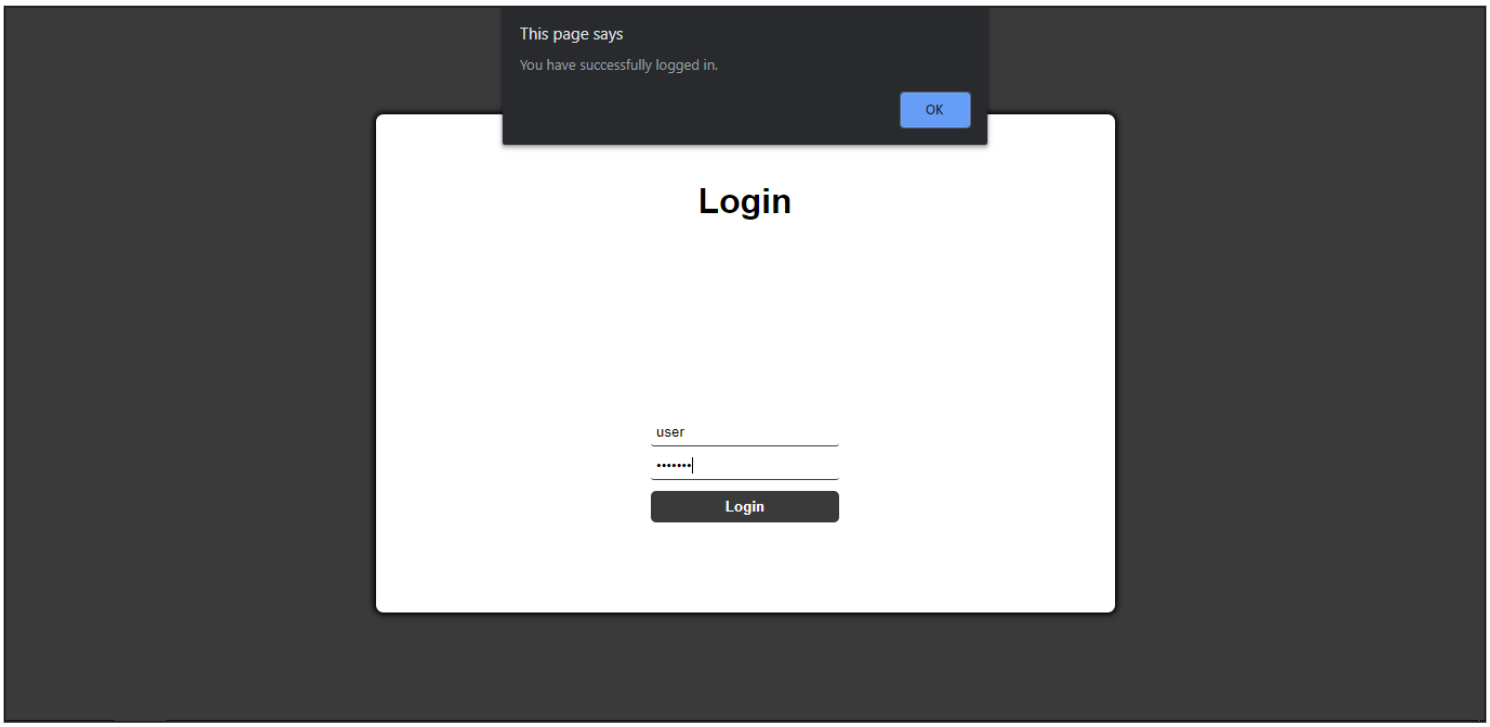
During testing, it was found that several employees at Acme Co were previously impacted by third-party data breaches that publicly exposed their plaintext credentials on the dark web. The GlitchSecure team analysed this publicly available breach data and utilised the information found to perform a targeted password stuffing attacks on the Acme web application. In doing so, the team successfully compromised a director-level employee account.

Technical Details

The GlitchSecure team searched through publicly available breach data and identified the following plaintext email/password combinations:

Username	Password
laura@acme.tld	hunter2
gary@acme.tld	leosatec9
stefan@acme.tld	stefan36
stella@acme.tld	1sammyy
stella@acme.tld	stelstar100

Using the list above, the team performed a password stuffing attack against `app.acme.tld` using various permutations of each password.



As demonstrated in this screenshot, the team was able to successfully login to the account of `laura@acme.tld` using the credentials noted in the table above.

Severity Detail

Due to the increased access level of the compromised account, the impact to confidentiality is significant. The compromised user is a director-level employee at Acme Co and had access to a large number of potentially confidential Acme customer names and information.

Remediation Steps

- Immediately reset the affected user's password.
- Investigate logs for signs of suspicious successful login activity.
- Implement internal employee training to advise against password reuse and encourage the use of password managers.
- Implement mandatory MFA for all privileged employee accounts.
- Implement login notifications when user accounts are accessed from a new device or IP.

References

- OWASP Credential Stuffing
- OWASP Credential Stuffing Prevention Cheat Sheet
- Have I Been Pwned?
- OWASP Multifactor Authentication Cheat Sheet
- Laura Person, Director at Acme Co (LinkedIn)

Email MFA bypass

#1041_2 Reported by Dirk Nyhof

high resolved

Category:

Broken Authentication and Session Management -> Second Factor Authentication (2FA) Bypass

CWE(s):

CWE-302: Authentication Bypass by Assumed-Immutable Data

CVSS 3.1 Base Score:

7.1 **(High) -** CVSS3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:L/A:N

Affected Assets

`app.acme.tld`

Affected Locations

`/api/2fa`

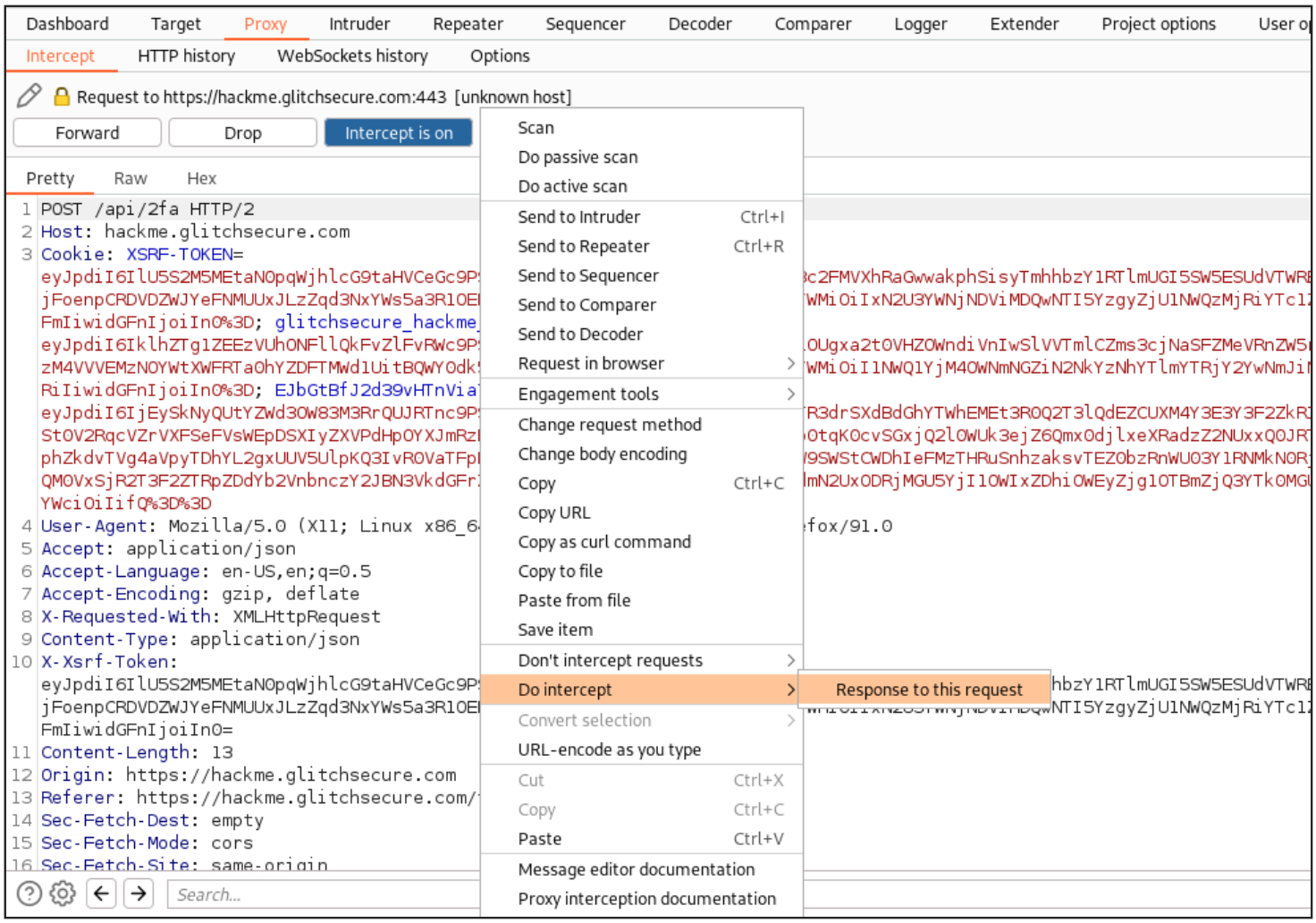
Overview

During the testing, it was discovered that it's possible to bypass email MFA and login to any account without confirming the email verification code.
Additional testing of the TOTP-based MFA flow revealed this that only accounts utilising email based MFA were affected.

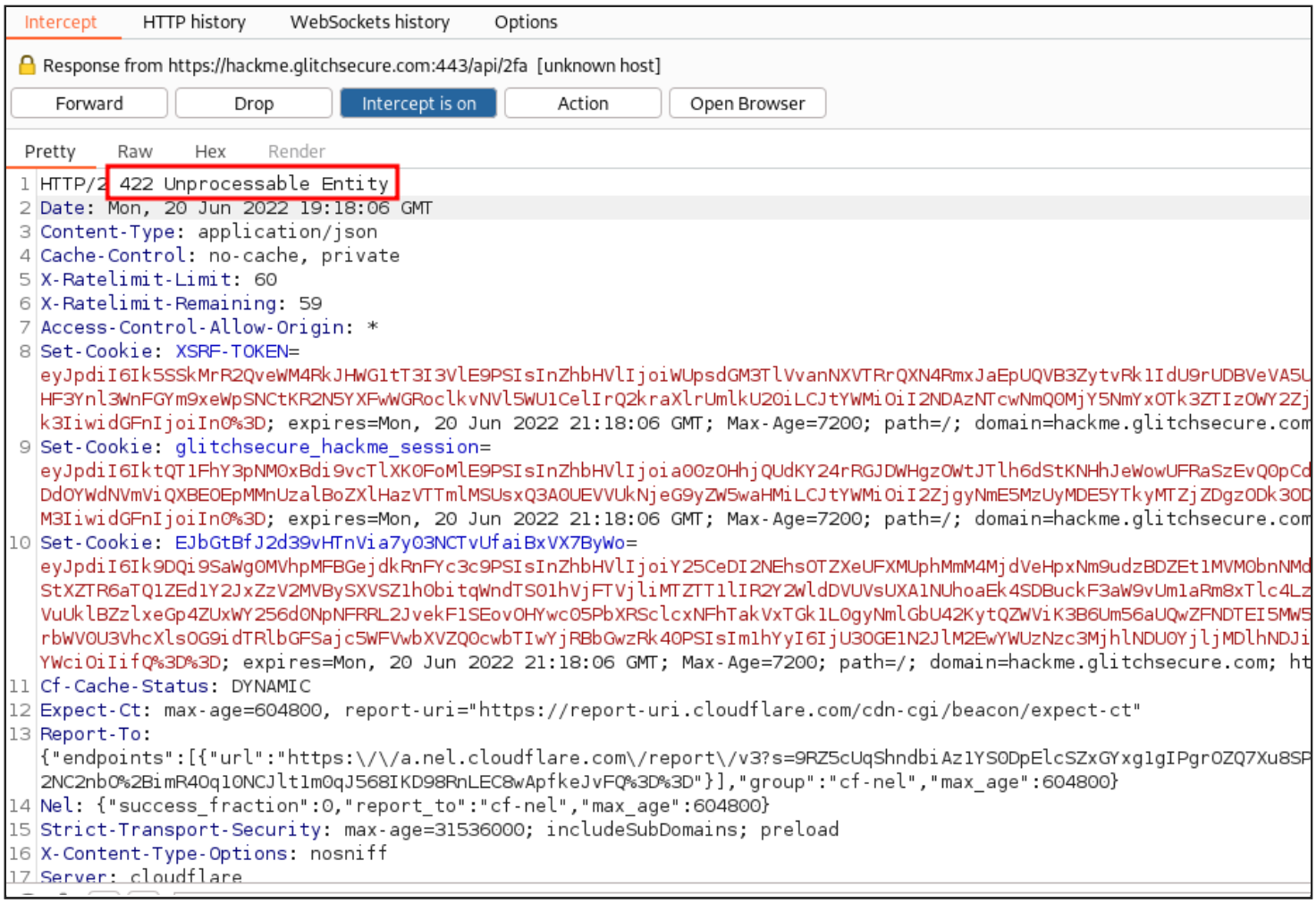
Technical Details

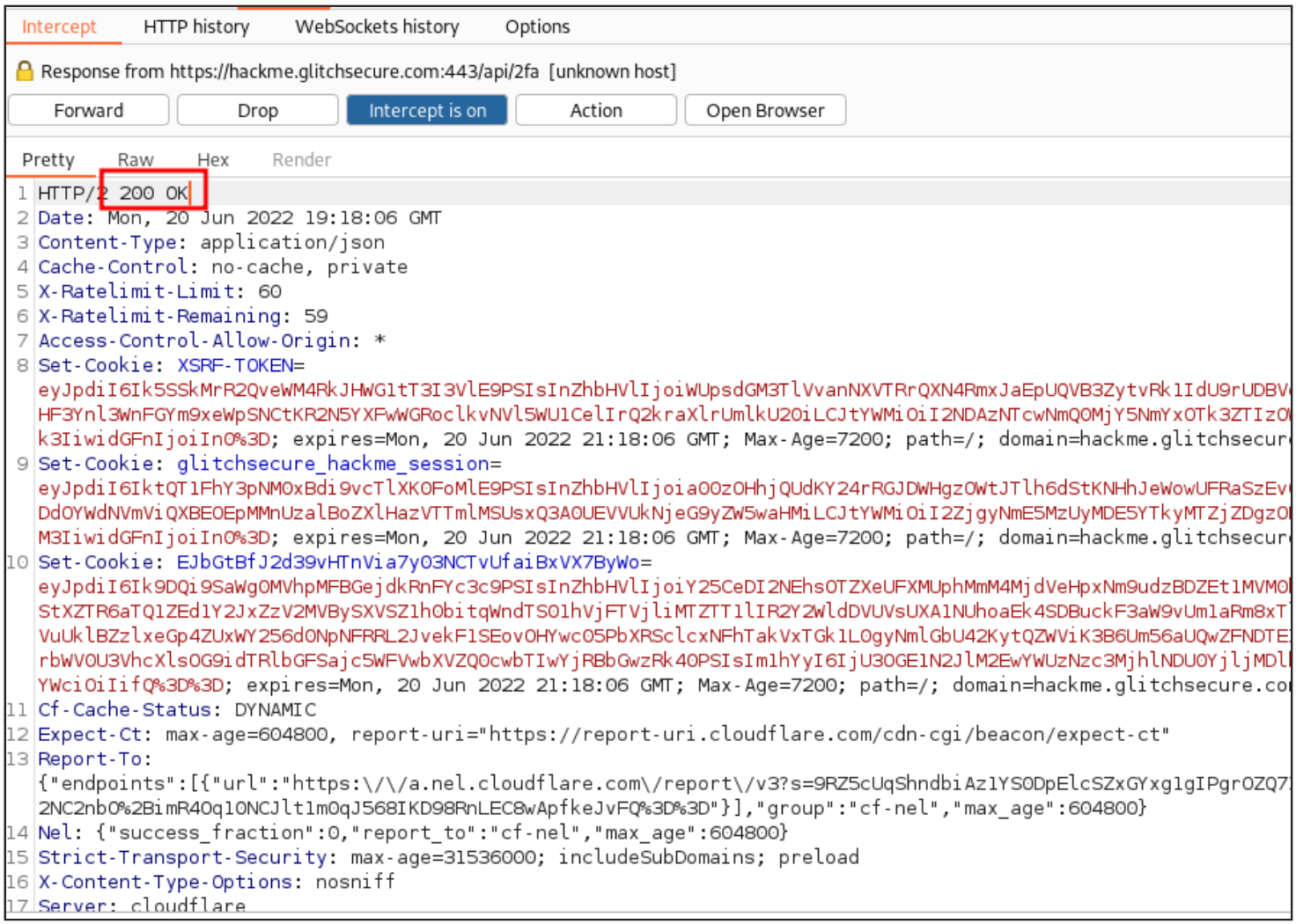
During the testing, the GlitchSecure team observed that when confirming the MFA email code, a POST request is sent to the `/api/mfa` endpoint, which is different from the endpoint used for TOTP MFA.
To reproduce this bug follow the steps:

- Login to an account that has email MFA enabled.
- Setup the proxy tool to intercept the request and submit the random MFA code.



- Intercept the response on the MFA submit request and change the response status code of `422 Unprocessable Entity` to the `200 OK`.


Intercept
HTTP history
WebSockets history
Options
Response from https://hackme.glitchsecure.com:443/api/2fa [unknown host]
Forward
Drop
Intercept is on
Action
Open Browser
Pretty
Raw
Hex
Render
1 HTTP/2 422 Unprocessable Entity
2 Date: Mon, 20 Jun 2022 19:18:06 GMT
3 Content-Type: application/json
4 Cache-Control: no-cache, private
5 X-Ratelimit-Limit: 60
6 X-Ratelimit-Remaining: 59
7 Access-Control-Allow-Origin: *
8 Set-Cookie: XSRF-TOKEN=
eyJpdiI6Ik5SSkMrR2QveWM4RkJHWG1tT3I3VlE9PSIsInZhbHVlIjoiWUpsdGM3TlVvanNXVTRrQXN4RmxJaEpUQVB3ZytvRk1IdU9rUDBVeVA5U
HF3Ynl3WnFGYm9xeWpSNCtKR2N5YXFwWGRoclkvNVl5WU1CelIrQ2kraXlrUmlkU20iLCJtYWMiOiI2NDAzNTcwNmQ0MjY5NmYxOTk3ZTIzOWY2Zj
k3IiwidGFnIjoiIn0%3D; expires=Mon, 20 Jun 2022 21:18:06 GMT; Max-Age=7200; path=/; domain=hackme.glitchsecure.com
9 Set-Cookie: glitchsecure_hackme_session=
eyJpdiI6IktQT1FhY3pNM0xBdi9vcTlXK0FoMlE9PSIsInZhbHVlIjoia00z0HhjQUdKY24rRGJDWHgz0WtJTlh6dStKNHhJeWowUFRaSzEvQ0pCd
DdOYWdNVmViQXBE0EpMMnUzalBoZXlHazVTTmlMSUsxQ3A0UEVVUkNjeG9yZW5waHMiLCJtYWMiOiI2ZjgyNmE5MzUyMDE5YTkyMTZjZDgz0Dk30D
M3IiwidGFnIjoiIn0%3D; expires=Mon, 20 Jun 2022 21:18:06 GMT; Max-Age=7200; path=/; domain=hackme.glitchsecure.com
10 Set-Cookie: EJbGtBfJ2d39vHTnVia7y03NCTvUfaiBxVX7ByWo=
eyJpdiI6Ik9DQi9SaWg0MVhpMFBGejdkRnFYc3c9PSIsInZhbHVlIjoiY25CeDI2NEhs0TZXeUFXMUphMmM4MjdVeHpxNm9udzBDZEt1MVM0bnNMd
StXZTR6aTQ1ZEd1Y2JxZzV2MVBySXVSZ1h0bitqWndTS01hVjFTVjliMTZTT1lIR2Y2WldDVUVsUXA1NUhoaEk4SDBuckF3aW9vUm1aRm8xTlc4Lz
VuUklBZzlxeGp4ZUxWY256d0NpNFRRL2JvekF1SEov0HYwc05PbXRSclcxNFhTakVxTGk1LOgyNmlGbU42KytQZWViK3B6Um56aUQwZFNDTEI5MW5
rbWV0U3VhcXlsOG9idTRlbGFSajc5WFVwbXVZQ0cwbTIwYjRBbGwzRk40PSIsIm1hYyI6IjU3OGE1N2JlM2EwYWUzNzc3MjhlNDU0YjljMDlhNDJi
YWciOiIifQ%3D%3D; expires=Mon, 20 Jun 2022 21:18:06 GMT; Max-Age=7200; path=/; domain=hackme.glitchsecure.com; ht
11 Cf-Cache-Status: DYNAMIC
12 Expect-Ct: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
13 Report-To:
{"endpoints":[{"url":"https:\/\/a.nel.cloudflare.com\/report\/v3?s=9RZ5cUqShndbiAz1YS0DpElcSZxGYxg1gIPgrOZQ7Xu8SP
2NC2nbO%2BimR4Oq10NCJlt1m0qJ568IKD98RnLEC8wApfkeJvFQ%3D%3D"}],"group":"cf-nel","max_age":604800}
14 Nel: {"success_fraction":0,"report_to":"cf-nel","max_age":604800}
15 Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
16 X-Content-Type-Options: nosniff
17 Server: cloudflare



- Forward the rest of the requests and note you are now fully logged into the account and MFA has been bypassed.

Severity Detail

Failure to validate the email MFA verification code can result in authentication bypass and result in account takeover as it defeats the last protection mechanism that guards the account.

Remediation Steps

Implement access control measures that prevents the users from using the application if email MFA has not been successfully validated.

Username Enumeration

#1041_3 Reported by Brad Bahls

low unfixed

Category:

Broken Access Control (BAC) -> Username/Email Enumeration

CWE(s):

CWE-200: Exposure of Sensitive Information to an Unauthorized Actor

CVSS 3.1 Base Score:

5.3 **(Medium) -** CVSS3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N

Affected Assets

`app.acme.tld`

Affected Locations

`/api/calendar/book?calendar=[ID]`

Overview

During testing it was found that some usernames within the target web application could be enumerated from a publicly accessible API endpoint using an insecure direct object reference (IDOR) attack. The information disclosed comprised of full names, usernames, and user IDs.

Technical Details

The Acme Co web application allows users to create public facing calendar scheduling pages. These pages are inherently public, however there is no central repository of this information.
The GlitchSecure team was able to utilise a publicly accessible API endpoint and submit a large number of `GET` requests with sequentially increasing IDs in the `calendar` parameter. This allowed the team to create a detailed list of all user calendars within the platform.
To demonstrate this, the team sent the following `GET` request with the `[ID]` parameters from 0 through 1000.
Request:

```
GET /api/calendar/book?calendar=[ID] HTTP/2
Host: cal.com
[...ommitted for brevity...]
```

As shown in the screenshot below, the team was able to compile a table with the information from the responses.

Request	Payload	Status	Error	Timeout	Length	"name":"	"username":"	"eventType":{"id":	,"description":"
48931	48931	200			7719	Luiz Mazini	luiz-mazini-nkah7q	48931	idashdiahdiksa.cac.a
58690	58690	200			7827	Carolina Kowanz	carokowanz	58690	lesson with Carlina.
234	234	200			230972	Johannes	johannes	234	let's be frens! ð¤
79271	79271	200			124033	Namaskar	namaskar	79271	let's catch up and hang ...
78233	78233	200			223907	Akhil K G	akhil	78233	let's catchup. No rigid a...
42077	42077	200			7790	Conrad Kramer	conradev	42077	let's chat
71803	71803	200			105125	jahed momand	jahed-momand	71803	let's chat about anythin...
46247	46247	200			145847	isaac	isaacxweb3	46247	let's chat!
76166	76166	200			374428	Kunal Desai	kunaljaydesai	76166	let's grab dinner! pleas...
76159	76159	200			374510	Kunal Desai	kunaljaydesai	76159	let's play tennis! please ...
32618	32618	200			271005	Thiago Avelino	avelino	32618	let's talk, see the best d...
14054	14054	200			7696	Kanan Rengaraju	kanan	14054	lets chat crypto and ev...
66056	66056	200			251087		-ngugi	66056	lets talk
45931	45931	200			7923	Matthew Cheney	mmlc	45931	letâs talk vegan food, c...
71094	71094	200			7935	project\	hjghghgghg	71094	lfslsmfms
12386	12386	200			7842	Rachide Ouattara	rachide	12386	ljhlhj
41907	41907	200			22723	Nixter Barbeque	nickowski	41907	lol version
59771	59771	200			7783	Bob	fzkbfjzebfl	59771	lorem ipsum
32179	32179	200			7634	aermike	aermike	32179	main
35693	35693	200			7807	Pearly	pearly	35693	medical check up
32817	32817	200			7913	Amaan	amaan	32817	meeting for crypto mar...
19831	19831	200			7942	Khalid Lam	eclipsegk	19831	meeting interview abo...
32816	32816	200			7838	Amaan	amaan	32816	meeting on discord wit...
14117	14117	200			7807	Senlima Sun	senlima	14117	meeting online
71404	71404	200			353922	Webstudio	thysultan	71404	meeting to discuss desi...
32822	32822	200			7908	Amaan	amaan	32822	meeting with BMA for ...
42121	42121	200			29482	Leonardo Ubbiali	leoubbiali	42121	meeting with Leo Ubbiali
71059	71059	200			7850	saugat neupane	saugat-neupane-uiq5dy	71059	meeting with my friends
51464	51464	200			7827			51464	mindworks nimmt die ...
22390	22390	200			7654	Mayank Prasad	mayank001	22390	mmmmm
41622	41622	200			459450	Pratul Kalia	pratul	41622	mobile release enginee...
26277	26277	200			7960	Mohammad Shahabadi	mohammadshahabadi	26277	mr.shadowkiller20@g...
227	227	200			7936	Chris	chrisgeorge	227	never be afraid that so...
77748	77748	200			7985	Sri Sathya Bayagani	sathya26	77748	new
44789	44789	200			114544	Victims Team	---img-src-x-onerror--al...	44789	new added by attacker
67110	67110	200			7843	New team Zydelo 2	devzydelo	67110	new zdelo meet 2
21444	21444	200			7566			21444	nnn
69010	69010	200			96126	Ohelo Studio	ohelo	69010	no idea what this is
38623	38623	200			317249	Belle	belle001	38623	nsa meet over coffee
38602	38602	200			317265	Belle	belle001	38602	nsa social meet over dr...
32495	32495	200			7631	Diksha	nega001	32495	nv

Severity Detail

While the information exposed was limited to public usernames and details, the issue highlights an unintended consequence of an IDOR vulnerability and the lack of rate limiting. A malicious attacker could utilise this vulnerability to compile a targeted list of known users of the Acme Co calendar service. This information could be further utilised in targeted attacks such as phishing and password stuffing.

Remediation Steps

- Replace the use of sequential ID numbers with strongly randomised UUIDs.
- Reduce the exposure of user controllable parameters when not needed.

References

OWASP: Insecure Direct Object Reference Prevention Cheat Sheet

Web Application Firewall Bypass

#1041_4 Reported by Dirk Nyhof

medium unfixed

Category:

Server Security Misconfiguration -> Web Application Firewall (WAF) Bypass

CWE(s):

CWE-693: Protection Mechanism Failure

CVSS 3.1 Base Score:

7.6 **(High) -** CVSS3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

Affected Assets

`13.55.55.37`
`app.acme.tld`

Overview

During testing, it was found that a subdomain in scope, which uses the Cloudflare Web Application Firewall for site protection, had it's IP addresses exposed via another subdomain discovered during the reconnaissance phase. By using the IP address discovered from the secondary subdomain, it became possible to establish a direct connection to the primary subdomain, bypassing the protection provided by the Web Application Firewall (WAF).

Technical Details

During testing, it was noted that `app.acme.tld` was utilising the Cloudflare web application firewall. Additionally, during reconnaissance, the team discovered the subdomain `app-old.acme.tld` which was noted to resolve to the IP address `13.55.55.37`.
Upon visiting `app-old.acme.tld` the user is redirect to `https://app.acme.tld/`

```
HTTP/2 302
server: nginx
date: Tue, 10 May 2023 18:27:53 GMT
content-type: text/html; charset=UTF-8
location: `https://app.acme.tld
x-httpd: 1
x-proxy-cache-info: DT:1
```

Using this information, the team attempted to connect to `app.acme.tld` through the context of the IP address disclosed, with the expectation that the underlying server would be the same and would accept the request.
To perform this bypass of the Cloudflare Web Application Firewall, the team added the following to our `host` file.

```
13.55.55.37   app.acme.tld
```

Upon doing this action, the team discovered that they were able to directly connect to the impacted site, bypassing any protection provided by the WAF.

Severity Detail

A Web Application Firewall (WAF) helps protect web applications by filtering and monitoring HTTP traffic between a web application and the Internet. It typically protects web applications from attacks such as cross-site forgery, cross-site-scripting (XSS), file inclusion, and SQL injection, among others.
The ability for an attacker to bypass the WAF means the protection provided is no longer in place, increasing the risk of serious exploitation and reducing the insight into potential attacks.

Remediation Steps

To resolve this issue, we recommend the following steps be taken.

- Disable the `app-old.acme.tld` DNS record that exposed the origin server's IP.
- Enable IP allowlisting to only accept traffic from the WAF's known IP addresses, in this case: Cloudflare.

Privilege Escalation

#1041_5 Reported by Jade Null

medium resolved

Category:

Broken Authentication and Session Management -> Privilege Escalation

CWE(s):

CWE-250: Execution with Unnecessary Privileges CWE-269: Improper Privilege Management CWE-284: Improper Access Control CWE-274: Improper Handling of Insufficient Privileges

Affected Assets

`app.acme.tld`

Affected Locations

`/v1/accounts/[org]/users/[user]`

Overview

During testing the team identified a privilege escalation vulnerability that would allow any user with the user role to upgrade their account to that of an admin. Doing so would allow the user account to take full ownership and control of the parent organisation and subsequently bypass all access controls in place.

Technical Details

To demonstrate this flaw, you will need two accounts. One initial "Admin" account (created when signing up) and a second invited user with the "User" role.
From the editor account, we first intercept any request within the application. This will show us that the account ID is `1337`.
Next, we intercept a request to update the user's profile using the "Job" dropdown. This will submit a PATCH request to `/v1/users/2`.
Using this information, the low-privileged user can now submit the following PATCH request to upgrade their account to that of the admin role.

```
PATCH /v1/accounts/1337/users/2 HTTP/2
Host: app.acme.tld
[...omitted for brevity...]

{"role":"ADMIN"}
```

As shown above, we send a PATCH request to update the `1337` organisation's `2` user account and assign the role `ADMIN`. Since no checks are in place, our underprivileged user now has full control of the account and can perform all actions within it.

Severity Detail

Successful exploitation of this attack would result in complete account takeover and compromise. A user who is able to escalate their privileges to that of an admin is able to bypass all access controls resulting in a complete lack of confidentiality and integrity of the target. It's important to note, for successful exploitation, the attacker must already have a user account within the target organisation, reducing the likelihood of attack. However when paired with the previously reported leaked credentials and MFA bypass, the likelihood is increased.

Remediation Steps

- Ensure proper logic is in place to prevent users from changing their role to a role of greater permissions.
- Implement access controls on the affected endpoint to ensure only administrative users can perform the reported action.

References

- OWASP: Authorization Cheat Sheet

Conclusion & General Comments

Overall, the security level of the Acme Co application was deemed fair, exhibiting only a limited number of vulnerabilities. However, it is strongly recommended that additional testing of related assets should be conducted to identify similar potential issues. Furthermore, it is advised to explore supplementary testing of pivot points into the internal network.
Additionally, the following points should be considered:

- Acme Co should continue to implement a consistent patch management cycle to include plugins and third-party libraries in use on all sites and infrastructure.
- Acme Co should provide public information for the desired point of contact and the process of reporting future issues. The transparency helps foster a collaborative environment and allows for the assistance of other potential researchers who find issues.
- Acme Co should consider performing security testing on a regular basis.

Document Change Log

Version	Date	Comment
v1	16 May 2023	Initial Report

Note: Document change log does not reflect updates to finding statuses as these are rendered dynamicly when downloading the report.
This PDF was generated and downloaded from the GlitchSecure platform on 02 August 2023 00:18 UTC.

Appendix - Coverage Checklist

As part of this assessment, GlitchSecure used the Application Security Verification Standard 4.0.3 Level 1 to ensure full and standardised coverage of the assets in scope. Below you will find a list of all tests completed.

V2 Authentication

V2.1 Password Security

✔ Verify that user set passwords are at least 12 characters in length (after multiple spaces are combined).
✔ Verify that passwords of at least 64 characters are permitted, and that passwords of more than 128 characters are denied.
✔ Verify that password truncation is not performed. However, consecutive multiple spaces may be replaced by a single space.
✔ Verify that any printable Unicode character, including language neutral characters such as spaces and Emojis are permitted in passwords.
✔ Verify users can change their password.
✔ Verify that password change functionality requires the user's current and new password.
✔ Verify that passwords submitted during account registration, login, and password change are checked against a set of breached passwords either locally (such as the top 1,000 or 10,000 most common passwords which match the system's password policy) or using an external API. If using an API a zero knowledge proof or other mechanism should be used to ensure that the plain text password is not sent or used in verifying the breach status of the password. If the password is breached, the application must require the user to set a new non-breached password.
✔ Verify that a password strength meter is provided to help users set a stronger password.
✔ Verify that there are no password composition rules limiting the type of characters permitted. There should be no requirement for upper or lower case or numbers or special characters.
✔ Verify that there are no periodic credential rotation or password history requirements.
✔ Verify that "paste" functionality, browser password helpers, and external password managers are permitted.

✔ Verify that the user can choose to either temporarily view the entire masked password, or temporarily view the last typed character of the password on platforms that do not have this as built-in functionality.

V2.2 General Authenticator Security

✔ Verify that anti-automation controls are effective at mitigating breached credential testing, brute force, and account lockout attacks. Such controls include blocking the most common breached passwords, soft lockouts, rate limiting, CAPTCHA, ever increasing delays between attempts, IP address restrictions, or risk-based restrictions such as location, first login on a device, recent attempts to unlock the account, or similar. Verify that no more than 100 failed attempts per hour is possible on a single account.
✔ Verify that the use of weak authenticators (such as SMS and email) is limited to secondary verification and transaction approval and not as a replacement for more secure authentication methods. Verify that stronger methods are offered before weak methods, users are aware of the risks, or that proper measures are in place to limit the risks of account compromise.
✔ Verify that secure notifications are sent to users after updates to authentication details, such as credential resets, email or address changes, logging in from unknown or risky locations. The use of push notifications - rather than SMS or email - is preferred, but in the absence of push notifications, SMS or email is acceptable as long as no sensitive information is disclosed in the notification.

V2.3 Authenticator Lifecycle

✔ Verify system generated initial passwords or activation codes SHOULD be securely randomly generated, SHOULD be at least 6 characters long, and MAY contain letters and numbers, and expire after a short period of time. These initial secrets must not be permitted to become the long term password.

V2.5 Credential Recovery

✔ Verify that a system generated initial activation or recovery secret is not sent in clear text to the user.
✔ Verify password hints or knowledge-based authentication (so-called "secret questions") are not present.
✔ Verify password credential recovery does not reveal the current password in any way.
✔ Verify shared or default accounts are not present (e.g. "root", "admin", or "sa").
✔ Verify that if an authentication factor is changed or replaced, that the user is notified of this event.
✔ Verify forgotten password, and other recovery paths use a secure recovery mechanism, such as time-based OTP (TOTP) or other soft token, mobile push, or

another offline recovery mechanism.

V2.7 Out of Band Verifier

✔ Verify that clear text out of band (NIST "restricted") authenticators, such as SMS or PSTN, are not offered by default, and stronger alternatives such as push notifications are offered first.
✔ Verify that the out of band verifier expires out of band authentication requests, codes, or tokens after 10 minutes.
✔ Verify that the out of band verifier authentication requests, codes, or tokens are only usable once, and only for the original authentication request.
✔ Verify that the out of band authenticator and verifier communicates over a secure independent channel.

V2.8 One Time Verifier

✔ Verify that time-based OTPs have a defined lifetime before expiring.

V3 Session Management

V3.1 Fundamental Session Management Security

✔ Verify the application never reveals session tokens in URL parameters.

V3.2 Session Binding

✔ Verify the application generates a new session token on user authentication.
✔ Verify that session tokens possess at least 64 bits of entropy.
✔ Verify the application only stores session tokens in the browser using secure methods such as appropriately secured cookies (see section 3.4) or HTML 5 session storage.

V3.3 Session Termination

✔ Verify that logout and expiration invalidate the session token, such that the back button or a downstream relying party does not resume an authenticated session, including across relying parties.
✔ If authenticators permit users to remain logged in, verify that re-authentication occurs both when actively used or after an idle period of 30 days.

V3.4 Cookie-based Session Management

✔ Verify that cookie-based session tokens have the 'Secure' attribute set.
✔ Verify that cookie-based session tokens have the 'HttpOnly' attribute set.
✔ Verify that cookie-based session tokens utilize the 'SameSite' attribute to limit exposure to cross-site request forgery attacks.

✔ Verify that cookie-based session tokens use the "__Host-" prefix so cookies are only sent to the host that initially set the cookie.
✔ Verify that if the application is published under a domain name with other applications that set or use session cookies that might disclose the session cookies, set the path attribute in cookie-based session tokens using the most precise path possible.

V3.7 Defenses Against Session Management Exploits

✔ Verify the application ensures a full, valid login session or requires re-authentication or secondary verification before allowing any sensitive transactions or account modifications.

V4 Access Control

V4.1 General Access Control Design

✔ Verify that the application enforces access control rules on a trusted service layer, especially if client-side access control is present and could be bypassed.
✔ Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized.
✔ Verify that the principle of least privilege exists - users should only be able to access functions, data files, URLs, controllers, services, and other resources, for which they possess specific authorization. This implies protection against spoofing and elevation of privilege.
✔ Verify that access controls fail securely including when an exception occurs.

V4.2 Operation Level Access Control

✔ Verify that sensitive data and APIs are protected against Insecure Direct Object Reference (IDOR) attacks targeting creation, reading, updating and deletion of records, such as creating or updating someone else's record, viewing everyone's records, or deleting all records.
✔ Verify that the application or framework enforces a strong anti-CSRF mechanism to protect authenticated functionality, and effective anti-automation or anti-CSRF protects unauthenticated functionality.

V4.3 Other Access Control Considerations

✔ Verify administrative interfaces use appropriate multi-factor authentication to prevent unauthorized use.
✔ Verify that directory browsing is disabled unless deliberately desired. Additionally, applications should not allow discovery or disclosure of file or directory metadata, such as Thumbs.db, .DS_Store, .git or .svn folders.

V5 Validation, Sanitization and Encoding

V5.1 Input Validation

✔ Verify that the application has defenses against HTTP parameter pollution attacks, particularly if the application framework makes no distinction about the source of request parameters (GET, POST, cookies, headers, or environment variables).
✔ Verify that frameworks protect against mass parameter assignment attacks, or that the application has countermeasures to protect against unsafe parameter assignment, such as marking fields private or similar.
✔ Verify that all input (HTML form fields, REST requests, URL parameters, HTTP headers, cookies, batch files, RSS feeds, etc) is validated using positive validation (allow lists).
✔ Verify that structured data is strongly typed and validated against a defined schema including allowed characters, length and pattern (e.g. credit card numbers, e-mail addresses, telephone numbers, or validating that two related fields are reasonable, such as checking that suburb and zip/postcode match).
✔ Verify that URL redirects and forwards only allow destinations which appear on an allow list, or show a warning when redirecting to potentially untrusted content.

V5.2 Sanitization and Sandboxing

✔ Verify that all untrusted HTML input from WYSIWYG editors or similar is properly sanitized with an HTML sanitizer library or framework feature.
✔ Verify that unstructured data is sanitized to enforce safety measures such as allowed characters and length.
✔ Verify that the application sanitizes user input before passing to mail systems to protect against SMTP or IMAP injection.
✔ Verify that the application avoids the use of eval() or other dynamic code execution features. Where there is no alternative, any user input being included must be sanitized or sandboxed before being executed.
✔ Verify that the application protects against template injection attacks by ensuring that any user input being included is sanitized or sandboxed.
✔ Verify that the application protects against SSRF attacks, by validating or sanitizing untrusted data or HTTP file metadata, such as filenames and URL input fields, and uses allow lists of protocols, domains, paths and ports.
✔ Verify that the application sanitizes, disables, or sandboxes user-supplied Scalable Vector Graphics (SVG) scriptable content, especially as they relate to XSS resulting from inline scripts, and foreignObject.
✔ Verify that the application sanitizes, disables, or sandboxes user-supplied scriptable or expression template language content, such as Markdown, CSS or XSL stylesheets,

BBCode, or similar.

V5.3 Output Encoding and Injection Prevention

✔ Verify that output encoding is relevant for the interpreter and context required. For example, use encoders specifically for HTML values, HTML attributes, JavaScript, URL parameters, HTTP headers, SMTP, and others as the context requires, especially from untrusted inputs (e.g. names with Unicode or apostrophes, such as ねこ or O'Hara).
✔ Verify that output encoding preserves the user's chosen character set and locale, such that any Unicode character point is valid and safely handled.
✔ Verify that context-aware, preferably automated - or at worst, manual - output escaping protects against reflected, stored, and DOM based XSS.
✔ Verify that data selection or database queries (e.g. SQL, HQL, ORM, NoSQL) use parameterized queries, ORMs, entity frameworks, or are otherwise protected from database injection attacks.
✔ Verify that where parameterized or safer mechanisms are not present, context-specific output encoding is used to protect against injection attacks, such as the use of SQL escaping to protect against SQL injection.
✔ Verify that the application protects against JSON injection attacks, JSON eval attacks, and JavaScript expression evaluation.
✔ Verify that the application protects against LDAP injection vulnerabilities, or that specific security controls to prevent LDAP injection have been implemented.
✔ Verify that the application protects against OS command injection and that operating system calls use parameterized OS queries or use contextual command line output encoding.
✔ Verify that the application protects against Local File Inclusion (LFI) or Remote File Inclusion (RFI) attacks.
✔ Verify that the application protects against XPath injection or XML injection attacks.

V5.5 Deserialization Prevention

✔ Verify that serialized objects use integrity checks or are encrypted to prevent hostile object creation or data tampering.
✔ Verify that the application correctly restricts XML parsers to only use the most restrictive configuration possible and to ensure that unsafe features such as resolving external entities are disabled to prevent XML eXternal Entity (XXE) attacks.
✔ Verify that deserialization of untrusted data is avoided or is protected in both custom code and third-party libraries (such as JSON, XML and YAML parsers).
✔ Verify that when parsing JSON in browsers or JavaScript-based backends, JSON.parse is used to parse the JSON document. Do not use eval() to parse JSON.

V6 Stored Cryptography

V6.2 Algorithms

✔ Verify that all cryptographic modules fail securely, and errors are handled in a way that does not enable Padding Oracle attacks.

V7 Error Handling and Logging

V7.1 Log Content

✔ Verify that the application does not log credentials or payment details. Session tokens should only be stored in logs in an irreversible, hashed form.
✔ Verify that the application does not log other sensitive data as defined under local privacy laws or relevant security policy.

V7.4 Error Handling

✔ Verify that a generic message is shown when an unexpected or security sensitive error occurs, potentially with a unique ID which support personnel can use to investigate.

V8 Data Protection

V8.2 Client-side Data Protection

✔ Verify the application sets sufficient anti-caching headers so that sensitive data is not cached in modern browsers.
✔ Verify that data stored in browser storage (such as localStorage, sessionStorage, IndexedDB, or cookies) does not contain sensitive data.
✔ Verify that authenticated data is cleared from client storage, such as the browser DOM, after the client or session is terminated.

V8.3 Sensitive Private Data

✔ Verify that sensitive data is sent to the server in the HTTP message body or headers, and that query string parameters from any HTTP verb do not contain sensitive data.
✔ Verify that users have a method to remove or export their data on demand.
✔ Verify that users are provided clear language regarding collection and use of supplied personal information and that users have provided opt-in consent for the use of that data before it is used in any way.
✔ Verify that all sensitive data created and processed by the application has been identified, and ensure that a policy is in place on how to deal with sensitive data.

V9 Communication

V9.1 Client Communication Security

✔ Verify that TLS is used for all client connectivity, and does not fall back to insecure or unencrypted communications.
✔ Verify using up to date TLS testing tools that only strong cipher suites are enabled, with the strongest cipher suites set as preferred.
✔ Verify that only the latest recommended versions of the TLS protocol are enabled, such as TLS 1.2 and TLS 1.3. The latest version of the TLS protocol should be the preferred option.

V10 Malicious Code

V10.3 Application Integrity

✔ Verify that if the application has a client or server auto-update feature, updates should be obtained over secure channels and digitally signed. The update code must validate the digital signature of the update before installing or executing the update.
✔ Verify that the application employs integrity protections, such as code signing or subresource integrity. The application must not load or execute code from untrusted sources, such as loading includes, modules, plugins, code, or libraries from untrusted sources or the Internet.
✔ Verify that the application has protection from subdomain takeovers if the application relies upon DNS entries or DNS subdomains, such as expired domain names, out of date DNS pointers or CNAMEs, expired projects at public source code repos, or transient cloud APIs, serverless functions, or storage buckets (autogen-bucket-id.cloud.example.com) or similar. Protections can include ensuring that DNS names used by applications are regularly checked for expiry or change.

V11 Business Logic

V11.1 Business Logic Security

✔ Verify that the application will only process business logic flows for the same user in sequential step order and without skipping steps.
✔ Verify that the application will only process business logic flows with all steps being processed in realistic human time, i.e. transactions are not submitted too quickly.
✔ Verify the application has appropriate limits for specific business actions or transactions which are correctly enforced on a per user basis.
✔ Verify that the application has anti-automation controls to protect against excessive calls such as mass data exfiltration, business logic requests, file uploads or denial of service attacks.

✔ Verify the application has business logic limits or validation to protect against likely business risks or threats, identified using threat modeling or similar methodologies.

V12 Files and Resources

V12.1 File Upload

✔ Verify that the application will not accept large files that could fill up storage or cause a denial of service.

V12.3 File Execution

✔ Verify that user-submitted filename metadata is not used directly by system or framework filesystems and that a URL API is used to protect against path traversal.
✔ Verify that user-submitted filename metadata is validated or ignored to prevent the disclosure, creation, updating or removal of local files (LFI).
✔ Verify that user-submitted filename metadata is validated or ignored to prevent the disclosure or execution of remote files via Remote File Inclusion (RFI) or Server-side Request Forgery (SSRF) attacks.
✔ Verify that the application protects against Reflective File Download (RFD) by validating or ignoring user-submitted filenames in a JSON, JSONP, or URL parameter, the response Content-Type header should be set to text/plain, and the Content-Disposition header should have a fixed filename.
✔ Verify that untrusted file metadata is not used directly with system API or libraries, to protect against OS command injection.

V12.4 File Storage

✔ Verify that files obtained from untrusted sources are stored outside the web root, with limited permissions.
✔ Verify that files obtained from untrusted sources are scanned by antivirus scanners to prevent upload and serving of known malicious content.

V12.5 File Download

✔ Verify that the web tier is configured to serve only files with specific file extensions to prevent unintentional information and source code leakage. For example, backup files (e.g. .bak), temporary working files (e.g. .swp), compressed files (.zip, .tar.gz, etc) and other extensions commonly used by editors should be blocked unless required.
✔ Verify that direct requests to uploaded files will never be executed as HTML/JavaScript content.

V12.6 SSRF Protection

✔ Verify that the web or application server is configured with an allow list of resources or systems to which the server can send requests or load data/files from.

V13 API and Web Service

V13.1 Generic Web Service Security

✔ Verify that all application components use the same encodings and parsers to avoid parsing attacks that exploit different URI or file parsing behavior that could be used in SSRF and RFI attacks.
✔ Verify API URLs do not expose sensitive information, such as the API key, session tokens etc.

V13.2 RESTful Web Service

✔ Verify that enabled RESTful HTTP methods are a valid choice for the user or action, such as preventing normal users using DELETE or PUT on protected API or resources.
✔ Verify that JSON schema validation is in place and verified before accepting input.
✔ Verify that RESTful web services that utilize cookies are protected from Cross-Site Request Forgery via the use of at least one or more of the following: double submit cookie pattern, CSRF nonces, or Origin request header checks.

V13.3 SOAP Web Service

✔ Verify that XSD schema validation takes place to ensure a properly formed XML document, followed by validation of each input field before any processing of that data takes place.

V14 Configuration

V14.2 Dependency

✔ Verify that all components are up to date, preferably using a dependency checker during build or compile time.
✔ Verify that all unneeded features, documentation, sample applications and configurations are removed.
✔ Verify that if application assets, such as JavaScript libraries, CSS or web fonts, are hosted externally on a Content Delivery Network (CDN) or external provider, Subresource Integrity (SRI) is used to validate the integrity of the asset.

V14.3 Unintended Security Disclosure

✔ Verify that web or application server and application framework debug modes are disabled in production to eliminate debug features, developer consoles, and unintended security disclosures.

✔ Verify that the HTTP headers or any part of the HTTP response do not expose detailed version information of system components.

V14.4 HTTP Security Headers

✔ Verify that every HTTP response contains a Content-Type header. Also specify a safe character set (e.g., UTF-8, ISO-8859-1) if the content types are text/*, /+xml and application/xml. Content must match with the provided Content-Type header.
✔ Verify that all API responses contain a Content-Disposition: attachment; filename="api.json" header (or other appropriate filename for the content type).
✔ Verify that a Content Security Policy (CSP) response header is in place that helps mitigate impact for XSS attacks like HTML, DOM, JSON, and JavaScript injection vulnerabilities.
✔ Verify that all responses contain a X-Content-Type-Options: nosniff header.
✔ Verify that a Strict-Transport-Security header is included on all responses and for all subdomains, such as Strict-Transport-Security: max-age=15724800; includeSubdomains.
✔ Verify that a suitable Referrer-Policy header is included to avoid exposing sensitive information in the URL through the Referer header to untrusted parties.
✔ Verify that the content of a web application cannot be embedded in a third-party site by default and that embedding of the exact resources is only allowed where necessary by using suitable Content-Security-Policy: frame-ancestors and X-Frame-Options response headers.

V14.5 HTTP Request Header Validation

✔ Verify that the application server only accepts the HTTP methods in use by the application/API, including pre-flight OPTIONS, and logs/alerts on any requests that are not valid for the application context.
✔ Verify that the supplied Origin header is not used for authentication or access control decisions, as the Origin header can easily be changed by an attacker.
✔ Verify that the Cross-Origin Resource Sharing (CORS) Access-Control-Allow-Origin header uses a strict allow list of trusted domains and subdomains to match against and does not support the "null" origin.