CAPEC-132: Symlink Attack (Version 3.9)

Archived: 2026-04-05 15:52:54 UTC

Description

An adversary positions a symbolic link in such a manner that the targeted user or application accesses the link's endpoint, assuming that it is accessing a file with the link's name.

Extended Description

The endpoint file may be either output or input. If the file is output, the result is that the endpoint is modified, instead of a file at the intended location. Modifications to the endpoint file may include appending, overwriting, corrupting, changing permissions, or other modifications.

In some variants of this attack the adversary may be able to control the change to a file while in other cases they cannot. The former is especially damaging since the adversary may be able to grant themselves increased privileges or insert false information, but the latter can also be damaging as it can expose sensitive information or corrupt or destroy vital system or application files. Alternatively, the endpoint file may serve as input to the targeted application. This can be used to feed malformed input into the target or to cause the target to process different information, possibly allowing the adversary to control the actions of the target or to cause the target to expose information to the adversary. Moreover, the actions taken on the endpoint file are undertaken with the permissions of the targeted user or application, which may exceed the permissions that the adversary would normally have.

Likelihood Of Attack

Typical Severity

Relationships

This table shows the other attack patterns and high level categories that are related to this attack pattern. These relationships are defined as ChildOf and ParentOf, and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as CanFollow, PeerOf, and CanAlsoBe are defined to show similar attack patterns that the user may want to explore.

Nature	Type
ChildOf	S Standard Attack Pattern - A standard level attack pattern in CAPEC is focused on a specific methodology or technique used in an attack. It

This table shows the views that this attack pattern belongs to and top level categories within that view.

Execution Flow

Explore

1. **Identify Target:** Adversary identifies the target application by determining whether there is sufficient check before writing data to a file and creating symlinks to files in different directories.

Techniques
The adversary writes to files in different directories to check whether the application has sufficient checking before file operations.
The adversary creates symlinks to files in different directories.

Experiment

1. **Try to create symlinks to different files:** The adversary then uses a variety of techniques, such as monitoring or guessing to create symlinks to the files accessed by the target application in the directories which are identified in the explore phase.

Techniques
The adversary monitors the file operations performed by the target application using a tool like dtrace or FileMon. And the adversary can delay the operations by using "sleep(2)" and "usleep()" to prepare the

appropriate conditions for the attack, or make the application perform expansive tasks (large files parsing, etc.) depending on the purpose of the application.
The adversary may need a little guesswork on the filenames on which the target application would operate.
The adversary tries to create symlinks to the various filenames.

Exploit

1. **Target application operates on created symlinks to sensitive files:** The adversary is able to create symlinks to sensitive files while the target application is operating on the file.

Techniques
Create the symlink to the sensitive file such as configuration files, etc.

Prerequisites

The targeted application must perform the desired activities on a file without checking whether the file is a symbolic link or not. The adversary must be able to predict the name of the file the target application is modifying and be able to create a new symbolic link where that file would appear.

Skills Required

[Level: Low] To create symlinks
[Level: High] To identify the files and create the symlinks during the file operation time window

Resources Required

None: No specialized resources are required to execute this type of attack. The only requirement is the ability to create the necessary symbolic link.

Consequences

This table specifies different individual consequences associated with the attack pattern. The Scope identifies the security property that is violated, while the Impact describes the negative technical impact that arises if an adversary succeeds in their attack. The Likelihood provides information about how likely the specific consequence is expected to be seen relative to the other consequences in the list. For example, there may be high likelihood that a pattern will be used to achieve a certain impact, but a low likelihood that it will be exploited to achieve a different impact.

Scope	**Impact**	**Likelihood**
Confidentiality	Other	
Integrity	Modify Data	
Confidentiality	Read Data	
Integrity	Modify Data	
Authorization	Execute Unauthorized Commands	

Accountability Authentication Authorization Non-Repudiation	Gain Privileges	
Access Control Authorization	Bypass Protection Mechanism	
Availability	Unreliable Execution	

Mitigations

Design: Check for the existence of files to be created, if in existence verify they are neither symlinks nor hard links before opening them.
Implementation: Use randomly generated file names for temporary files. Give the files restrictive permissions.

Example Instances

The adversary creates a symlink with the "same" name as the file which the application is intending to write to. The application will write to the file- "causing the data to be written where the symlink is pointing". An attack like this can be demonstrated as follows:

root# vulprog myFile

{...program does some processing...]

adversary# ln –s /etc/nologin myFile

[...program writes to 'myFile', which points to /etc/nologin...]

In the above example, the root user ran a program with poorly written file handling routines, providing the filename "myFile" to vulnprog for the relevant data to be written to. However, the adversary happened to be looking over the shoulder of "root" at the time, and created a link from myFile to /etc/nologin. The attack would make no user be able to login.

Taxonomy Mappings

CAPEC mappings to ATT&CK techniques leverage an inheritance model to streamline and minimize direct CAPEC/ATT&CK mappings. Inheritance of a mapping is indicated by text stating that the parent CAPEC has relevant ATT&CK mappings. Note that the ATT&CK Enterprise Framework does not use an inheritance model as part of the mapping to CAPEC.

Relevant to the ATT&CK taxonomy mapping (also see parent)

Entry ID	Entry Name
1547.009	Boot or Logon Autostart Execution:Shortcut Modification

References

Content History

Submissions		
Submission Date	**Submitter**	**Organization**
2014-06-23 (Version 2.6)	CAPEC Content Team	The MITRE Corporation
Modifications		
Modification Date	**Modifier**	**Organization**

2015-11-09 (Version 2.7)	CAPEC Content Team	The MITRE Corporation
	Updated References	
2017-08-04 (Version 2.11)	CAPEC Content Team	The MITRE Corporation
	Updated Resources_Required	
2019-04-04 (Version 3.1)	CAPEC Content Team	The MITRE Corporation
	Updated Consequences	
2020-07-30 (Version 3.3)	CAPEC Content Team	The MITRE Corporation
	Updated Taxonomy_Mappings	
2022-02-22 (Version 3.7)	CAPEC Content Team	The MITRE Corporation
	Updated Description, Example_Instances, Execution_Flow, Extended_Description, Prerequisites	
2022-09-29 (Version 3.8)	CAPEC Content Team	The MITRE Corporation
	Updated Example_Instances	

More information is available — Please select a different filter.
