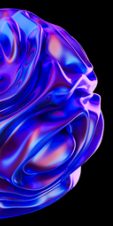

LOADERS
UNLEASHED
HADESS
WWW.HADESS.IO



INTRODUCTION

In the ever-evolving landscape of programming languages and their capabilities, one aspect remains constant: the need for loaders. Loaders, in their various forms, play a crucial role in software development by facilitating the execution of code from external sources. Across a diverse spectrum of programming languages, from stalwarts like C/C++ to dynamic languages like Python and JavaScript, loaders serve as gateways to harnessing the full potential of these languages. In this exploration, we delve into the realm of loaders, unveiling their mechanisms, functionalities, and nuances across different languages.

C/C++ stands as the bedrock of modern software development, and its loaders are pivotal in linking external libraries and resources with the main program. These loaders enable seamless integration of modules written in C/C++, enriching the application's functionality and efficiency.
Python, revered for its simplicity and versatility, offers multiple avenues for loading external code. Through mechanisms like **eval** and **exec**, Python empowers developers to dynamically execute snippets of code, fostering flexibility and adaptability in script execution.

The interoperability of Python extends even further with its integration into other languages like C#. By embedding Python within C# applications, developers can leverage Python's extensive libraries and functionalities, enriching their C# projects with additional capabilities.

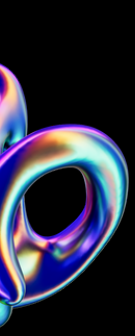
PHP, a cornerstone of web development, relies on loaders to include external files and execute dynamic code snippets. With functionalities like **include** and **eval**, PHP empowers developers to dynamically incorporate content into their web applications, enhancing their flexibility and responsiveness.

PowerShell, a powerful automation and scripting language in the Windows environment, offers loaders that facilitate the execution of external scripts and commands. With functionalities like **IEX** and integration into C#, PowerShell empowers developers to automate tasks and streamline workflows across various platforms.

In the dynamic landscape of software development, the synergy between languages is crucial. The integration of C# assemblies into PowerShell scripts and the execution of JScript within PowerShell environments exemplify the seamless interoperability and versatility of modern programming languages.

As a bonus, the incorporation of loaders with encoders further enhances the security and efficiency of software deployment. By encoding loaders, developers can obfuscate their code and mitigate potential security risks, ensuring the integrity and confidentiality of their applications.

In conclusion, loaders serve as the linchpin of modern software development, enabling the seamless integration and execution of external code across a myriad of programming languages. Whether it's linking C/C++ libraries, dynamically executing Python scripts, or automating tasks with PowerShell, loaders play a pivotal role in harnessing the full potential of programming languages, facilitating innovation and efficiency in software development.



DOCUMENT INFO

HADESS

To be the vanguard of cybersecurity, Hadess envisions a world where digital assets are safeguarded from malicious actors. We strive to create a secure digital ecosystem, where businesses and individuals can thrive with confidence, knowing that their data is protected. Through relentless innovation and unwavering dedication, we aim to establish Hadess as a symbol of trust, resilience, and retribution in the fight against cyber threats.

At Hadess, our mission is twofold: to unleash the power of white hat hacking in punishing black hat hackers and to fortify the digital defenses of our clients. We are committed to employing our elite team of expert cybersecurity professionals to identify, neutralize, and bring to justice those who seek to exploit vulnerabilities. Simultaneously, we provide comprehensive solutions and services to protect our client's digital assets, ensuring their resilience against cyber attacks. With an unwavering focus on integrity, innovation, and client satisfaction, we strive to be the guardian of trust and security in the digital realm.

Security Researcher
Amir Gholizadeh (@arimaqz)

TABLE OF CONTENT

C#

- Assembly Load
- C# in PowerShell

Bonus

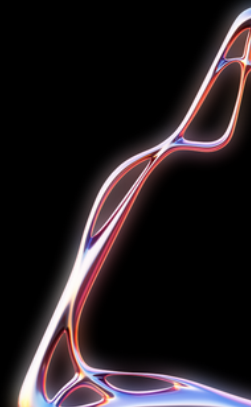
- Executing JScript in PowerShell
- Loader with Encoder

Executive Summary

1. This technical summary explores loaders across various languages, highlighting their functionalities and mechanisms.
2. C/C++ loaders play a crucial role in linking external libraries and resources with main programs. They facilitate the incorporation of C/C++ modules, enhancing application functionality and efficiency by enabling seamless integration of external code.
3. Python offers versatile mechanisms such as **eval** and **exec** for dynamically executing code snippets. Additionally, Python's integration into other languages like C# extends its capabilities, enabling developers to embed Python scripts within C# applications, thus leveraging Python's extensive libraries and functionalities.
4. PHP relies on loaders to include external files and execute dynamic code snippets. With functionalities like **include** and **eval**, PHP empowers developers to dynamically incorporate content into web applications, enhancing their flexibility and responsiveness.
5. Javascript's loaders, including **load** and **require**, facilitate the fetching and execution of external scripts, enabling developers to modularize their codebase and improve maintainability and performance in client-side scripting.
6. Ruby's loaders, represented by **load** and **require**, facilitate the inclusion of external modules and libraries. These mechanisms empower developers to seamlessly integrate external code into Ruby applications, enhancing functionality and extensibility.
7. In Perl, loaders enable the evaluation and execution of external code snippets through mechanisms like **eval** and **do**. This flexibility allows developers to dynamically incorporate functionality into Perl scripts, enhancing adaptability and efficiency.
8. PowerShell loaders, such as **IEX**, facilitate the execution of external scripts and commands, enabling automation and streamlining workflows in Windows environments. Integration with C# further extends PowerShell's capabilities, enabling the incorporation of C# assemblies into PowerShell scripts for enhanced automation and versatility.
9. C# loaders, represented by assembly loading mechanisms, enable the integration of external assemblies into C# applications. Additionally, C# integration into PowerShell environments enhances automation capabilities, allowing developers to execute C# code within PowerShell scripts for streamlined workflows.

Key Findings

As a bonus, loaders with encoders enhance security and efficiency in software deployment. By encoding loaders, developers can obfuscate their code, mitigating security risks and ensuring the integrity and confidentiality of their applications. Overall, loaders play a critical role in modern software development, facilitating seamless integration and execution of external code across diverse programming languages.



Abstract

Loaders serve as indispensable tools in software development, facilitating the integration and execution of external code across a diverse range of programming languages. From the foundational C/C++ to dynamic languages like Python and JavaScript, loaders play a pivotal role in enhancing application functionality and efficiency by seamlessly incorporating external resources.

Python, known for its versatility, offers mechanisms like **eval** and **exec** for dynamically executing code snippets, while its integration into other languages like C# expands its capabilities, enabling developers to embed Python scripts within C# applications. Similarly, PHP relies on loaders like **include** and **eval** to incorporate external files and execute dynamic code, enhancing the flexibility and responsiveness of web applications.

JavaScript's loaders enable the fetching and execution of external scripts, allowing developers to modularize their codebase and improve maintainability and performance in client-side scripting. Meanwhile, loaders in languages like Ruby and Perl provide mechanisms for seamlessly integrating external modules and libraries, enhancing functionality and adaptability.

In addition to facilitating code execution, loaders play a crucial role in enhancing security and efficiency in software deployment. By encoding loaders, developers can obfuscate their code and mitigate potential security risks, ensuring the integrity and confidentiality of their applications. Overall, loaders stand as fundamental components in modern software development, enabling seamless integration and execution of external code across a myriad of programming languages.

HADESS.IO

Loader

Load Good or Bad Thing

01

Attacks

When developing malware/red teaming tools, it's often needed to dynamically execute code inside a program. For example executing python code inside a python file. The reason that it's needed is for evasion, is because when the code is being loaded like that, it resides in memory so AV/EDR has more overhead when examining the process. Plus it can be encrypted/encoded and decrypted/decoded and then executed for more evasion.

C/C++

We say C/C++ but it's about loading compiled programs that are in standard PE format. There are already many programs written to reflectively load C/C++ EXEs/DLLs. The first PoC was written by Stephen Fewers to reflectively load DLLs: https://github.com/stephenfewer/ReflectiveDLLInjection
Other PoCs are mostly based on Stephen Fewer's work. What we'll discuss in this section is what reflective loading is about. It's about creating the PE's structure in memory which includes imports, exports, sections and so on and then linked together and executed in the process's memory. It's identical to the way Windows itself loads executables but instead of relying on Windows to execute it for us, we implement it on our own.

Python

To dynamically execute python code inside python there are two ways:

- eval
- exec

Furthermore python version 2 can also be executed in C# using IronPython.

For python we are going to load this file:

tobeloaded.py

```
print("loaded")
```

Eval

Eval has two methods when loading python files:

- Eval: one line can be executed
- Exec: multiline can be executed

```
fp = open("tobeloaded.py","r").read()
eval(compile(fp,'<string>','eval'))
```

```
fp = open("tobeloaded.py","r").read()
eval(compile(fp,'<string>','exec'))
```

Exec

Exec can be used to dynamically load python files:

```
fp = open("tobeloaded.py","r").read()
exec(fp)
```

```
PS C:\Users [redacted]Documents\projects\test\python-loader> python .\loader.py
loaded
```

Python in C#

Python2 code can be loaded in C# using IronPython to be dynamically executed:

```
using System;
using IronPython.Hosting;
using IronPython.Runtime;
using IronPython;
using Microsoft.Scripting.Hosting;
namespace MalTestSharp
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            ScriptEngine engine = Python.CreateEngine();
            string lines = File.ReadAllText(@"C:\\Users [redacted]ocuments\\projects\\test\\python-loader\\tobeloaded.py");
            var result = engine.Execute(lines);
        }
    }
}
```

And when executed:

```
Microsoft Visual Studio Debug Console
loaded
```

PHP

PHP code can be executed in two ways:

- Include: to execute PHP files
- Eval: to execute PHP code

Include

To execute a PHP file one simply needs to include it:

```
└─$ cat load.php; echo "----------"; cat loader.php; echo "----------"; php loader.php
<?php print "loaded";?>
----------
<?php include "load.php"; ?>
----------
loaded
```

Eval

To execute PHP code eval can be used:

Javascript

Javascript code can dynamically be executed inside Javascript using the eval method which is considered dangerous a function and rarely used:

Ruby

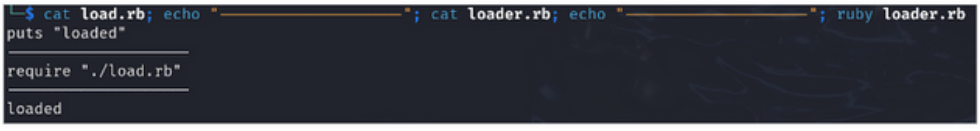
To load Ruby there are two ways::

- Load
- Require

Both of these methods execute Ruby files.

Load

Require



Perl

In Perl language there are also two ways to execute Perl files:

- Eval
- Do

Eval

To load a perl file using eval it first needs to be loaded and read, and then executed:

Do

Using the do method, Perl files can be loaded using only one line:

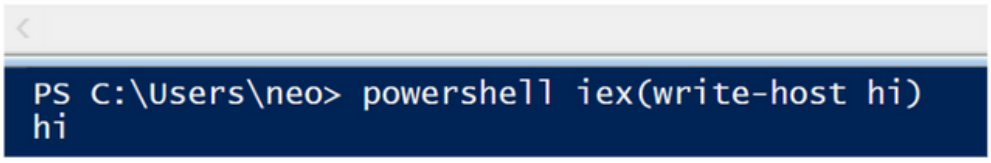
PowerShell

PowerShell code can be executed using iex and in C# using the powershell engine.

IEX

One only needs to pass the powershell code to IEX to be executed:

```
1   powershell iex(write-host hi)
```



PowerShell in C#

To load PowerShell code/file inside C#, its DLL, System.Management.Automation, has to included and its engine created:

```
using System;
using System.Management.Automation;
using System.Management.Automation.Runspaces;
using System.Runtime.InteropServices;
namespace MalTestSharp
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            Runspace rs = RunspaceFactory.CreateRunspace();
            rs.Open();
            PowerShell ps = PowerShell.Create();
            ps.Runspace = rs;
            ps.AddScript("powershell -ep bypass write-host loaded");
            var results = ps.Invoke();
            if (results.Count == 0)
            {
                Console.WriteLine("[-] Powershell returned error");
                return;
            }
            foreach (var obj in results)
            {
                if (obj != null)
                {
                    Console.WriteLine(obj.BaseObject.ToString());
                }
            }
            rs.Close();
        }
    }
}
```

```
C:\Use[redacted]rce\repos\Loader\Loader\bin\x64\Debug\Loader.exe
loaded
```

C#

C# executable file can be executed inside C# using assembly loader and C# code itself in PowerShell using Add-Type.

Assembly Load

To load the C# executable using assembly loader, it needs to be read and then an instance of the class should be created, then the method that needs to be executed should be found and invoked:

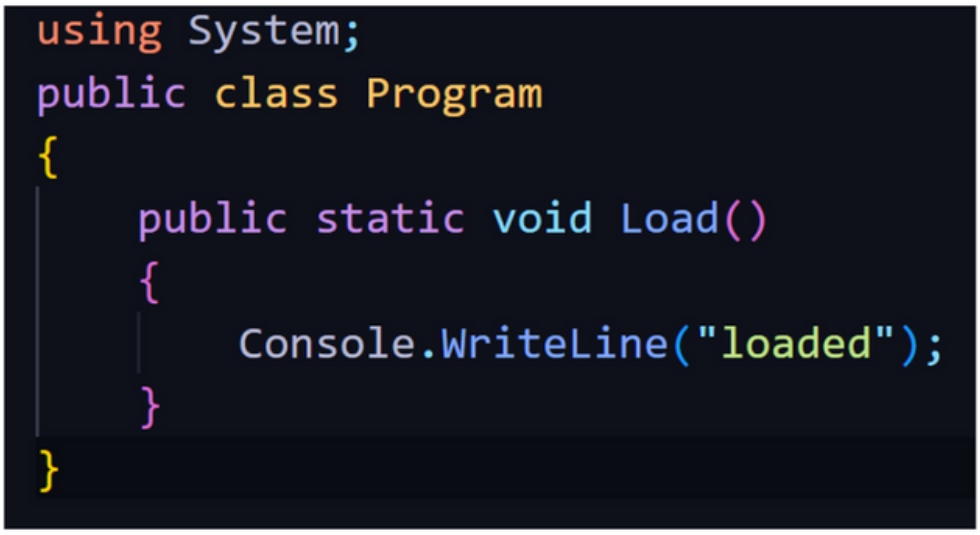
```
using System;

namespace ToBeLoaded2
{
    0 references
    internal class Program
    {
        0 references
        private static void Main(string[] args)
        {
        }
        0 references
        private void Load()
        {
            Console.WriteLine("loaded");
        }
    }
}
```

```
byte[] file = System.IO.File.ReadAllBytes("C:\\ToBeLoaded2.exe");
var loaded = System.Reflection.Assembly.Load(file);
var internalClass = loaded.GetType("ToBeLoaded2.Program");
var obj = System.Activator.CreateInstance(internalClass);
var method = internalClass.GetMethod("Load", System.Reflection.BindingFlags.NonPublic | System.Reflection.BindingFlags.Instance);
method.Invoke(obj, null);
Console.ReadLine();
```

```
C:\Use[redacted]urce\repos\Loader\Loader\bin\x64\Debug\Loader.exe
loaded
```

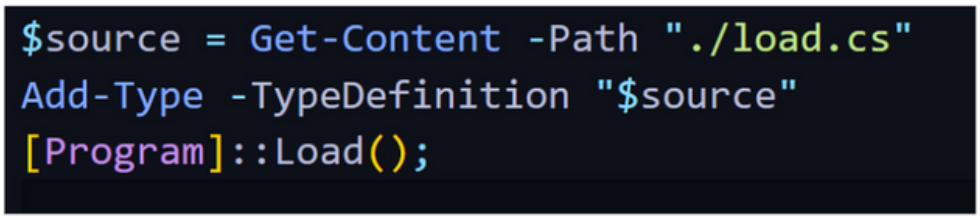
C# in PowerShell

The C# file that'll be used:



```
using System;
public class Program
{
    public static void Load()
    {
        Console.WriteLine("loaded");
    }
}
```

And to load the C# file in PowerShell:



```
$source = Get-Content -Path "./load.cs"
Add-Type -TypeDefinition "$source"
[Program]::Load();
```

Bonus

Executing JScript in PowerShell

JScript/VisualBasic can be used using PowerShell. This enables malware authors to evade more security measures because they are not frequently used nowadays:

```
$src = @'
static function run(arg){
    var shell = new ActiveXObject("WScript.shell");
    shell.run(arg);
}
'@
$JSC = (Add-Type -Language JScript -MemberDefinition $src -Name "JSC" -PassThru)[1]
$JSC::run("cmd");
```

Loader with Encoder

As a bonus for reading till now, we'll be writing a loader in python and also implement an encoder/decoder to encode the python file, and then decode it when loading it giving us more a more sense of evasion:

```
from base64 import b64encode,b64decode
import sys

if sys.argv[1] == "encrypt":
    f = open(sys.argv[2], "r")
    content = f.read()
    f.close()

    f2 = open(sys.argv[2]+".enc", "w")
    f2.write((b64encode(content.encode())).decode())
    f2.close()

elif sys.argv[1] == "load":
    f = open(sys.argv[2], "r")
    content = f.read()
    content = b64decode(content.encode())
    exec(content)
```

```
PS C:\User [redacted] ocuments\projects\test\py> python .\final-loader.py encrypt .\tobeloaded.py
PS C:\User [redacted] ocuments\projects\test\py> python .\final-loader.py load .\tobeloaded.py.enc
loaded
```

And this is just a dirty simple Python code prepared for this article, there are many ways to improve it further to be a capable Python loader.

Conclusion

In conclusion, the exploration of loaders across various programming languages underscores their fundamental importance in modern software development. From C/C++ to Python, PHP, JavaScript, Ruby, Perl, PowerShell, and C#, loaders serve as the linchpin for integrating and executing external code seamlessly. These loaders empower developers to enhance application functionality, flexibility, and efficiency by incorporating external resources dynamically.

The versatility of loaders is evident in their diverse functionalities across different languages. Whether it's dynamically executing code snippets in Python, fetching external scripts in JavaScript, or integrating external modules in Ruby and Perl, loaders enable developers to modularize their codebase and improve maintainability, performance, and adaptability.

Furthermore, loaders play a critical role in ensuring the security and integrity of software applications. By encoding loaders, developers can obfuscate their code and mitigate potential security risks, safeguarding sensitive information and preserving the confidentiality of their applications.

Overall, loaders stand as indispensable tools in the toolkit of every software developer, facilitating the seamless integration and execution of external code across a wide spectrum of programming languages. As technology continues to evolve, loaders will undoubtedly remain a cornerstone of software development, enabling innovation, efficiency, and reliability in the creation of diverse and sophisticated applications.

cat ~/.hadess

"Hadess" is a cybersecurity company focused on safeguarding digital assets and creating a secure digital ecosystem. Our mission involves punishing hackers and fortifying clients' defenses through innovation and expert cybersecurity services.

Website:
WWW.HADESS.IO

Email
MARKETING@HADESS.IO

To be the vanguard of cybersecurity, Hadess envisions a world where digital assets are safeguarded from malicious actors. We strive to create a secure digital ecosystem, where businesses and individuals can thrive with confidence, knowing that their data is protected. Through relentless innovation and unwavering dedication, we aim to establish Hadess as a symbol of trust, resilience, and retribution in the fight against cyber threats.