T1574.006 - Dynamic Linker Hijacking

Description from ATT&CK

> Adversaries may execute their own malicious payloads by hijacking environment variables the dynamic linker uses to load shared libraries. During the execution preparation phase of a program, the dynamic linker loads specified absolute paths of shared libraries from environment variables and files, such as `LD_PRELOAD` on Linux or `DYLD_INSERT_LIBRARIES` on macOS. Libraries specified in environment variables are loaded first, taking precedence over system libraries with the same function name.(Citation: Man LD.SO)(Citation: TLDP Shared Libraries)(Citation: Apple Doco Archive Dynamic Libraries) These variables are often used by developers to debug binaries without needing to recompile, deconflict mapped symbols, and implement custom functions without changing the original library.(Citation: Baeldung LD_PRELOAD)
> On Linux and macOS, hijacking dynamic linker variables may grant access to the victim process's memory, system/network resources, and possibly elevated privileges. This method may also evade detection from security products since the execution is masked under a legitimate process. Adversaries can set environment variables via the command line using the `export` command, `setenv` function, or `putenv` function. Adversaries can also leverage Dynamic Linker Hijacking to

> export variables in a shell or set variables programmatically using higher level syntax such Python's `os.environ`.
>
> On Linux, adversaries may set `LD_PRELOAD` to point to malicious libraries that match the name of legitimate libraries which are requested by a victim program, causing the operating system to load the adversary's malicious code upon execution of the victim program. `LD_PRELOAD` can be set via the environment variable or `/etc/ld.so.preload` file.(Citation: Man LD.SO)(Citation: TLDP Shared Libraries) Libraries specified by `LD_PRELOAD` are loaded and mapped into memory by `dlopen()` and `mmap()` respectively.(Citation: Code Injection on Linux and macOS)(Citation: Uninformed Needle) (Citation: Phrack halfdead 1997)(Citation: Brown Exploiting Linkers)
>
> On macOS this behavior is conceptually the same as on Linux, differing only in how the macOS dynamic libraries (dyld) is implemented at a lower level. Adversaries can set the `DYLD_INSERT_LIBRARIES` environment variable to point to malicious libraries containing names of legitimate libraries or functions requested by a victim program.(Citation: TheEvilBit DYLD_INSERT_LIBRARIES)(Citation: Timac DYLD_INSERT_LIBRARIES)(Citation: Gabilondo DYLD_INSERT_LIBRARIES Catalina Bypass)

Atomic Tests

Atomic Test #1 - Shared Library Injection via /etc/ld.so.preload

This test adds a shared library to the `ld.so.preload` list to execute and intercept API calls. This technique was used by threat actor Rocke during the exploitation of Linux web servers. This requires the `glibc` package.

Upon successful execution, bash will echo `../bin/T1574.006.so` to /etc/ld.so.preload.

Supported Platforms: Linux

auto_generated_guid: 39cb0e67-dd0d-4b74-a74b-c072db7ae991

Inputs:

Name	Description	Type	Default Value
path_to_shared_library_source	Path to a shared library source code	Path	PathToAtomicsFolder/T1574.006/src/Linux
path_to_shared_library	Path to a shared library object	Path	/tmp/T1574006.so

Attack Commands: Run with `bash`! Elevation Required (e.g. root or admin)

```
sudo sh -c 'echo #{path_to_shared_library} > /etc/ld.so.preload'
```

Cleanup Commands:

```
sudo sed -i 's##{path_to_shared_library}##' /etc/ld.so.preload
```

Dependencies: Run with `bash`!

Description: The shared library must exist on disk at specified location (#{path_to_shared_library})

Check Prereq Commands:

```
if [ -f #{path_to_shared_library} ]; then exit 0; else exit 1; fi;
```

Get Prereq Commands:

```
gcc -shared -fPIC -o #{path_to_shared_library} #{path_to_shared_library_source}
```

Atomic Test #2 - Shared Library Injection via LD_PRELOAD

This test injects a shared object library via the LD_PRELOAD environment variable to execute. This technique was used by threat actor Rocke during the exploitation of Linux web servers. This requires the `glibc` package.

Upon successful execution, bash will utilize LD_PRELOAD to load the shared object library `/etc/ld.so.preload`. Output will be via stdout.

Supported Platforms: Linux

auto_generated_guid: bc219ff7-789f-4d51-9142-ecae3397deae

Inputs:

Name	Description	Type	Default Value
path_to_shared_library_source	Path to a shared library source code	Path	PathToAtomicsFolder/T1574.006/src/Linux
path_to_shared_library	Path to a shared library object	Path	/tmp/T1574006.so

Attack Commands: Run with `bash`!

```
LD_PRELOAD=#{path_to_shared_library} ls
```

Dependencies: Run with `bash`!

Description: The shared library must exist on disk at specified location (#{path_to_shared_library})

Check Prereq Commands:

```
if [ -f #{path_to_shared_library} ]; then exit 0; else exit 1; fi;
```

Get Prereq Commands:

```
gcc -shared -fPIC -o #{path_to_shared_library} #{path_to_shared_library_source}
```

Atomic Test #3 - Dylib Injection via DYLD_INSERT_LIBRARIES

injects a dylib that opens calculator via env variable

Supported Platforms: macOS

auto_generated_guid: 4d66029d-7355-43fd-93a4-b63ba92ea1be

Inputs:

Name	Description	Type	Default Value
file_to_inject	Path of executable to be injected. Mostly works on non-apple default apps.	Path	/Applications/Firefox.app/Contents/MacOS/firefox
source_file	Path of c source file	Path	PathToAtomicsFolder/T1574.006/src/MacOS/T1574.006.c
dylib_file	Path of dylib file	Path	/tmp/T1574006MOS.dylib

Attack Commands: Run with `bash`!

```
DYLD_INSERT_LIBRARIES=#{dylib_file} #{file_to_inject}
```

Cleanup Commands:

```
kill `pgrep Calculator`
kill `pgrep firefox`
```

Dependencies: Run with `bash`!

Description: Compile the dylib from (#{source_file}). Destination is #{dylib_file}

Check Prereq Commands:

```
gcc -dynamiclib #{source_file} -o #{dylib_file}
```

Get Prereq Commands:

```
gcc -dynamiclib #{source_file} -o #{dylib_file}
```