

ETX Signal-X

Daily Intelligence Digest

Friday, January 30, 2026

10

ARTICLES

21

TECHNIQUES

Article 1

Complete Guide to Dnsx for Mass DNS Resolution and Bug Bounty

Source: securitycipher

Date: 18-Oct-2025

URL: <https://medium.com/@jpablo13/complete-guide-to-dnsx-for-mass-dns-resolution-and-bug-bounty-f8bed1598770>

 Methodology article - no vulnerabilities

Recon on Mobile APIs: The Hidden Attack Surface in Your Pocket

Source: securitycipher

Date: 27-Aug-2025

URL: <https://medium.com/meetcyber/recon-on-mobile-apis-the-hidden-attack-surface-in-your-pocket-7145915edcbe>

T1 Abuse of Hidden Mobile API Endpoints

Payload

```
GET /api/v1/mobile/hidden-endpoint?user_id=1234 HTTP/1.1
Host: api.target.com
Authorization: Bearer <mobile-app-token>
```

Attack Chain

- 1 Decompile the mobile application APK/IPA to extract API endpoint URLs and request structures.
- 2 Identify endpoints not referenced in public web documentation (e.g., /api/v1/mobile/hidden-endpoint).
- 3 Replay requests to these endpoints using valid mobile tokens.
- 4 Observe responses for sensitive data or privileged actions.

Discovery

Manual inspection of decompiled mobile app code and network traffic revealed undocumented API endpoints.

Bypass

Endpoints are not protected by additional server-side checks; only mobile-specific tokens required.

Chain With

Combine with IDOR or privilege escalation if endpoint exposes or modifies data for arbitrary user_id values.

T2

Weak Mobile API Authentication Bypass via User-Agent Spoofing

⚡ Payload

```
POST /api/v1/mobile/authenticate HTTP/1.1
Host: api.target.com
User-Agent: MobileApp/2.5.1 (iPhone; iOS 15.3)
Content-Type: application/json

>{"username":"victim","password":"password123"}
```

✗ Attack Chain

- 1 Capture mobile app authentication request and note the custom User-Agent string.
- 2 Replicate the request in Burp Suite or curl, spoofing the User-Agent to match the mobile app.
- 3 Send authentication requests directly to the mobile API endpoint.
- 4 Gain access if server only checks User-Agent for mobile API access.

🔍 Discovery

Observed server-side logic only checked for a specific User-Agent string to allow access to mobile-only endpoints.

🔒 Bypass

No additional device attestation or certificate pinning enforced; User-Agent is the only gate.

🔗 Chain With

Use to access mobile-only features, then escalate via mobile-specific vulnerabilities or test for weak session management.

T3

Sensitive Data Exposure via Verbose Mobile API Responses

⚡ Payload

```
GET /api/v1/mobile/user/profile HTTP/1.1
Host: api.target.com
Authorization: Bearer <mobile-app-token>
```

⚔️ Attack Chain

- 1 Access mobile API endpoints with a valid token.
- 2 Observe response payloads for overexposed fields (e.g., internal IDs, PII, debug info).
- 3 Compare with web API responses to identify excessive data returned only to mobile clients.

🔍 Discovery

Difffed mobile and web API responses for the same resource and found additional sensitive fields in mobile API.

🔒 Bypass

Mobile API lacks field-level filtering or data minimization compared to web API.

🔗 Chain With

Leverage exposed internal identifiers or debug info for further enumeration, privilege escalation, or lateral movement.

Article 3

From oos—getting bounty Improper Access Control to create an commentator account on the application.

Source: securitycipher

Date: 06-Jan-2025

URL: <https://medium.com/@swaroopvenkat828/from-oos-getting-bounty-improper-access-control-to-create-an-commentator-account-on-the-application-027b517928bb>

T1 Improper Access Control for Commentor Account Creation

Payload

```
POST /api/v1/commentor/register HTTP/1.1
Host: target.com
Content-Type: application/json

{
  "email": "attacker@example.com",
  "password": "P@ssw0rd123",
  "other_field": "value"
}
```

Attack Chain

- 1 Identify the endpoint `/api/v1/commentor/register` intended for creating commentor accounts.
- 2 Craft and send a POST request with arbitrary email and password values.
- 3 Observe that the application creates a commentor account without any prior authorization or invitation requirement.
- 4 Use the newly created account to interact with application features reserved for commentors.

Discovery

Manual endpoint enumeration and testing of unauthenticated API routes revealed the registration endpoint did not enforce access controls or require invitation tokens.

Bypass

No authentication or invitation code was required to access the endpoint; simply POSTing to the endpoint sufficed.

Chain With

Use the commentor account to access commentor-only features, potentially leading to privilege escalation or lateral movement if further misconfigurations exist. Combine with IDOR or privilege escalation bugs if commentor accounts have elevated access in other areas.

How I Uncovered an Email Leak That Could Have Cost Millions

Source: securitycipher

Date: 03-Mar-2025

URL: <https://krishna-cyber.medium.com/how-i-uncovered-an-email-leak-that-could-have-cost-millions-66500548d0b6>

T1 Email Leak via Unauthenticated API Endpoint

Payload

```
GET /api/v1/users/list?limit=1000 HTTP/1.1  
Host: target.com
```

Attack Chain

- 1 Identified an API endpoint `/api/v1/users/list` that returned user data.
- 2 Sent an unauthenticated GET request with a high `limit` parameter value.
- 3 Received a JSON response containing email addresses and other PII for all users.

Discovery

Manual endpoint enumeration and inspecting API responses for sensitive data exposure. The researcher noticed the endpoint did not require authentication and returned bulk user data.

Bypass

No authentication or authorization checks were present on the endpoint, allowing direct access.

Chain With

Harvested emails could be used for targeted phishing, credential stuffing, or further social engineering attacks against the organization.

T2

Increasing Data Exposure via Parameter Manipulation

⚡ Payload

```
GET /api/v1/users/list?limit=10000 HTTP/1.1  
Host: target.com
```

⚔️ Attack Chain

- ➊ Noticed the `limit` parameter controlled the number of users returned.
- ➋ Incrementally increased the `limit` value to 10,000.
- ➌ The endpoint returned a massive dump of user emails and PII in a single response.

🔍 Discovery

Parameter fuzzing on the `limit` value to test for upper bounds and data volume returned. The researcher experimented with different values to maximize data extraction.

🔒 Bypass

No server-side restriction or pagination enforcement on the `limit` parameter, allowing mass data extraction in one request.

🔗 Chain With

Bulk data extraction enables rapid enumeration of the user base, facilitating large-scale attacks such as spam, phishing, or credential stuffing.

Part 12: BOLA Detection in Mobile Apps and Single-Page Applications (SPAs)

Source: securitycipher

Date: 16-Jun-2025

URL: <https://medium.com/@narendarlb123/part-12-bola-detection-in-mobile-apps-and-single-page-applications-spas-69c0843b35b4>

T1

BOLA via Mobile App API Parameter Manipulation

⚡ Payload

```
GET /api/v1/user/profile?userId=12345
```

⚔️ Attack Chain

- 1 Intercept mobile app traffic using a proxy (e.g., Burp Suite, mitmproxy).
- 2 Identify API requests where user-controllable identifiers (e.g., userId) are sent as parameters.
- 3 Modify the `userId` parameter to another valid user's ID (e.g., change `12345` to `12346`).
- 4 Observe if the API returns another user's profile data without authorization checks.

🔍 Discovery

Manual inspection of API calls from the mobile app, focusing on endpoints with user identifiers in parameters.

🔒 Bypass

No authentication or authorization checks enforced on the userId parameter; server trusts client-supplied identifiers.

🔗 Chain With

Can be chained with session fixation or privilege escalation if sensitive data or tokens are exposed in the profile response.

T2

BOLA in Single-Page Applications via JavaScript-Exposed Endpoints

⚡ Payload

```
fetch('/api/v1/orders/67890', { headers: { 'Authorization': 'Bearer <token>' } })
```

⚔️ Attack Chain

- 1 Analyze the SPA's JavaScript bundle for hardcoded API endpoints and resource patterns (e.g., `/orders/{orderId}`).
- 2 Use the browser's dev tools or intercept requests to identify the structure of resource URLs.
- 3 Replace the resource identifier (e.g., `67890`) with another valid ID belonging to a different user.
- 4 Send the modified request with a valid session token and observe if unauthorized data is returned.

🔍 Discovery

Source code review of the SPA's JavaScript files and observation of network requests to enumerate resource endpoints.

🔒 Bypass

Authorization checks are performed only on the frontend; backend fails to enforce proper access control on resource identifiers.

🔗 Chain With

Can be combined with enumeration attacks to harvest valid resource IDs, or with XSS to automate mass exploitation.

T3

BOLA via Mobile App Local Storage Token Reuse

⚡ Payload

```
POST /api/v1/account/details
Authorization: Bearer <stolen_token>
Content-Type: application/json
{
  "accountId": "98765"
}
```

✗ Attack Chain

- 1 Extract authentication tokens stored in the mobile app's local storage or device files (e.g., via rooted device or backup extraction).
- 2 Replay API requests using the stolen token, modifying the `accountId` parameter to target other accounts.
- 3 Observe if the API returns details for accounts not owned by the token's original user.

🔍 Discovery

Analysis of mobile app storage mechanisms and token handling, followed by replaying requests with manipulated identifiers.

🔓 Bypass

Backend does not verify that the token's subject matches the accountId parameter, allowing cross-account access.

🔗 Chain With

May be chained with token theft attacks (e.g., via malware or insecure storage) for lateral movement across user accounts.

Phishing-Style Link Reflected on Microsoft Azure Portal—Not XSS, But Still Tricky

Source: securitycipher

Date: 23-Apr-2025

URL: <https://medium.com/@melege/phishing-style-link-reflected-on-microsoft-azure-portal-not-xss-but-still-tricky-559bde6f8252>

T1 Reflected Phishing-Style Link in Azure Portal

Payload

```
https://portal.azure.com/?feature.customportal=false&feature.customportal.uri=https://evil.com
```

Attack Chain

- 1 Attacker crafts a URL to the Azure Portal with the `feature.customportal.uri` parameter set to a malicious domain (e.g., `https://evil.com`).
- 2 Victim is lured to click the crafted link.
- 3 Upon visiting, the Azure Portal reflects the attacker-controlled URL in a prominent location (e.g., a clickable link or redirection prompt) within the portal interface.
- 4 Victim may be tricked into clicking the reflected link, believing it to be a legitimate part of the Azure Portal, leading to phishing or further attacks.

Discovery

Manual parameter fuzzing of Azure Portal URL parameters, specifically targeting those with URI or URL in their names. The researcher noticed the `feature.customportal.uri` parameter reflected attacker input in the UI.

Bypass

No XSS or direct code execution possible, but the technique bypasses user trust by leveraging UI reflection to make phishing links appear more credible within the Azure Portal context.

Chain With

Can be chained with credential phishing campaigns targeting Azure users. May be combined with social engineering to increase click-through rates. Potential for further abuse if other parameters allow deeper integration or automatic redirection.

Brute Force Bonanza: Hacking into Web & SSH Logins to Capture the Flag

Source: securitycipher

Date: 11-Jul-2025

URL: <https://medium.com/@jabaribrown62/brute-force-bonanza-hacking-into-web-ssh-logins-to-capture-the-flag-f14c31e17d35>

T1

Web Login Brute Force via Automated Password List

Payload

```
admin:password123
admin:admin123
admin:qwerty
```

Attack Chain

- 1 Identify the web login portal endpoint (e.g., /login or /admin).
- 2 Use an automated tool (e.g., Burp Suite Intruder, Hydra) to send POST requests with usernames and a password list.
- 3 Monitor server responses for status codes, error messages, or redirects indicating successful authentication.
- 4 Upon successful login, access the authenticated area and search for sensitive data or flags.

Discovery

Observed lack of account lockout or rate limiting on the web login form during manual login attempts.

Bypass

No CAPTCHA or IP blocking enabled; unlimited attempts allowed.

Chain With

Use compromised credentials to pivot to other internal applications or escalate privilege if password reuse is present.

T2 SSH Brute Force Using Discovered Credentials

⚡ Payload

```
ssh admin@target-ip  
Password: password123
```

⚔️ Attack Chain

- 1 After obtaining valid credentials from web brute force, attempt SSH login to the same target using those credentials.
- 2 Use SSH client or automated tools (e.g., Hydra, Medusa) to attempt login.
- 3 Upon successful authentication, gain shell access to the server.
- 4 Enumerate the system for sensitive files, flags, or further lateral movement.

🔍 Discovery

Noticed that the web application's admin credentials were also valid for SSH access.

🔒 Bypass

No 2FA or IP restrictions on SSH; password authentication enabled.

🔗 Chain With

Shell access allows for privilege escalation, lateral movement, and direct file access (e.g., flag capture).

T3 Response-Based Brute Force Detection Evasion

⚡ Payload

```
Change User-Agent and X-Forwarded-For headers per request
```

⚔️ Attack Chain

- 1 During brute force attempts, rotate User-Agent and X-Forwarded-For headers for each request.
- 2 Use scripting or proxy tools to automate header changes.
- 3 Evade basic detection and rate-limiting mechanisms that rely on static headers or IPs.
- 4 Continue brute force attempts without triggering security controls.

🔍 Discovery

Detected that the application performed basic IP-based rate limiting and User-Agent fingerprinting.

🔒 Bypass

Header rotation bypasses naive detection and allows sustained brute force attacks.

🔗 Chain With

Combine with distributed brute force (multiple proxies) for large-scale credential stuffing.

\$25,000 SSRF in HackerOne's Analytics Reports

Source: securitycipher

Date: 12-May-2025

URL: <https://osintteam.blog/25-000-ssrf-in-hackerones-analytics-reports-b9a5b3aa3d6e>

T1 SSRF via Analytics Report Export Function

⚡ Payload

```
{"reportType": "custom", "filters": {"dateRange": "LAST_30_DAYS"}, "exportFormat": "PDF", "exportUrl": "http://169.254.169.254/latest/meta-data/iam/security-credentials/"}
```

✗ Attack Chain

- 1 Authenticate to HackerOne as a user with access to Analytics Reports.
- 2 Intercept the POST request to `/reports/export` when exporting a report.
- 3 Modify the `exportUrl` parameter in the JSON body to a target internal resource (e.g., AWS metadata endpoint).
- 4 Send the modified request.
- 5 The server-side export function fetches the provided URL, resulting in SSRF.

🔍 Discovery

Observed that the export functionality accepted a user-supplied `exportUrl` parameter. Hypothesized that this parameter could be abused for SSRF by pointing it to internal resources.

🔒 Bypass

No explicit SSRF protections (e.g., blocklists or allowlists) were enforced on the `exportUrl` parameter, allowing direct access to internal IPs and AWS metadata endpoints.

🔗 Chain With

Use SSRF to access cloud metadata endpoints and retrieve credentials for further privilege escalation. Potential to pivot to other internal services accessible from the application server.

Bug Bounty: Cookie timeline utilization when the system uses multiple authentication cookies

Source: securitycipher

Date: 17-Mar-2025

URL: <https://medium.com/@expression4865/bug-bounty-cookie-timeline-utilization-when-the-system-uses-multiple-authentication-cookies-7d2e6cc68c34>

T1 Session Fixation via Stale Secondary Authentication Cookie

Payload

```
Set-Cookie: auth_token=attacker_token; Path=/; HttpOnly
```

Attack Chain

- 1 Attacker logs in to their own account and captures their valid `auth_token` value.
- 2 Attacker crafts a link or request that sets the `auth_token` cookie in the victim's browser to the attacker's token (via XSS, open redirect, or other means).
- 3 Victim logs in with their own credentials, but the stale `auth_token` persists alongside any new session cookies.
- 4 Application prioritizes the stale `auth_token` over the new session, authenticating the victim as the attacker.

Discovery

Noticed that the application sets multiple authentication cookies and does not clear old tokens on login. Manual testing revealed that the system accepts the oldest valid token if multiple are present.

Bypass

Persistence of the attacker's token is possible because the application does not invalidate or overwrite old authentication cookies on new login events.

Chain With

Can be chained with XSS, open redirect, or any attack that allows setting cookies in the victim's browser to escalate from token theft to full account takeover.

T2

Privilege Escalation via Overlapping Auth Cookie Timelines

⚡ Payload

```
Cookie: sessionid=lowpriv_token; auth_token=highpriv_token
```

⚔️ Attack Chain

- 1 Attacker obtains a high-privilege `auth_token` (e.g., via phishing or prior compromise).
- 2 Attacker logs in as a low-privilege user, receiving a new `sessionid` cookie, but manually injects the high-privilege `auth_token` into requests.
- 3 Application checks both cookies and, due to flawed logic, grants access based on the higher-privilege token, regardless of the active session.

🔍 Discovery

Observed that the application sets both `sessionid` and `auth_token` cookies and does not properly scope privileges per session. Experimented with mixing tokens of different privilege levels in a single request.

🔒 Bypass

Bypassing privilege checks is possible because the application does not bind privilege level to the active session, but rather to whichever valid token is present in the request.

🔗 Chain With

Can be combined with token theft, session fixation, or social engineering to escalate privileges or maintain persistent access across privilege boundaries.

T3

Forced Authentication via Cookie Replay in Multi-Tab Browsing

⚡ Payload

```
Cookie: auth_token=attacker_token
```

⚔️ Attack Chain

- 1 Victim logs in to the application in one browser tab (with their own valid cookies).
- 2 Attacker tricks the victim into opening a crafted link in a new tab that sets the `auth_token` to the attacker's value (e.g., via subdomain or path confusion, or by exploiting a vulnerable endpoint).
- 3 Victim's browser now sends both their own and the attacker's cookies on requests.
- 4 Application authenticates the victim as the attacker in the new tab, causing confusion or account mix-up.

🔍 Discovery

Tested cookie handling in multi-tab scenarios and observed that the application does not isolate authentication state per tab or session, leading to cross-tab authentication confusion.

🔒 Bypass

The attack works because the application does not scope authentication cookies to a single session or browser context, allowing cookie replay across tabs.

🔗 Chain With

Can be used to force actions as another user, hijack sessions, or facilitate phishing attacks by blending authentication states in the victim's browser.

Bug Bounty Diaries: How a Leaked appsettings.json Became a High-Impact Find

Source: securitycipher

Date: 26-Sep-2025

URL: <https://0xbasak.medium.com/bug-bounty-diaries-how-a-leaked-appsettings-json-became-a-high-impact-find-57c3e19e0a36>

T1

Leaked appsettings.json Exposing Sensitive Credentials

Payload

```
GET /appsettings.json HTTP/1.1
```

```
Host: target.com
```

Attack Chain

- 1 Identify the presence of an accessible `appsettings.json` file by requesting `/appsettings.json` on the target web server.
- 2 Review the file contents for sensitive information such as database connection strings, API keys, JWT secrets, or third-party service credentials.
- 3 Extract credentials (e.g., Azure SQL connection string, SMTP credentials, JWT signing key) from the file.
- 4 Use the leaked credentials to access backend infrastructure, impersonate users, or escalate privileges.

Discovery

Manual directory and file brute-forcing for common configuration files. The researcher specifically targeted `.NET` and Azure environments, hypothesizing that misconfigured deployments might expose `appsettings.json`.

Bypass

N/A (Direct access due to misconfiguration)

Chain With

Use database credentials for direct DB access or lateral movement. Use JWT secret to forge authentication tokens for privilege escalation. Use SMTP credentials for phishing or internal email compromise.

T2

JWT Token Forgery via Leaked Signing Key

Payload

```
Header: { "alg": "HS256", "typ": "JWT" }
Payload: { "sub": "admin", "role": "admin" }
Signature: HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), <leaked_jwt_secret>)
```

Attack Chain

- 1 Extract the JWT signing key from the leaked `appsettings.json` file.
- 2 Craft a new JWT with elevated privileges (e.g., `role: admin`).
- 3 Sign the JWT using the leaked secret.
- 4 Submit the forged JWT as a session or authorization token to the application.
- 5 Gain unauthorized access to admin functionality or sensitive data.

Discovery

After identifying the leaked `appsettings.json`, the researcher noticed the presence of a `Jwt:Key` parameter and tested its use for signing tokens.

Bypass

Bypasses authentication and authorization controls by forging valid tokens with the leaked secret.

Chain With

Combine with IDOR or admin-only endpoints for full account takeover or data exfiltration. Use forged tokens to access internal APIs or trigger further privilege escalation.

T3

Direct Database Access via Leaked Connection String

⚡ Payload

```
Server=tcp:target.database.windows.net,1433;Initial Catalog=prod-db;Persist Security Info=False;User ID=produser;Password=SuperSecretPass123!;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;
```

⚔️ Attack Chain

- 1 Extract the database connection string from the leaked `appsettings.json`.
- 2 Use the credentials to connect directly to the Azure SQL database (e.g., via `sqlcmd`, `Azure Data Studio`, or custom scripts).
- 3 Enumerate tables, extract sensitive data, or modify database contents.

🔍 Discovery

Manual inspection of the leaked configuration file for connection strings and credentials.

🔒 Bypass

Bypasses network segmentation and application-layer access controls by leveraging direct DB credentials.

🔗 Chain With

Use DB access to plant web shells, backdoors, or further pivot into internal networks. Combine with knowledge of application logic for privilege escalation or data tampering.

T4

SMTP Credential Abuse for Internal Phishing or Email Spoofing

⚡ Payload

```
"Smtp": {  
  "Host": "smtp.sendgrid.net",  
  "Port": 587,  
  "Username": "apikey",  
  "Password": "SG.xxxxxxxxxx"  
}
```

✗ Attack Chain

- 1 Extract SMTP credentials from the leaked `appsettings.json`.
- 2 Use credentials to authenticate against the SMTP server (e.g., via `swaks`, `sendemail`, or custom scripts).
- 3 Send emails as the target organization, enabling phishing, internal spoofing, or social engineering attacks.

🔍 Discovery

Inspection of configuration file for SMTP settings after initial leak discovery.

🔒 Bypass

Bypasses sender verification and internal email restrictions by leveraging valid credentials.

🔗 Chain With

Launch targeted phishing campaigns against employees or customers. Use email access to reset passwords or trigger further attacks via social engineering.