# ServiceDLL v1.3
# Grasshopper Component User Guide

# 1 Description

ServiceDLL is a Grasshopper component that provides a way to persist a payload as a Windows Service DLL.

The ServiceDLL component installs a stub Service DLL to the Net Services (netsvcs) Service Host using manual registry modifications. The stub is configured to run the input payload whenever the service starts. The stub is stored at a user specified location on the target file system.

The payload is stored as a resource of the ServiceDLL stub. If the payload adheres to the NOD Persistence Spec v1 Interface, the stub will load and execute the payload from memory if using stub A or stub B. If not, the stub will write the payload to the filesystem and load or run it normally. The payload will be placed adjacent to the stub with a `.tlb` file extension for default stub A, DLLNAMEhlp.{dll|exe} for stub type stub B, DLLNAMEext.{dll|exe} for stub type stub C, DLLNAMEapi.{dll|exe} for stub type stub D, DLLNAMElib.{dll|exe} for stub type stub E, or DLLNAMEres.{dll|exe} for stub type stub F.

Due to caching by the Service Control Manager, the service cannot be started directly when first installed. The ServiceDLL component can, optionally, hijack an existing, stopped service DLL's entry in the SCM database to gain immediate execution. This requires that the component write an "Unhijack DLL" to the filesystem, which is deleted by the stub during the first run.

# 2 Usage

## 2.1 Builder Command Line

```
add component servicedll -n NAME –p PATH [–d DESC] [-u PATH]
```

`-n/--name NAME` cover name of the service dll

`-p/--path PATH` target path of the service dll stub

`-d/--description DESC` cover description of the service dll

`-u/--unhijack PATH` uses service hijack technique and provides path for the unhijack dll for resetting service cache

`--hijack` hijack a service for immediate execution without resetting service cache

`--stubname STUBNAME` alternate stubname to use {A|B|C|D|E|F} [default A]

`-k/--killfile PATH` kill file path which causes persistence and payload to be uninstalled [default no kill file]

### Example

```
(gh) add component servicedll
     -n ExampleService
     –p "c:\windows\system32\example.dll"
     -d "An example of how to create a service dll component."
     -u "%temp%\examplehelper.dll"
```

## 2.2 Supported Payload Types

ServiceDLL accepts input payloads in EXE or DLL formats for the x86 or x64 architectures. If a payload DLL supports the NOD Persistence Specification, the stub will memory load it during execution if using Stub A or B otherwise it is written to

disk and loaded. ServiceDLL is a terminating component and does not output a payload.

| Input Type | Output Type(s) |
|---|---|
| x86 DLL nod-persist | None |
| x64 DLL nod-persist | None |
| x86 DLL | None |
| x64 DLL | None |
| x86 EXE | None |
| x64 EXE | None |

## 2.3 Supported Variant Stubnames

As part of the ServiceDLL component version 1.3, variant stubs were added. Six stubs are available the default stub A, and stub B, stub C, stub D, stub E, and stub F.

1. The default stub A uses the grasshopper common code base and uses resources data to store configuration information as well as the compressed and obfuscated payload(using xor with random key). Stub A uses a payload file name identical to service dll filename except with a .tlb extension. It utilizes the CRT. Stub A also supports NOD-persist dlls and performs memory loading of the payload when NOD persist dlls are specified.
2. Stub B stub uses alternate resource ids, and uses deleteservice function to remove service entries vs. using registry manipulation in standard stub, additionally it does not use grasshopper common code. It stores the configuration data as an obfuscated resource(using xor with random key). It stores the payload as an obfuscated resource for nod-persist. For a non-nod-persist payload it is written directly to disk from the grasshopper module so, stub does not contain an obfuscated copy of payload. Stub B uses a payload file name identical to service dll filename except with a hlp.{exe|dll} suffix and extension. It utilizes the CRT. Stub B also supports NOD-persist dlls and performs memory loading of the payload when NOD persist payload dlls are specified.
3. Stub C stub uses alternate resource ids, and uses Reg.exe utility to remove service entries vs. using registry api in standard stub, additionally it does not use grasshopper common code. It stores the configuration data as an obfuscated resource(using xor with random key). The payload is written directly to disk from the grasshopper module so, stub does not contain an obfuscated copy of payload. Stub C uses a payload file name identical to service dll filename except with a ext.{exe|dll} suffix and extension. There is no CRT dependency for stub C. Stub C does not support memory loading of NOD-persist dlls.
4. Stub D stub uses alternate resource ids, and uses the SC command to remove service entries vs. using registry manipulation in standard stub, additionally it does not use grasshopper common code. It stores the configuration data as an obfuscated resource(using xor with random key). The payload is written directly to disk from the grasshopper module so, stub does not contain an

obfuscated copy of payload. Stub D uses a payload file name identical to service dll filename except with a api.{exe|dll} suffix and extension. There is no CRT because internally tinylibC is used to provide crt functionality. Stub D does not support memory loading of NOD-persist dlls.

5. Stub E stub uses alternate resource ids, and uses the SC command to remove service entries vs. using registry manipulation in standard stub, additionally it does not use grasshopper common code. It stores the configuration data as an obfuscated resource(using rc4 with random key). The payload is written directly to disk from the grasshopper module so, stub does not contain an obfuscated copy of payload. Stub E uses a payload file name identical to service dll filename except with a lib.{exe|dll} suffix and extension. There is a CRT dependency for stub E. Stub E does not support memory loading of NOD-persist dlls.
6. Stub F stub uses alternate resource ids, and uses the uses registry manipulation to remove service, additionally it does not use grasshopper common code. It stores the configuration data as an obfuscated resource(rolling xor with random key The payload is written directly to disk from the grasshopper module so, stub does not contain an obfuscated copy of payload. Stub F uses a payload file name identical to service dll filename except with a res.{exe|dll} suffix and extension. There is no CRT dependency for stub F. Stub F does not support memory loading of NOD-persist dlls.

## 2.4 Uninstall Procedure

### Manual

The manual uninstall procedure consists of the following steps:

1. Stop the service, if it is running.
   ```
   sc stop <SERVICE_NAME>
   ```
2. Delete the service from the Service Control Manager.
   ```
   sc delete <SERVICE_NAME>
   ```
3. Reboot the target.
4. Delete the stub and payload executables from the filesystem.
   ```
   del /F <SERVICE_PATH> <PAYLOAD_PATH>
   ```

### Autonomous

The autonomous uninstall procedure consists of the following steps:

1. Delete the payload from the files system while the stub is running.

When the stub detects that the payload has been deleted, it will execute the autonomous uninstall. The stub checks for the payload every 5 seconds. The autonomous uninstall will perform the following steps:

1. Remove the service from the Windows registry.
2. Delete itself from the filesystem.

### Kill File

The kill file uninstall procedure consists of the following steps:

1. Create a file on the file system at path specified for kill file parameter at build time.

When the stub detects the presence of the kill file, it will execute the kill file uninstall procedure. The stub checks for the kill file every minute. The uninstall proceeds through the following steps:

1. Wait half a minute before starting uninstall.
2. Attempt to signal and/or stop the payload for uninstall.
3. Secure delete the payload. If this fails, arrange to delete on reboot.
4. Remove the service from the SCM.
5. Remove the kill file.
6. Delete itself from the filesystem.

*NOTE:* If the payload is a DLL, the stub will attempt to free library. If the payload has not performed a “safety load” on itself and does not shutdown, it may crash the host process.

*NOTE:* If the payload is a NOD-persisted DLL, it will have been memory loaded. On uninstall, the stub will call DLLMain with DLL_DETACH_PROCESS to notify the payload of the uninstall event. However, the memory is leaked and the payload left running to avoid potentially crashing the host process.

*NOTE:* If payload is an EXE payload, the payload will be terminated using TerminateProcess and securely deleted.

*NOTE:* If the uninstall fails, the kill file remains and the uninstall function will be attempted again on the next boot.

NOTE: The hijack technique may fail to find a useable service and fail to hijack. If this happens then the service will start normally on next reboot of system.

NOTE: All Stubs perform secure self-deletion of themselves during un-installation.

# 3 Footprint

### File System

- Service Stub Executable, located at a user specified location `<STUB_PATH>`
- Service Stub Directory, may have been created
- Nodpersist interface payloads are not on disk but are an obfuscated resource in the stub
- Standard Payload Executables are located at (for default Stub A) `<STUB_PATH.tlb> or` (for default Stub B) `<STUB_PATH>hlp.<exe|dll> or` (for default Stub C) `<STUB_PATH>ext.<exe|dll> or` (for default Stub D) `<STUB_PATH>api.<exe|dll> or` (for default Stub E) `<STUB_PATH>lib.<exe|dll> or` (for default Stub F) `<STUB_PATH>res.<exe|dll> depending on payload type`
- Payload Directory, may have been created
- Unhijack Executable, located at a user specified location `<UNHIJACK_PATH>`
- Unhijack Directory, may have been created

## Registry Keys

### Created

- HKLM\SYSTEM\CurrentControlSet\Services\<SERVICE_NAME>
- HKLM\SYSTEM\CurrentControlSet\Services\<SERVICE_NAME>\ImagePath
- HKLM\SYSTEM\CurrentControlSet\Services\<SERVICE_NAME>\ObjectName
- HKLM\SYSTEM\CurrentControlSet\Services\<SERVICE_NAME>\DelayedAutoStart
- HKLM\SYSTEM\CurrentControlSet\Services\<SERVICE_NAME>\ErrorControl
- HKLM\SYSTEM\CurrentControlSet\Services\<SERVICE_NAME>\Start
- HKLM\SYSTEM\CurrentControlSet\Services\<SERVICE_NAME>\Type
- HKLM\SYSTEM\CurrentControlSet\Services\<SERVICE_NAME>\Parameters
- HKLM\SYSTEM\CurrentControlSet\Services\<SERVICE_NAME>\Parameters\ServiceDll
- HKLM\SYSTEM\CurrentControlSet\Services\<SERVICE_NAME>\Description
- HKLM\SYSTEM\CurrentControlSet\Services\<SERVICE_NAME>\DisplayName

### Modified

- HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SvcHost\netsvcs

### Modified (during hijack)

- HKLM\SYSTEM\CurrentControlSet\Services\<HIJACKED_SERVICE>\Parameters\ServiceDll
- HKLM\SYSTEM\CurrentControlSet\Services\<HIJACKED_SERVICE>\Parameters\ServiceDll UnloadOnStop