Shellcode

From Wikipedia, the free encyclopedia

[Jump to navigation](https://en.wikipedia.org/wiki/Shellcode#mw-head)[Jump to search](https://en.wikipedia.org/wiki/Shellcode#searchInput)

In [hacking](https://en.wikipedia.org/wiki/Hacker_(computer_security)), a **shellcode** is a small piece of code used as the [payload](https://en.wikipedia.org/wiki/Payload_(computing)) in the [exploitation](https://en.wikipedia.org/wiki/Exploit_(computer_security)) of a software [vulnerability](https://en.wikipedia.org/wiki/Vulnerability_(computing)). It is called "shellcode" because it typically starts a [command shell](https://en.wikipedia.org/wiki/Shell_(computing)) from which the attacker can control the compromised machine, but any piece of code that performs a similar task can be called shellcode. Because the function of a payload is not limited to merely spawning a shell, some have suggested that the name shellcode is insufficient.[[1]](https://en.wikipedia.org/wiki/Shellcode#cite_note-1) However, attempts at replacing the term have not gained wide acceptance. Shellcode is commonly written in [machine code](https://en.wikipedia.org/wiki/Machine_code).



**Contents**

* [1Types of shellcode](https://en.wikipedia.org/wiki/Shellcode#Types_of_shellcode)
  + [1.1Local](https://en.wikipedia.org/wiki/Shellcode#Local)
  + [1.2Remote](https://en.wikipedia.org/wiki/Shellcode#Remote)
  + [1.3Download and execute](https://en.wikipedia.org/wiki/Shellcode#Download_and_execute)
  + [1.4Staged](https://en.wikipedia.org/wiki/Shellcode#Staged)
  + [1.5Egg-hunt](https://en.wikipedia.org/wiki/Shellcode#Egg-hunt)
  + [1.6Omelette](https://en.wikipedia.org/wiki/Shellcode#Omelette)
* [2Shellcode execution strategy](https://en.wikipedia.org/wiki/Shellcode#Shellcode_execution_strategy)
* [3Shellcode encoding](https://en.wikipedia.org/wiki/Shellcode#Shellcode_encoding)
  + [3.1Percent encoding](https://en.wikipedia.org/wiki/Shellcode#Percent_encoding)
  + [3.2Null-free shellcode](https://en.wikipedia.org/wiki/Shellcode#Null-free_shellcode)
  + [3.3Alphanumeric and printable shellcode](https://en.wikipedia.org/wiki/Shellcode#Alphanumeric_and_printable_shellcode)
  + [3.4Unicode proof shellcode](https://en.wikipedia.org/wiki/Shellcode#Unicode_proof_shellcode)
* [4Platforms](https://en.wikipedia.org/wiki/Shellcode#Platforms)
* [5Shellcode analysis](https://en.wikipedia.org/wiki/Shellcode#Shellcode_analysis)
* [6See also](https://en.wikipedia.org/wiki/Shellcode#See_also)
* [7References](https://en.wikipedia.org/wiki/Shellcode#References)
* [8External links](https://en.wikipedia.org/wiki/Shellcode#External_links)

Types of shellcode[[edit](https://en.wikipedia.org/w/index.php?title=Shellcode&action=edit&section=1)]

Shellcode can either be *local* or *remote*, depending on whether it gives an attacker control over the machine it runs on (local) or over another machine through a network (remote).

**Local**[[edit](https://en.wikipedia.org/w/index.php?title=Shellcode&action=edit&section=2)]

*Local* shellcode is used by an attacker who has limited access to a machine but can exploit a vulnerability, for example a [buffer overflow](https://en.wikipedia.org/wiki/Buffer_overflow), in a higher-privileged process on that machine. If successfully executed, the shellcode will provide the attacker access to the machine with the same higher privileges as the targeted process.

**Remote**[[edit](https://en.wikipedia.org/w/index.php?title=Shellcode&action=edit&section=3)]

*Remote* shellcode is used when an attacker wants to target a vulnerable process running on another machine on a [local network](https://en.wikipedia.org/wiki/Local_area_network), [intranet](https://en.wikipedia.org/wiki/Intranet), or a [remote network](https://en.wikipedia.org/wiki/Internet). If successfully executed, the shellcode can provide the attacker access to the target machine across the network. Remote shellcodes normally use standard [TCP/IP](https://en.wikipedia.org/wiki/Internet_protocol_suite) [socket](https://en.wikipedia.org/wiki/Stream_socket) connections to allow the attacker access to the shell on the target machine. Such shellcode can be categorized based on how this connection is set up: if the shellcode establishes the connection, it is called a "reverse shell" or a *connect-back* shellcode because the shellcode *connects back* to the attacker's machine. On the other hand, if the attacker establishes the connection, the shellcode is called a *bindshell* because the shellcode *binds* to a certain port on the victim's machine. A third, much less common type, is *socket-reuse* shellcode. This type of shellcode is sometimes used when an exploit establishes a connection to the vulnerable process that is not closed before the shellcode is run. The shellcode can then *re-use* this connection to communicate with the attacker. Socket re-using shellcode is more elaborate, since the shellcode needs to find out which connection to re-use and the machine may have many connections open.[[2]](https://en.wikipedia.org/wiki/Shellcode#cite_note-2)

A [firewall](https://en.wikipedia.org/wiki/Firewall_(computer)) can be used to detect outgoing connections made by connect-back shellcode as well as incoming connections made by bindshells. They can therefore offer some protection against an attacker, even if the system is vulnerable, by preventing the attacker from connecting to the shell created by the shellcode. This is one reason why socket re-using shellcode is sometimes used: it does not create new connections and therefore is harder to detect and block.

**Download and execute**[[edit](https://en.wikipedia.org/w/index.php?title=Shellcode&action=edit&section=4)]

*Download and execute* is a type of remote shellcode that [*downloads*](https://en.wikipedia.org/wiki/Downloads)*and*[*executes*](https://en.wikipedia.org/wiki/Execution_(computers)) some form of malware on the target system. This type of shellcode does not spawn a shell, but rather instructs the machine to download a certain executable file off the network, save it to disk and execute it. Nowadays, it is commonly used in [drive-by download](https://en.wikipedia.org/wiki/Drive-by_download) attacks, where a victim visits a malicious webpage that in turn attempts to run such a download and execute shellcode in order to install software on the victim's machine. A variation of this type of shellcode downloads and [loads](https://en.wikipedia.org/wiki/Dynamic_loading) a [library](https://en.wikipedia.org/wiki/Library_(computing)).[[3]](https://en.wikipedia.org/wiki/Shellcode#cite_note-3)[[4]](https://en.wikipedia.org/wiki/Shellcode#cite_note-4) Advantages of this technique are that the code can be smaller, that it does not require the shellcode to spawn a new process on the target system, and that the shellcode does not need code to clean up the targeted process as this can be done by the library loaded into the process.

**Staged**[[edit](https://en.wikipedia.org/w/index.php?title=Shellcode&action=edit&section=5)]

When the amount of data that an attacker can inject into the target process is too limited to execute useful shellcode directly, it may be possible to execute it in stages. First, a small piece of shellcode (stage 1) is executed. This code then downloads a larger piece of shellcode (stage 2) into the process's memory and executes it.

**Egg-hunt**[[edit](https://en.wikipedia.org/w/index.php?title=Shellcode&action=edit&section=6)]

This is another form of *staged* shellcode, which is used if an attacker can inject a larger shellcode into the process but cannot determine where in the process it will end up. Small *egg-hunt* shellcode is injected into the process at a predictable location and executed. This code then searches the process's address space for the larger shellcode (the *egg*) and executes it.[[5]](https://en.wikipedia.org/wiki/Shellcode#cite_note-5)

**Omelette**[[edit](https://en.wikipedia.org/w/index.php?title=Shellcode&action=edit&section=7)]

This type of shellcode is similar to *egg-hunt* shellcode, but looks for multiple small blocks of data (*eggs*) and recombines them into one larger block (the *omelette*) that is subsequently executed. This is used when an attacker can only inject a number of small blocks of data into the process.[[6]](https://en.wikipedia.org/wiki/Shellcode#cite_note-6)

Shellcode execution strategy[[edit](https://en.wikipedia.org/w/index.php?title=Shellcode&action=edit&section=8)]

An exploit will commonly inject a shellcode into the target process before or at the same time as it exploits a vulnerability to gain control over the [program counter](https://en.wikipedia.org/wiki/Program_counter). The program counter is adjusted to point to the shellcode, after which it gets executed and performs its task. Injecting the shellcode is often done by storing the shellcode in data sent over the network to the vulnerable process, by supplying it in a file that is read by the vulnerable process or through the command line or environment in the case of local exploits.

Shellcode encoding[[edit](https://en.wikipedia.org/w/index.php?title=Shellcode&action=edit&section=9)]

Because most processes filter or restrict the data that can be injected, shellcode often needs to be written to allow for these restrictions. This includes making the code small, null-free or [alphanumeric](https://en.wikipedia.org/wiki/Alphanumeric_code). Various solutions have been found to get around such restrictions, including:

* Design and implementation optimizations to decrease the size of the shellcode.
* Implementation modifications to get around limitations in the range of bytes used in the shellcode.
* [Self-modifying code](https://en.wikipedia.org/wiki/Self-modifying_code) that modifies a number of the bytes of its own code before executing them to re-create bytes that are normally impossible to inject into the process.

Since [intrusion detection](https://en.wikipedia.org/wiki/Intrusion_detection) can detect signatures of simple shellcodes being sent over the network, it is often encoded, made self-decrypting or [polymorphic](https://en.wikipedia.org/wiki/Polymorphic_code) to avoid detection.

**Percent encoding**[[edit](https://en.wikipedia.org/w/index.php?title=Shellcode&action=edit&section=10)]

Exploits that target browsers commonly encode shellcode in a JavaScript string using [percent-encoding](https://en.wikipedia.org/wiki/Percent-encoding), escape sequence encoding "\uXXXX" or [entity encoding](https://en.wikipedia.org/wiki/Character_encodings_in_HTML).[[7]](https://en.wikipedia.org/wiki/Shellcode#cite_note-7) Some exploits also obfuscate the encoded shellcode string further to prevent detection by [IDS](https://en.wikipedia.org/wiki/Intrusion_detection).

For example, on the [IA-32](https://en.wikipedia.org/wiki/IA-32) architecture, here's how two [NOP](https://en.wikipedia.org/wiki/NOP_(code)) (no-operation) instructions would look, first unencoded:

90 NOP

90 NOP

|  |  |
| --- | --- |
| **Encoded double-NOPs:** | |
| **percent-encoding** | unescape("%u9090") |
| **unicode literal** | "\u9090" |
| **HTML/XML entity** | "&amp;#x9090;" or "&amp;#37008;" |

This instruction is used in [NOP slides](https://en.wikipedia.org/wiki/NOP_slide).

**Null-free shellcode**[[edit](https://en.wikipedia.org/w/index.php?title=Shellcode&action=edit&section=11)]

Most shellcodes are written without the use of [null](https://en.wikipedia.org/wiki/Null_character) bytes because they are intended to be injected into a target process through [null-terminated strings](https://en.wikipedia.org/wiki/Null-terminated_string). When a null-terminated string is copied, it will be copied up to and including the first null but subsequent bytes of the shellcode will not be processed. When shellcode that contains nulls is injected in this way, only part of the shellcode would be injected, making it incapable of running successfully.

To produce null-free shellcode from shellcode that contains [null](https://en.wikipedia.org/wiki/Null_character) bytes, one can substitute machine instructions that contain zeroes with instructions that have the same effect but are free of nulls. For example, on the [IA-32](https://en.wikipedia.org/wiki/IA-32) architecture one could replace this instruction:

B8 01000000 [MOV](https://en.wikipedia.org/wiki/MOV_(x86_instruction)) EAX,1 // Set the register EAX to 0x000000001

which contains zeroes as part of the literal (1 expands to 0x00000001) with these instructions:

33C0 [XOR](https://en.wikipedia.org/w/index.php?title=XOR_(x86_instruction)&action=edit&redlink=1) EAX,EAX // Set the register EAX to 0x000000000

40 [INC](https://en.wikipedia.org/w/index.php?title=INC_(x86_instruction)&action=edit&redlink=1) EAX // Increase EAX to 0x00000001

which have the same effect but take fewer bytes to encode and are free of nulls.

**Alphanumeric and printable shellcode**[[edit](https://en.wikipedia.org/w/index.php?title=Shellcode&action=edit&section=12)]

*See also:*[*Alphanumeric code*](https://en.wikipedia.org/wiki/Alphanumeric_code)

In certain circumstances, a target process will filter any byte from the injected shellcode that is not a [printable](https://en.wikipedia.org/wiki/Printable_character) or [alphanumeric](https://en.wikipedia.org/wiki/Alphanumeric) character. Under such circumstances, the range of instructions that can be used to write a shellcode becomes very limited. A solution to this problem was published by Rix in [Phrack](https://en.wikipedia.org/wiki/Phrack) 57[[8]](https://en.wikipedia.org/wiki/Shellcode#cite_note-8) in which he showed it was possible to turn any code into alphanumeric code. A technique often used is to create self-modifying code, because this allows the code to modify its own bytes to include bytes outside of the normally allowed range, thereby expanding the range of instructions it can use. Using this trick, a self-modifying decoder can be created that initially uses only bytes in the allowed range. The main code of the shellcode is encoded, also only using bytes in the allowed range. When the output shellcode is run, the decoder can modify its own code to be able to use any instruction it requires to function properly and then continues to decode the original shellcode. After decoding the shellcode the decoder transfers control to it, so it can be executed as normal. It has been shown that it is possible to create arbitrarily complex shellcode that looks like normal text in English.[[9]](https://en.wikipedia.org/wiki/Shellcode#cite_note-9)

**Unicode proof shellcode**[[edit](https://en.wikipedia.org/w/index.php?title=Shellcode&action=edit&section=13)]

Modern programs use [Unicode](https://en.wikipedia.org/wiki/Unicode) strings to allow internationalization of text. Often, these programs will convert incoming [ASCII](https://en.wikipedia.org/wiki/ASCII) strings to Unicode before processing them. Unicode strings encoded in [UTF-16](https://en.wikipedia.org/wiki/UTF-16) use two bytes to encode each character (or four bytes for some special characters). When an [ASCII](https://en.wikipedia.org/wiki/ASCII) ([Latin-1](https://en.wikipedia.org/wiki/Latin-1) in general) string is transformed into UTF-16, a zero byte is inserted after each byte in the original string. Obscou proved in [Phrack](https://en.wikipedia.org/wiki/Phrack) 61[[10]](https://en.wikipedia.org/wiki/Shellcode#cite_note-10) that it is possible to write shellcode that can run successfully after this transformation. Programs that can automatically encode any shellcode into alphanumeric UTF-16-proof shellcode exist, based on the same principle of a small self-modifying decoder that decodes the original shellcode.

Platforms[[edit](https://en.wikipedia.org/w/index.php?title=Shellcode&action=edit&section=14)]

Most shellcode is written in [machine code](https://en.wikipedia.org/wiki/Machine_code) because of the low level at which the vulnerability being exploited gives an attacker access to the process. Shellcode is therefore often created to target one specific combination of [processor](https://en.wikipedia.org/wiki/Central_processing_unit), [operating system](https://en.wikipedia.org/wiki/Operating_system) and [service pack](https://en.wikipedia.org/wiki/Service_pack), called a [platform](https://en.wikipedia.org/wiki/Platform_(computing)). For some exploits, due to the constraints put on the shellcode by the target process, a very specific shellcode must be created. However, it is not impossible for one shellcode to work for multiple exploits, service packs, operating systems and even processors.[[11]](https://en.wikipedia.org/wiki/Shellcode#cite_note-11) Such versatility is commonly achieved by creating multiple versions of the shellcode that target the various platforms and creating a header that branches to the correct version for the platform the code is running on. When executed, the code behaves differently for different platforms and executes the right part of the shellcode for the platform it is running on.

Shellcode analysis[[edit](https://en.wikipedia.org/w/index.php?title=Shellcode&action=edit&section=15)]

Shellcode cannot be executed directly. In order to analyze what a shellcode attempts to do it must be loaded into another process. One common analysis technique is to write a small C program which holds the shellcode as a byte buffer, and then use a function pointer or use inline assembler to transfer execution to it. Another technique is to use an online tool, such as shellcode\_2\_exe, to embed the shellcode into a pre-made executable husk which can then be analyzed in a standard debugger. Specialized shellcode analysis tools also exist, such as the iDefense sclog project which was originally released in 2005 as part of the Malcode Analyst Pack. Sclog is designed to load external shellcode files and execute them within an API logging framework. Emulation based shellcode analysis tools also exist such as the sctest application which is part of the cross platform libemu package. Another emulation based shellcode analysis tool, built around the libemu library, is scdbg which includes a basic debug shell and integrated reporting features.

See also[[edit](https://en.wikipedia.org/w/index.php?title=Shellcode&action=edit&section=16)]

* [Alphanumeric code](https://en.wikipedia.org/wiki/Alphanumeric_code)
* [Computer security](https://en.wikipedia.org/wiki/Computer_security)
* [Buffer overflow](https://en.wikipedia.org/wiki/Buffer_overflow)
* [Exploit (computer security)](https://en.wikipedia.org/wiki/Exploit_(computer_security))
* [Heap overflow](https://en.wikipedia.org/wiki/Heap_overflow)
* [Metasploit Project](https://en.wikipedia.org/wiki/Metasploit_Project)
* [Shell (computing)](https://en.wikipedia.org/wiki/Shell_(computing))
* [Shell shoveling](https://en.wikipedia.org/wiki/Shell_shoveling)
* [Stack buffer overflow](https://en.wikipedia.org/wiki/Stack_buffer_overflow)
* [Vulnerability (computing)](https://en.wikipedia.org/wiki/Vulnerability_(computing))

References[[edit](https://en.wikipedia.org/w/index.php?title=Shellcode&action=edit&section=17)]

1. [**^**](https://en.wikipedia.org/wiki/Shellcode#cite_ref-1) *Foster, James C.; and Price, Mike (April 12, 2005).*[*Sockets, Shellcode, Porting, & Coding: Reverse Engineering Exploits and Tool Coding for Security Professionals*](https://books.google.com/books?id=ZNI5dvBSfZoC&printsec=frontcover&q=)*. Elsevier Science & Technology Books.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*1-59749-005-9*](https://en.wikipedia.org/wiki/Special:BookSources/1-59749-005-9)*.*
2. [**^**](https://en.wikipedia.org/wiki/Shellcode#cite_ref-2) *BHA (6 June 2013).*[*"Shellcode/Socket-reuse"*](http://www.blackhatlibrary.net/Shellcode/Socket-reuse)*. Retrieved 2013-06-07.*
3. [**^**](https://en.wikipedia.org/wiki/Shellcode#cite_ref-3) *SkyLined (11 January 2010).*[*"Download and LoadLibrary shellcode released"*](https://web.archive.org/web/20100123014637/http:/skypher.com/index.php/2010/01/11/download-and-loadlibrary-shellcode-released/)*. Archived from*[*the original*](http://skypher.com/index.php/2010/01/11/download-and-loadlibrary-shellcode-released/)*on 23 January 2010. Retrieved 2010-01-19.*
4. [**^**](https://en.wikipedia.org/wiki/Shellcode#cite_ref-4) *SkyLined (11 January 2010).*[*"Download and LoadLibrary shellcode for x86 Windows"*](https://code.google.com/p/w32-dl-loadlib-shellcode/)*. Retrieved 2010-01-19.*
5. [**^**](https://en.wikipedia.org/wiki/Shellcode#cite_ref-5) *Skape (9 March 2004).*[*"Safely Searching Process Virtual Address Space"*](http://www.hick.org/code/skape/papers/egghunt-shellcode.pdf)*(PDF). nologin. Retrieved 2009-03-19.*
6. [**^**](https://en.wikipedia.org/wiki/Shellcode#cite_ref-6) *SkyLined (16 March 2009).*[*"w32 SEH omelet shellcode"*](https://web.archive.org/web/20090323030636/http:/skypher.com/wiki/index.php?title=Shellcode%2Fw32_SEH_omelet_shellcode)*. Skypher.com. Archived from*[*the original*](http://skypher.com/wiki/index.php?title=Shellcode/w32_SEH_omelet_shellcode)*on 23 March 2009. Retrieved 2009-03-19.*
7. [**^**](https://en.wikipedia.org/wiki/Shellcode#cite_ref-7) [JavaScript large unescape](http://www.iss.net/security_center/reference/vuln/JavaScript_Large_Unescape.htm) IBM internet security systems
8. [**^**](https://en.wikipedia.org/wiki/Shellcode#cite_ref-8) *Rix (8 November 2001).*[*"Writing ia32 alphanumeric shellcodes"*](http://www.phrack.org/issues.html?issue=57&id=15#article)*. Phrack. Retrieved 2008-02-29.*
9. [**^**](https://en.wikipedia.org/wiki/Shellcode#cite_ref-9) *Mason, Joshua; Small, Sam; Monrose, Fabian; MacManus, Greg (November 2009).*[*"English Shellcode"*](http://www.cs.jhu.edu/~sam/ccs243-mason.pdf)*(PDF). Retrieved 2010-01-10.*
10. [**^**](https://en.wikipedia.org/wiki/Shellcode#cite_ref-10) *Obscou (13 August 2003).*[*"Building IA32 'Unicode-Proof' Shellcodes"*](http://www.phrack.org/issues.html?issue=61&id=11#article)*. Phrack. Retrieved 2008-02-29.*
11. [**^**](https://en.wikipedia.org/wiki/Shellcode#cite_ref-11) *Eugene (11 August 2001).*[*"Architecture Spanning Shellcode"*](http://www.phrack.org/issues.html?issue=57&id=14#article)*. Phrack. Retrieved 2008-02-29.*

External links[[edit](https://en.wikipedia.org/w/index.php?title=Shellcode&action=edit&section=18)]

* [Shell-Storm](http://www.shell-storm.org/shellcode/) Database of shellcodes Multi-Platform.
* [An introduction to buffer overflows and shellcode](http://www.phrack.org/issues.html?issue=49&id=14#article)
* [The Basics of Shellcoding](http://www.infosecwriters.com/text_resources/pdf/basics_of_shellcoding.pdf) (PDF) An overview of [x86](https://en.wikipedia.org/wiki/X86) shellcoding by [Angelo Rosiello](http://www.rosiello.org/)
* [An introduction to shellcode development](https://web.archive.org/web/20120109070051/http:/goodfellas.shellcode.com.ar/docz/bof/Writing_shellcode.html)
* [Contains x86 and non-x86 shellcode samples and an online interface for automatic shellcode generation and encoding, from the Metasploit Project](https://web.archive.org/web/20080302111910/http:/www.metasploit.com/shellcode/)
* [a shellcode archive, sorted by Operating system](https://web.archive.org/web/20060619025456/http:/www.linux-secure.com/endymion/shellcodes/).
* [Microsoft Windows and Linux shellcode design tutorial going from basic to advanced](https://web.archive.org/web/20061112203748/http:/www.milw0rm.com/papers/11).
* [Windows and Linux shellcode tutorial containing step by step examples](http://www.vividmachines.com/shellcode/shellcode.html).
* [Designing shellcode demystified](http://www.enderunix.org/docs/en/sc-en.txt)
* [ALPHA3](https://code.google.com/p/alpha3/) A shellcode encoder that can turn any shellcode into both Unicode and ASCII, uppercase and mixedcase, alphanumeric shellcode.
* [Writing Small shellcode by Dafydd Stuttard](https://web.archive.org/web/20061115040739/http:/www.ngssoftware.com/research/papers/WritingSmallShellcode.pdf) A whitepaper explaining how to make shellcode as small as possible by optimizing both the design and implementation.
* [Writing IA32 Restricted Instruction Set Shellcode Decoder Loops by SkyLined](http://skypher.com/wiki/index.php?title=Www.edup.tudelft.nl/~bjwever/whitepaper_shellcode.html.php) A whitepaper explaining how to create shellcode when the bytes allowed in the shellcode are very restricted.
* [BETA3](https://code.google.com/p/beta3/) A tool that can encode and decode shellcode using a variety of encodings commonly used in exploits.
* [Smallest GNU/Linux x86 exec(”/bin/sh”,0,0) Stable asm (21 bytes)](https://github.com/geyslan/SLAE/blob/master/4th.assignment/tiny_execve_sh.asm/)
* [Smallest GNU/Linux x86 exec(”/bin/sh”,0,0) Stable shellcode (21 bytes)](https://github.com/geyslan/SLAE/blob/master/4th.assignment/tiny_execve_sh_shellcode.c)
* [Smallest GNU/Linux x86 setuid(0) & exec(”/bin/sh”,0,0) Stable shellcode](http://safetybits.net/2008/11/26/gnulinux-setuid0-execbinsh00-stable/)
* [Shellcode 2 Exe](http://sandsprite.com/shellcode_2_exe.php) - Online converter to embed shellcode in exe husk
* [Sclog](https://github.com/dzzie/sclog) - Updated build of the iDefense sclog shellcode analysis tool (Windows)
* [Libemu](https://archive.is/20130219020328/http:/libemu.carnivore.it/) - emulation based shellcode analysis library (\*nix/Cygwin)
* [Scdbg](http://sandsprite.com/blogs/index.php?uid=7&pid=152) - shellcode debugger built around libemu emulation library (\*nix/Windows)

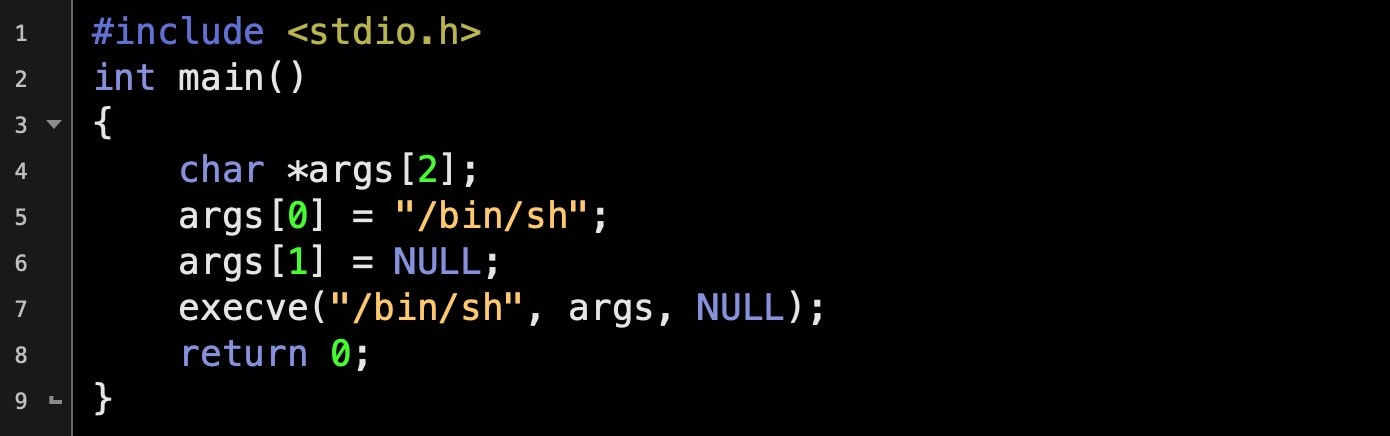
You know all about [hashes](https://www.sentinelone.com/blog/what-is-hash-how-does-it-work/) in cybersecurity and how to [decode Base64](https://www.sentinelone.com/blog/guide-encode-decoded-base64/); you’re likely also familiar with [steganography](https://www.sentinelone.com/blog/hiding-code-inside-images-malware-steganography/), and maybe you can even recite the [history](https://www.sentinelone.com/blog/history-of-cyber-security/) of cybersecurity and the development of [EDR](https://www.sentinelone.com/blog/short-history-of-edr/). But how about explaining the malicious use of shellcode? You know it has nothing to do with shell scripts or shell scripting languages like Bash, but can you hold your own talking about what shellcode really is, and why it’s such a great tool for attackers?



Not sure? No problem. We’ve got just the post for you. In the next ten minutes, we’ll take you through the basics of shellcode, what it is, how it works and how hackers use it as malicious input.

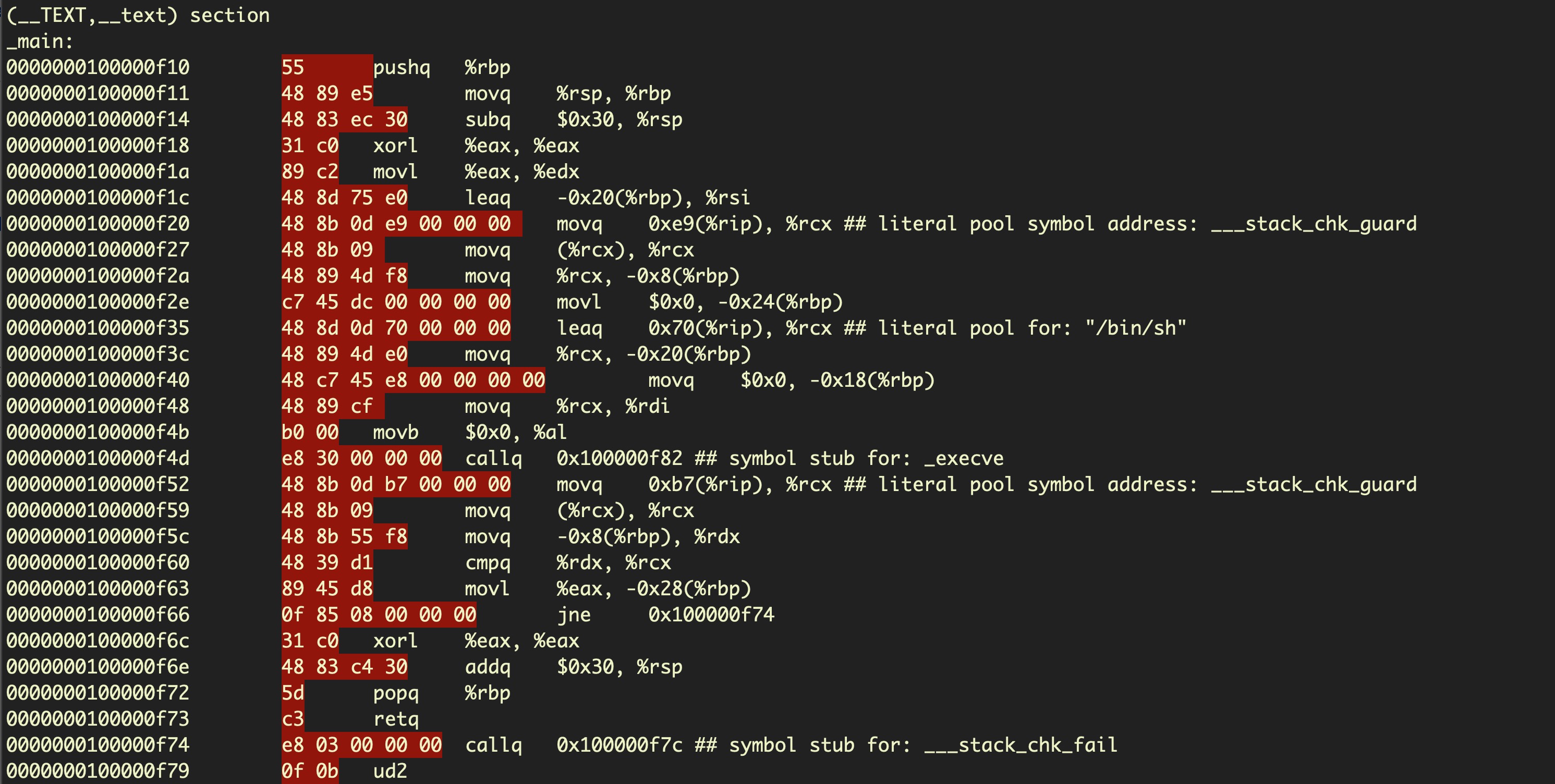
## What is ShellCode?

We know shellcode has nothing to do with shell scripting, so why the name? The term “shellcode” was historically used to describe code executed by a target program due to a vulnerability exploit and used to open a remote shell – that is, an instance of a command line interpreter – so that an attacker could use that shell to further interact with the victim’s system. It usually only takes a few lines of code to spawn a new shell process, so popping shells is a very lightweight, efficient means of attack, so long as we can provide the right input to a target program.



Standard C code like that above will pop a shell. You can compile and run it simply enough in an editor like [Geany](https://www.geany.org/). It’s possible to turn small programs such as the one above into input strings that can be supplied to a vulnerable program to achieve the same effect. The bonus for attackers is that if your target program is running with elevated privileges, your newly spawned shell will inherit those same privileges, too.

Creating a shellcode string from ordinary code like that above requires using a disassembler to reveal the assembly “underneath” the compiled C. We can do that in any disassembler such as IDA, Ghidra, OllyDbg, Radare2 or otool on macOS. Here’s what the program above looks like in a disassembler.



The opcodes that we would need to create our shellcode are highighted in red. To the right of those are the same instructions in the more human-readable assembly language.

Once we have our opcodes, we need to put them into a format that can be used as string input to another program. This involves concatenating the opcodes into a string and prepending each hex byte with \x to produce a string with the following format:

\x55\x48\x89\xe5\x48\x83\xec\x30\x31\xc0\x89\xc2\x48\x8d\x75\xe0\x48\x8b\x3b\x0d\xe9\x...

## How To Create Shellcode

So far so good, but there’s a problem. Our shellcode instructions are not allowed to contain zeroes, as any zero character in our input string will be interpreted by the target program as a [null-terminator](https://en.wikipedia.org/wiki/Null-terminated_string) character, and the rest of the shellcode will consequently be discarded. As we can see, our program above has a great many null bytes.

In order to solve this problem and create well-formed shellcode, we need to  replace any instructions containing null bytes with other instructions. Doing so is much easier if we code directly in an assembly language like [NASM](https://en.wikipedia.org/wiki/Netwide_Assembler) rather than starting in a language like C and extracting the assembly. Let’s look at another example that has been specially-crafted to avoid the null-bytes problem. Below is the disassembly for a [similar problem](https://www.exploit-db.com/shellcodes/46907) that also pops a shell using execve(), but the assembly is much smaller and more efficient than before.



Notice at line 8 the hexadecimal 48 31 f6, which represents the instructions for the following assembly:

xor %rsi, %rsi

The use of [XOR](https://hackernoon.com/xor-the-magical-bit-wise-operator-24d3012ed821) here is an example of sidestepping the restriction of not being able to use zeroes we mentioned above. This particular program needs to push the integer 0 onto the [stack](https://en.wikipedia.org/wiki/Stack_(abstract_data_type)). To do so, it first loads 0 into the CPU’s %rsi [register](https://softwareengineering.stackexchange.com/questions/209760/what-are-cpu-registers). The natural way to do that would be:

mov $0x0, %rsi

But as we can see if we input that into an [online disassembler](https://defuse.ca/online-x86-assembler.htm#disassembly), that produces raw hex with zeroes in the instruction operand.

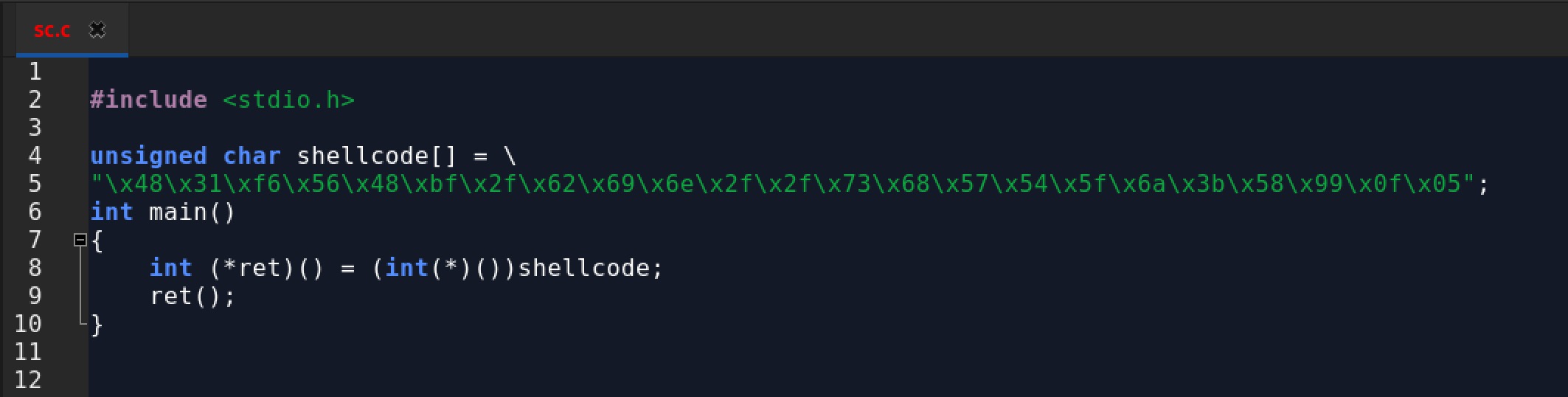
4889342500000000

We can get around that by xoring the value of %rsi with itself. When both input values of XOR are the same, the result will be zero, but the instruction doesn’t require any zeroes in the raw hex.

We can now produce a well-formed shellcode string that contains our complete program:

\x48\x31\xf6\x56\x48\xbf\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x57\x54\x5f\x6a\x3b\x58\x99\x0f\x05

Let’s create a program that executes our shellcode to see that it works:

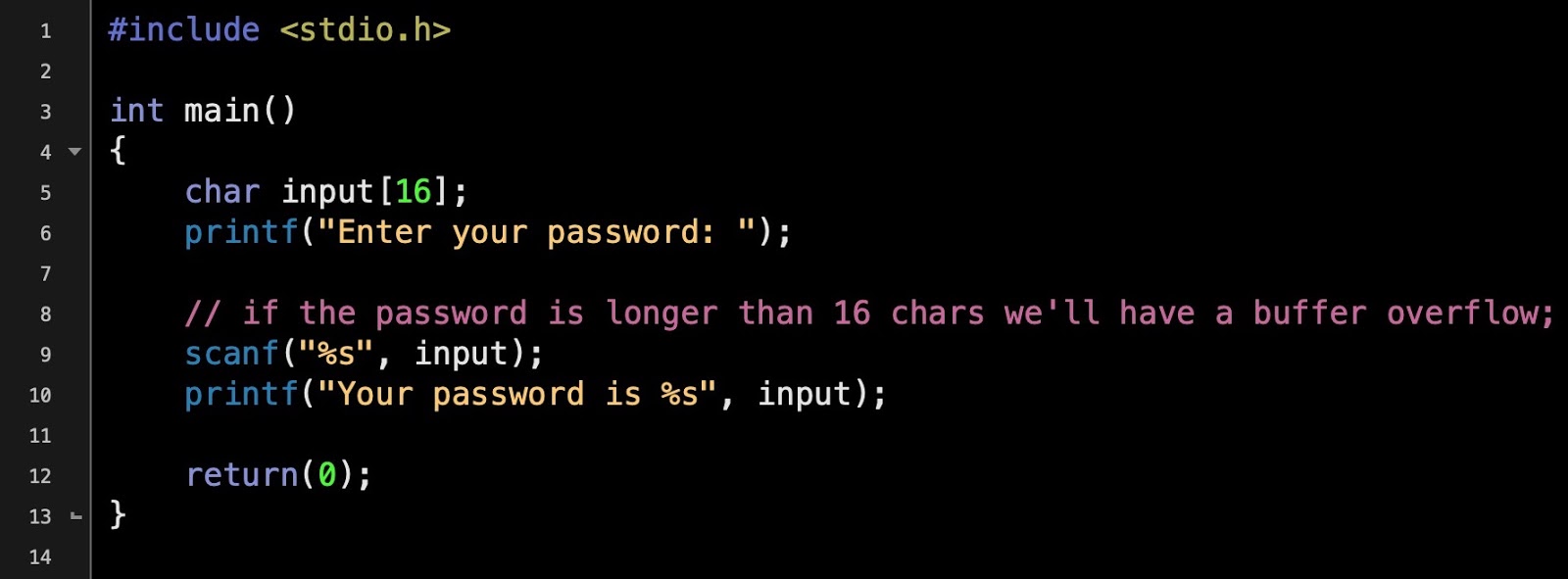


## Finding Vulnerable Programs

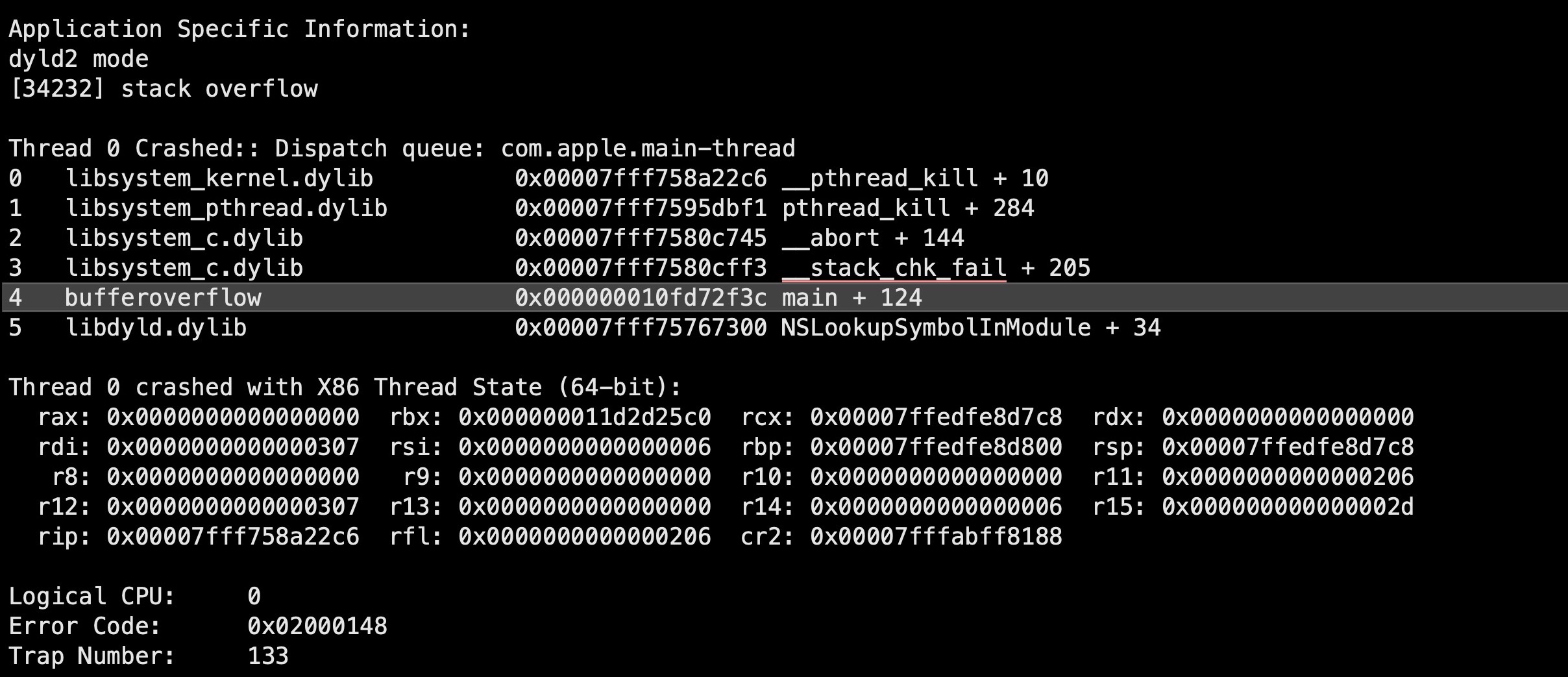
So now we have some shellcode and a proof of concept, but finding vulnerable programs that we can feed our shellcode to isn’t a simple matter. One way is through [reverse engineering](https://www.sentinelone.com/blog/how-to-reverse-macos-malware-part-one/) a program, [fuzzing](https://en.wikipedia.org/wiki/Fuzzing) and experimenting in the hope of finding a target program that mishandles certain edge cases of input data. In cases where the program code mishandles some unexpected form of input, this can sometimes be used to alter the program execution flow and either make it crash or allow us to run our own instructions delivered by the shellcode. One common programming error that can often be used to achieve this is a buffer overflow.

## What is a Buffer Overflow?

A buffer overflow occurs when a program writes data into memory that is larger than the area of memory, the buffer, the program has reserved for it, thus overwriting some unrelated program data. This is a programming error, as code should always check first that the length of any input data will not exceed the size of the buffer that’s been allocated. When this happens the program may crash, but specially-crafted input like our shellcode may instead allow an attacker to execute their own code. Here’s a simple example of a buffer overflow waiting to happen.



The program reserves 16 bytes of memory for the input, but the size of the input is never checked. If the user enters a string longer than 16 bytes, the data will overwrite adjacent memory – a buffer overflow. The image below shows what happens if you try to execute the above program and supply it with input greater than 16 bytes:

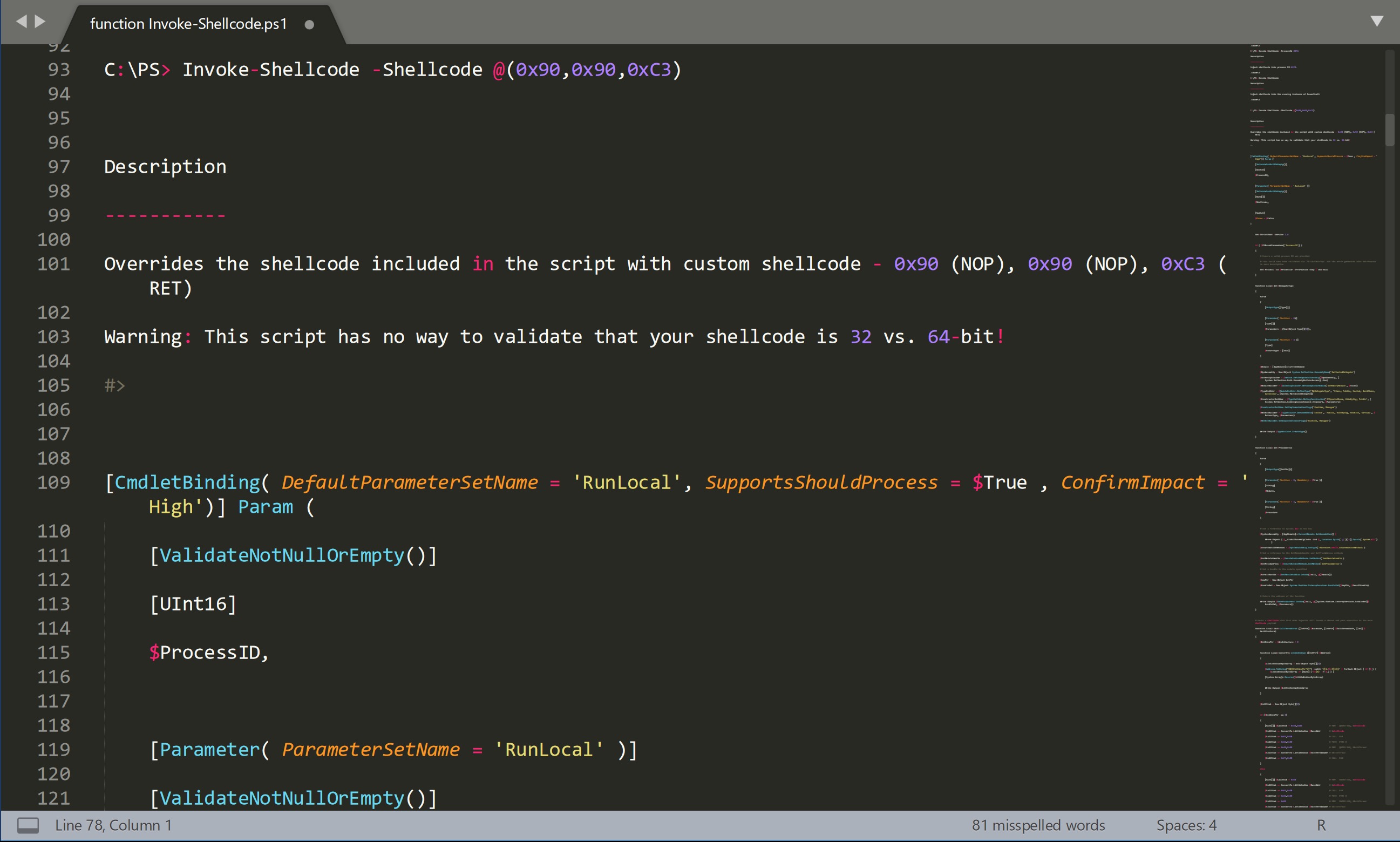


However, just causing a buffer overflow in a program isn’t on its own much use to attackers, unless all they want to do is bring the application to a crashing halt. While that would represent a win of sorts for attackers whose objective is some kind of denial of service attack, the greater prize in most cases is not just causing the overflow but using it as a means to take control of execution.

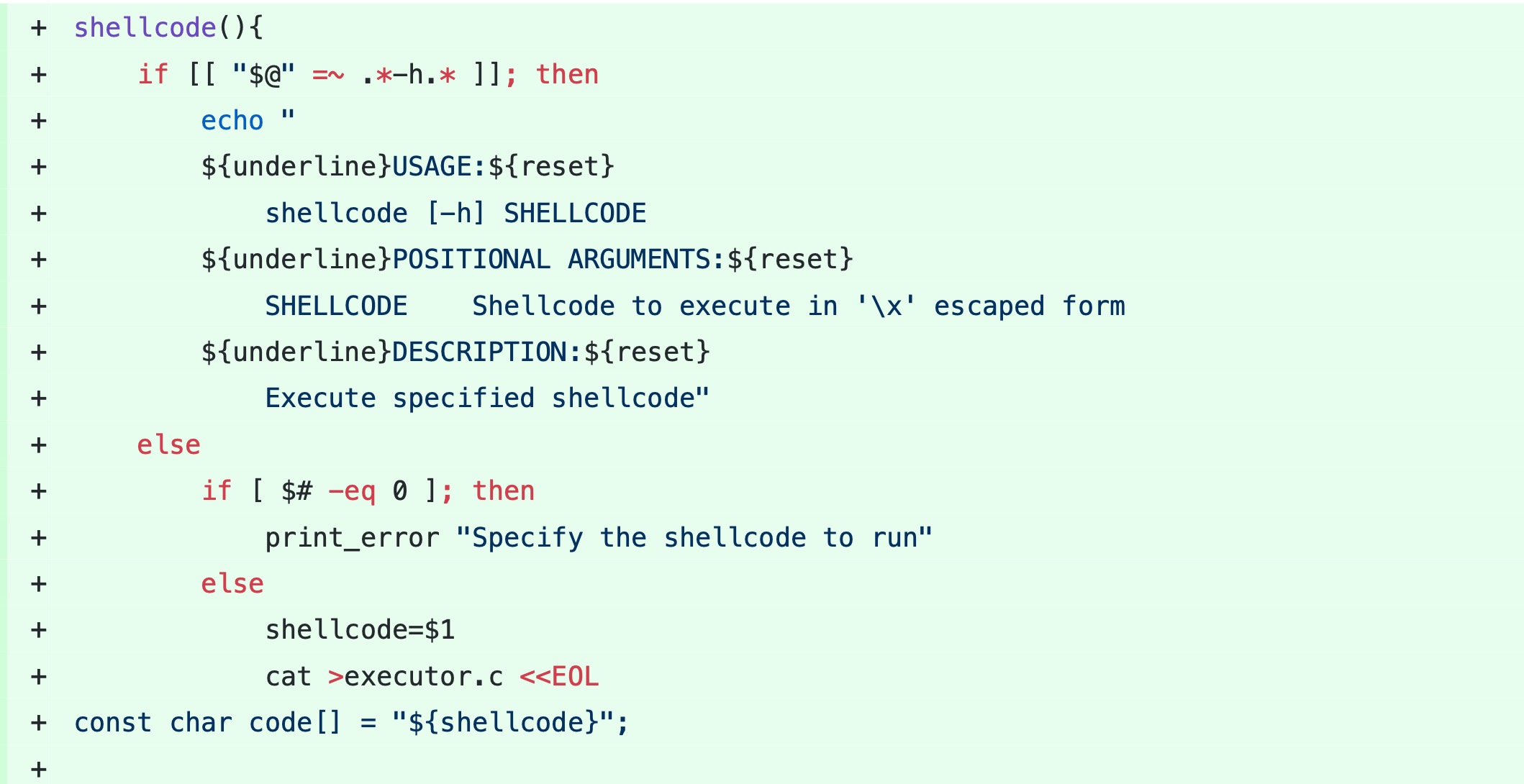
Taking control of execution is a complex matter, but essentially involves determining precisely how much data we need to write to overflow the buffer sufficiently to ensure our shellcode is executed. This requires writing our own code both at a given address and ensuring that the target program’s current function – that block of code which is handling our shellcode string and deciding what should happen next – returns to the address where our exploit code is waiting. If we can control that, we have a good chance of getting our exploit to successfully execute.

## Shellcode  & Exploitation Kits

While writing your own shellcode is a task that requires a certain amount of skill, including knowledge of assembly, attackers have a variety of tools available to help them out. There are publicly available post exploitation kits like [Metasploit](https://www.metasploit.com/) and [PowerSploit](https://github.com/PowerShellMafia/PowerSploit) that offer things like encoders to help generate compliant shellcode, tools to create payloads and functions that can inject shellcode directly into processes.



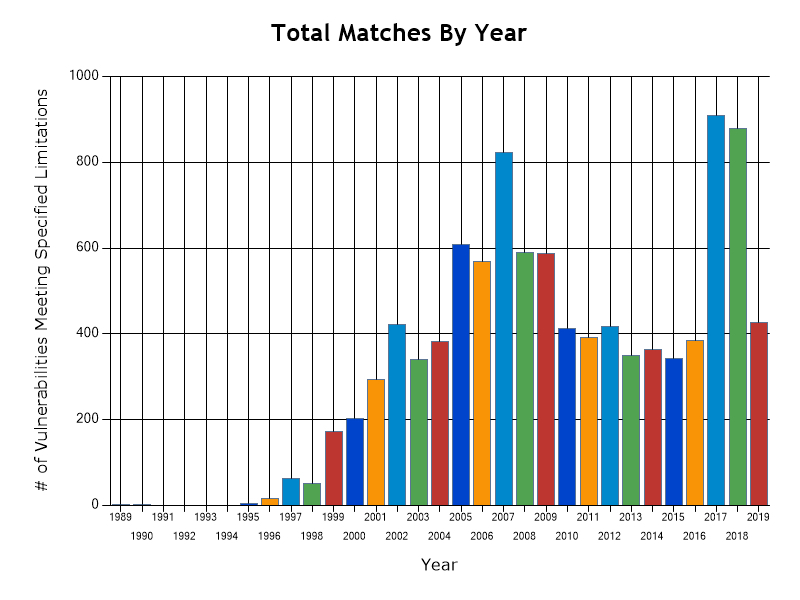
On Linux and macOS, even a simple bash post-exploit kit like [Bashark](https://github.com/TheSecondSun/Bashark) will offer a function to execute shellcode.



Examples of pre-made shellcode can readily be found across the internet, including in resources for penetration testers and red teamers like the [Exploit Database](https://www.exploit-db.com/), although real-world attacks will often require some degree of customization to ensure the shellcode is suited to the target program, execution environment and attacker objectives.

## Protecting Against Shellcode

You would think that input mishandling resulting in buffer overflows, which have been known about for decades, would be becoming rarer, but in fact the opposite is true. Statistics from the CVE database at NIST show that vulnerabilities caused by buffer overflows increased dramatically during 2017 and 2018. The number known for this year is already higher than every year from 2010 to 2016, and we still have almost 5 months of the year left to go.



Clearly, there’s a lot of unsafe code out there, and the only real way you can protect yourself from exploits that inject shellcode into vulnerable programs is with a [multi-layered security solution](https://www.sentinelone.com/platform/) that can not only use [Firewall](https://www.sentinelone.com/blog/feature-spotlight-firewall-control/) or [Device](https://www.sentinelone.com/blog/feature-spotlight-device-control/) controls to protect your software stack from unwanted connections, but also that uses static and behavioral AI to catch malicious activity both before and on execution. With a comprehensive security solution that uses [machine learning](https://www.sentinelone.com/blog/really-matters-machine-learning/) to identify malicious behavior, attacks by shellcode are seen just like any other attack and stopped before they can do any damage.

## Conclusion

In this post, we’ve taken a look at what shellcode is and how hackers can use it as malicious input to exploit vulnerabilities in legitimate programs. Despite the long history of the dangers of buffer overflows, even today we see an increasing number of CVEs being attributed to this vector. Looking on the bright side, attacks that utilize shellcode can be stopped with a good security solution. On top of that, if you find yourself in the midst of a thread or a chat concerning shellcode and malicious input, you should now be able to parti

cipate and see what more you can learn from, or share with, others!

**Shellcodes database for study cases**

**Description**

Although these kinds of shellcode presented on this page are rarely used for real exploitations, this page lists some of them for study cases and proposes an API to search specific ones. Thanks all for your contributions of this database but we stopped to accept shellcodes. To learn modern exploitation, checkout how to the [Return Oriented Programming](http://shell-storm.org/talks/ROP_course_lecture_jonathan_salwan_2014.pdf) works.

**API**

This is very straightforward to communicate with this API. Just send a simple **GET** method. The "**s**" argument contains your keyword.

http://shell-storm.org/api/?s=<keyword>

Use "**\***" for multiple keywords search.

/?s=<keyword1>\*<keyword2>\*<keyword3>

The output should be like this:

<auteur 1>::::<plateforme 1>::::<shellcode title 1>::::<shellcode id 1>::::<shellcode url 1>

<auteur 2>::::<plateforme 2>::::<shellcode title 2>::::<shellcode id 2>::::<shellcode url 2>

<auteur 3>::::<plateforme 3>::::<shellcode title 3>::::<shellcode id 3>::::<shellcode url 3>

For more information about how can you use it, read this [shell-storm API](http://shell-storm.org/files/shell-storm-api.py) python script. You can also find this API utilization in the [Peda GDB](https://github.com/longld/peda) project *(shellcode command)*.

**AIX**

* [Aix - execve /bin/sh - 88 bytes](http://shell-storm.org/shellcode/files/shellcode-132.php) *by Georgi Guninski*

**Alpha**

* [Alpha - /bin/sh - 80 bytes](http://shell-storm.org/shellcode/files/shellcode-134.php) *by Lamont Granquist*
* [Alpha - execve() - 112 bytes](http://shell-storm.org/shellcode/files/shellcode-136.php) *by n/a*
* [Alpha - setuid() - 156 bytes](http://shell-storm.org/shellcode/files/shellcode-135.php) *by n/a*

**BSD**

* [BSD/32bits - Passive Connection - 126 bytes](http://shell-storm.org/shellcode/files/shellcode-90.php) *by Scrippie*
* [BSD/ppc - execve(/bin/sh) - 128 bytes](http://shell-storm.org/shellcode/files/shellcode-107.php) *by Palante*
* [BSD/x86 - setreuid(geteuid(), geteuid()) and execve(/bin/sh, /bin/sh, 0)](http://shell-storm.org/shellcode/files/shellcode-814.php) *by Jihyeog Lim*
* [BSD/x86 - setuid/execve - 30 bytes](http://shell-storm.org/shellcode/files/shellcode-95.php) *by Marco Ivaldi*
* [BSD/x86 - setuid/portbind - 94 bytes](http://shell-storm.org/shellcode/files/shellcode-94.php) *by Marco Ivaldi*
* [BSD/x86 - break chroot - 45 bytes](http://shell-storm.org/shellcode/files/shellcode-356.php) *by Matias Sedalo*
* [BSD/x86 - cat /etc/master.passwd & mail root@localhost - 92 bytes](http://shell-storm.org/shellcode/files/shellcode-91.php) *by Matias Sedalo*
* [BSD/x86 - execve(/bin/sh) & setuid(0) - 29 bytes](http://shell-storm.org/shellcode/files/shellcode-92.php) *by Matias Sedalo*
* [BSD/x86 - bindshell on port 2525 - 167 bytes](http://shell-storm.org/shellcode/files/shellcode-601.php) *by beosroot*
* [BSD/x86 - execve /bin/sh Crypt /bin/sh - 49 bytes](http://shell-storm.org/shellcode/files/shellcode-362.php) *by dev0id*
* [BSD/x86 - execve(/bin/sh) - 27 bytes](http://shell-storm.org/shellcode/files/shellcode-93.php) *by n0gada*
* [BSD/x86 - Connect Back Port 6969 - 133 bytes](http://shell-storm.org/shellcode/files/shellcode-676.php) *by Marcetam*
* [BSD/x86 - back-connect TCP/2222 - 93 bytes](http://shell-storm.org/shellcode/files/shellcode-360.php) *by dev0id*

**Cisco**

* [Cisco IOS - Connectback shellcode v1.0](http://shell-storm.org/shellcode/files/shellcode-144.php) *by Gyan Chawdhary*
* [Cisco IOS - Tiny shellcode v1.0](http://shell-storm.org/shellcode/files/shellcode-142.php) *by Gyan Chawdhary*
* [Cisco IOS - Bind shellcode v1.0](http://shell-storm.org/shellcode/files/shellcode-143.php) *by Varun Uppal*

**Sco**

* [Sco/x86 - execve(/bin/sh, ..., NULL) - 43 bytes](http://shell-storm.org/shellcode/files/shellcode-131.php) *by minervini*

**FreeBSD**

**Intel x86-64**

* [FreeBSD/x86-64 - execve - 28 bytes](http://shell-storm.org/shellcode/files/shellcode-866.php) *by Gitsnik*
* [FreeBSD/x86-64 - bind\_tcp with passcode - 127 bytes](http://shell-storm.org/shellcode/files/shellcode-865.php) *by Gitsnik*
* [FreeBSD/x86-64 - exec(/bin/sh) Shellcode - 31 bytes](http://shell-storm.org/shellcode/files/shellcode-106.php) *by Hack'n Roll*
* [FreeBSD/x86-64 - execve /bin/sh shellcode 34 bytes](http://shell-storm.org/shellcode/files/shellcode-104.php) *by Hack'n Roll*
* [FreeBSD/x86-64 - Execve /bin/sh - Anti-Debugging](http://shell-storm.org/shellcode/files/shellcode-103.php) *by c0d3\_z3r0*

**Intel x86**

* [FreeBSD/x86 - execve /tmp/sh - 34 bytes](http://shell-storm.org/shellcode/files/shellcode-100.php) *by Claes M. Nyberg*
* [FreeBSD/x86 - execve /bin/sh 23 bytes](http://shell-storm.org/shellcode/files/shellcode-170.php) *by IZ*
* [FreeBSD/x86 - reboot(RB\_AUTOBOOT) - 7 bytes](http://shell-storm.org/shellcode/files/shellcode-101.php) *by IZ*
* [FreeBSD/x86 - bind port:4883 with auth shellcode](http://shell-storm.org/shellcode/files/shellcode-169.php) *by MahDelin*
* [FreeBSD/x86 - connect back /bin/sh. 81 bytes](http://shell-storm.org/shellcode/files/shellcode-747.php) *by Tosh*
* [FreeBSD/x86 - execv(/bin/sh) - 23 bytes](http://shell-storm.org/shellcode/files/shellcode-749.php) *by Tosh*
* [FreeBSD/x86 - portbind shell + fork - 111 bytes](http://shell-storm.org/shellcode/files/shellcode-748.php) *by Tosh*
* [FreeBSD/x86 - 8.0-RELEASE - //sbin/pfctl -F all Shellcode 47 Bytes](http://shell-storm.org/shellcode/files/shellcode-609.php) *by antrhacks*
* [FreeBSD/x86 - encrypted shellcode /bin/sh 48 bytes](http://shell-storm.org/shellcode/files/shellcode-168.php) *by c0d3\_z3r0*
* [FreeBSD/x86 - kldload /tmp/o.o - 74 bytes](http://shell-storm.org/shellcode/files/shellcode-172.php) *by dev0id*
* [FreeBSD/x86 - /bin/sh - 23 bytes](http://shell-storm.org/shellcode/files/shellcode-675.php) *by marcetam*
* [FreeBSD/x86 - execve /bin/sh 37 bytes](http://shell-storm.org/shellcode/files/shellcode-171.php) *by preedator*
* [FreeBSD/x86 - portbind shellcode - 167 bytes](http://shell-storm.org/shellcode/files/shellcode-570.php) *by sbz*
* [FreeBSD/x86 - execve(/bin/cat & /etc/master.passwd) - 65 bytes](http://shell-storm.org/shellcode/files/shellcode-99.php) *by sm4x*
* [FreeBSD/x86 - reverse connect dl(shellcode) and execute, exit - 90 bytes](http://shell-storm.org/shellcode/files/shellcode-167.php) *by sm4x*
* [FreeBSD/x86 - reverse portbind /bin/sh - 89 bytes](http://shell-storm.org/shellcode/files/shellcode-98.php) *by sm4x*
* [FreeBSD/x86 - setuid(0)&execve({//sbin/ipf,-Faa,0},0); - 57 bytes](http://shell-storm.org/shellcode/files/shellcode-97.php) *by sm4x*
* [FreeBSD/x86 - connect back.send.exit /etc/passwd - 112 bytes](http://shell-storm.org/shellcode/files/shellcode-166.php) *by suN8Hclf*
* [FreeBSD/x86 - kill all processes - 12 bytes](http://shell-storm.org/shellcode/files/shellcode-102.php) *by suN8Hclf*
* [FreeBSD/x86 - setreuid(0, 0) & execve(pfctl -d) - 56 bytes](http://shell-storm.org/shellcode/files/shellcode-96.php) *by suN8Hclf*
* [FreeBSD/x86 - bind sh port 41254 - 115 bytes](http://shell-storm.org/shellcode/files/shellcode-674.php) *by zillion*
* [FreeBSD/x86 - reboot() - 15 bytes](http://shell-storm.org/shellcode/files/shellcode-677.php) *by zillion*

**Hp-Ux**

* [Hp-Ux - execve(/bin/sh) - 58 bytes](http://shell-storm.org/shellcode/files/shellcode-133.php) *by K2*

**Irix**

* [Irix - execve(/bin/sh -c) - 72 bytes](http://shell-storm.org/shellcode/files/shellcode-139.php) *by n/a*
* [Irix - execve(/bin/sh) - 43 bytes](http://shell-storm.org/shellcode/files/shellcode-141.php) *by n/a*
* [Irix - Bind Port - 364 bytes](http://shell-storm.org/shellcode/files/shellcode-138.php) *by scut/teso*
* [Irix - execve(/bin/sh) - 68 bytes](http://shell-storm.org/shellcode/files/shellcode-140.php) *by scut/teso*
* [Irix - stdin-read shellcode - 40 bytes](http://shell-storm.org/shellcode/files/shellcode-137.php) *by scut/teso*

**Linux**

**ARM**

* [Linux/ARM - execve("/bin/sh", NULL, 0) - 34 bytes](http://shell-storm.org/shellcode/files/shellcode-904.php) *by Jonathan 'dummys' Borgeaud*
* [Linux/ARM - Add map in /etc/hosts file - 79 bytes](http://shell-storm.org/shellcode/files/shellcode-901.php) *by Osanda Malith Jayathissa*
* [Linux/ARM - chmod("/etc/passwd", 0777) - 39 bytes](http://shell-storm.org/shellcode/files/shellcode-853.php) *by gunslinger\_*
* [Linux/ARM - creat("/root/pwned", 0777) - 39 bytes](http://shell-storm.org/shellcode/files/shellcode-854.php) *by gunslinger\_*
* [Linux/ARM - execve("/bin/sh", [], [0 vars]) - 35 bytes](http://shell-storm.org/shellcode/files/shellcode-855.php) *by gunslinger\_*
* [Linux/ARM - Bind Connect UDP Port 68](http://shell-storm.org/shellcode/files/shellcode-729.php) *by Daniel Godas-Lopez*
* [Linux/ARM - Bindshell port 0x1337](http://shell-storm.org/shellcode/files/shellcode-730.php) *by Daniel Godas-Lopez*
* [Linux/ARM - Loader Port 0x1337](http://shell-storm.org/shellcode/files/shellcode-728.php) *by Daniel Godas-Lopez*
* [Linux/ARM - ifconfig eth0 and Assign Address](http://shell-storm.org/shellcode/files/shellcode-727.php) *by Daniel Godas-Lopez*
* [Linux/ARM - chmod(/etc/shadow, 0777) Shellcode - 35 Bytes](http://shell-storm.org/shellcode/files/shellcode-668.php) *by Florian Gaultier*
* [Linux/ARM - polymorphic chmod(/etc/shadow, 0777) - 84 Bytes](http://shell-storm.org/shellcode/files/shellcode-670.php) *by Florian Gaultier*
* [Linux/ARM - Polymorphic execve("/bin/sh", ["/bin/sh"], NULL); - XOR - 78 bytes](http://shell-storm.org/shellcode/files/shellcode-671.php) *by Jonathan Salwan*
* [Linux/ARM - execve(/bin/sh, /bin/sh, 0) - 30 bytes](http://shell-storm.org/shellcode/files/shellcode-665.php) *by Jonathan Salwan*
* [Linux/ARM - execve(/bin/sh, [0], [0 vars]) - 27 bytes](http://shell-storm.org/shellcode/files/shellcode-698.php) *by Jonathan Salwan*
* [Linux/ARM - execve(/bin/sh,NULL,0) - 31 bytes](http://shell-storm.org/shellcode/files/shellcode-696.php) *by Jonathan Salwan*
* [Linux/ARM - setuid(0) & execve(/bin/sh, /bin/sh, 0) - 38 bytes](http://shell-storm.org/shellcode/files/shellcode-666.php) *by Jonathan Salwan*
* [Linux/ARM - connect back /bin/sh. 79 bytes](http://shell-storm.org/shellcode/files/shellcode-754.php) *by Neil Klopfenstein*
* [Linux/ARM - chmod(/etc/shadow, 0777) - 41 bytes](http://shell-storm.org/shellcode/files/shellcode-820.php) *by midnitesnake*
* [Linux/ARM - execve(/bin/sh, [0], [0 vars]) - 30 bytes](http://shell-storm.org/shellcode/files/shellcode-819.php) *by midnitesnake*
* [Linux/ARM - reverse\_shell(tcp,10.1.1.2,0x1337)](http://shell-storm.org/shellcode/files/shellcode-821.php) *by midnitesnake*

**Strong ARM**

* [Linux/StrongARM - bind() portshell - 203 bytes](http://shell-storm.org/shellcode/files/shellcode-661.php) *by funkysh*
* [Linux/StrongARM - execve() - 47 bytes](http://shell-storm.org/shellcode/files/shellcode-659.php) *by funkysh*
* [Linux/StrongARM - setuid() - 20 bytes](http://shell-storm.org/shellcode/files/shellcode-660.php) *by funkysh*

**Super-H**

* [Linux/SuperH - sh4 - Bind /bin/sh on port 31337](http://shell-storm.org/shellcode/files/shellcode-774.php) *by Dad`*
* [Linux/SuperH - sh4 execve(/bin/sh, 0, 0) - 19 bytes](http://shell-storm.org/shellcode/files/shellcode-771.php) *by Florian Gaultier*
* [Linux/SuperH - sh4 - setuid(0) ; execve(/bin/sh, NULL, NULL) - 27 bytes](http://shell-storm.org/shellcode/files/shellcode-787.php) *by Jonathan Salwan*

**MIPS**

* [Linux/mips - Reverse Shell Shellcode - 200 bytes](http://shell-storm.org/shellcode/files/shellcode-860.php) *by Jacob Holcomb*
* [Linux/mips - execve(/bin/sh) - 56 bytes](http://shell-storm.org/shellcode/files/shellcode-79.php) *by core*
* [Linux/mips - execve(/bin/sh, \*/bin/sh, 0) - 52 bytes](http://shell-storm.org/shellcode/files/shellcode-782.php) *by entropy*
* [Linux/mips - add user(UID 0) with password - 164 bytes](http://shell-storm.org/shellcode/files/shellcode-793.php) *by rigan*
* [Linux/mips - connect back shellcode (port 0x7a69) - 168 bytes](http://shell-storm.org/shellcode/files/shellcode-794.php) *by rigan*
* [Linux/mips - execve /bin/sh - 48 bytes](http://shell-storm.org/shellcode/files/shellcode-792.php) *by rigan*
* [Linux/mips - reboot() - 32 bytes](http://shell-storm.org/shellcode/files/shellcode-795.php) *by rigan*
* [Linux/mips - execve(/bin/sh,[/bin/sh],[]); - 60 bytes](http://shell-storm.org/shellcode/files/shellcode-80.php) *by vaicebine*
* [Linux/mips - port bind 4919 - 276 bytes](http://shell-storm.org/shellcode/files/shellcode-81.php) *by vaicebine*

**PPC**

* [Linux/ppc - connect back execve /bin/sh - 240 bytes](http://shell-storm.org/shellcode/files/shellcode-87.php) *by Charles Stevenson*
* [Linux/ppc - execve /bin/sh - 60 bytes](http://shell-storm.org/shellcode/files/shellcode-86.php) *by Charles Stevenson*
* [Linux/ppc - read & exec shellcode - 32 bytes](http://shell-storm.org/shellcode/files/shellcode-88.php) *by Charles Stevenson*
* [Linux/ppc - execve /bin/sh - 112 bytes](http://shell-storm.org/shellcode/files/shellcode-89.php) *by Palante*

**Sparc**

* [Linux/sparc - [setreuid(0,0); execve() of /bin/sh] - 64 bytes](http://shell-storm.org/shellcode/files/shellcode-83.php) *by anathema*
* [Linux/sparc - Portbind 8975/tcp - 284 bytes](http://shell-storm.org/shellcode/files/shellcode-84.php) *by killah*
* [Linux/sparc - connect back - 216 bytes](http://shell-storm.org/shellcode/files/shellcode-85.php) *by killah*
* [Linux/sparc - setreuid(0,0)&standard execve() - 72 bytes](http://shell-storm.org/shellcode/files/shellcode-82.php) *by michel kaempf*

**CRISv32**

* [Linux/CRISv32 - Connect Back Shellcode - 189 bytes](http://shell-storm.org/shellcode/files/shellcode-903.php) *by bashis*

**RISC-V64**

* [Linux/RISC-V64 - execve(/bin/sh, NULL, 0) - 34 bytes](http://shell-storm.org/shellcode/files/shellcode-908.php) *by Christina Quast*

**Intel x86-64**

* [Linux/x86-64 - Dynamic null-free reverse TCP shell - 65 bytes](http://shell-storm.org/shellcode/files/shellcode-907.php) *by Philippe Dugre*
* [Linux/x86-64 - execveat("/bin//sh") - 29 bytes](http://shell-storm.org/shellcode/files/shellcode-905.php) *by ZadYree, vaelio and DaShrooms*
* [Linux/x86-64 - Add map in /etc/hosts file - 110 bytes](http://shell-storm.org/shellcode/files/shellcode-896.php) *by Osanda Malith Jayathissa*
* [Linux/x86-64 - Connect Back Shellcode - 139 bytes](http://shell-storm.org/shellcode/files/shellcode-895.php) *by MadMouse*
* [Linux/x86-64 - access() Egghunter - 49 bytes](http://shell-storm.org/shellcode/files/shellcode-894.php) *by Doreth.Z10*
* [Linux/x86-64 - Shutdown - 64 bytes](http://shell-storm.org/shellcode/files/shellcode-892.php) *by Keyman*
* [Linux/x86-64 - Read password - 105 bytes](http://shell-storm.org/shellcode/files/shellcode-891.php) *by Keyman*
* [Linux/x86-64 - Password Protected Reverse Shell - 136 bytes](http://shell-storm.org/shellcode/files/shellcode-890.php) *by Keyman*
* [Linux/x86-64 - Password Protected Bind Shell - 147 bytes](http://shell-storm.org/shellcode/files/shellcode-889.php) *by Keyman*
* [Linux/x86-64 - Add root - Polymorphic - 273 bytes](http://shell-storm.org/shellcode/files/shellcode-888.php) *by Keyman*
* [Linux/x86-64 - Bind TCP stager with egghunter - 157 bytes](http://shell-storm.org/shellcode/files/shellcode-884.php) *by Christophe G*
* [Linux/x86-64 - Add user and password with open,write,close - 358 bytes](http://shell-storm.org/shellcode/files/shellcode-880.php) *by Christophe G*
* [Linux/x86-64 - Add user and password with echo cmd - 273 bytes](http://shell-storm.org/shellcode/files/shellcode-879.php) *by Christophe G*
* [Linux/x86-64 - Read /etc/passwd - 82 bytes](http://shell-storm.org/shellcode/files/shellcode-878.php) *by Mr.Un1k0d3r*
* [Linux/x86-64 - shutdown -h now - 65 bytes](http://shell-storm.org/shellcode/files/shellcode-877.php) *by Osanda Malith Jayathissa*
* [Linux/x86-64 - TCP Bind 4444 with password - 173 bytes](http://shell-storm.org/shellcode/files/shellcode-873.php) *by Christophe G*
* [Linux/x86-64 - TCP reverse shell with password - 138 bytes](http://shell-storm.org/shellcode/files/shellcode-871.php) *by Andriy Brukhovetskyy*
* [Linux/x86-64 - TCP bind shell with password - 175 bytes](http://shell-storm.org/shellcode/files/shellcode-870.php) *by Andriy Brukhovetskyy*
* [Linux/x86-64 - Reads data from /etc/passwd to /tmp/outfile - 118 bytes](http://shell-storm.org/shellcode/files/shellcode-867.php) *by Chris Higgins*
* [Linux/x86-64 - shell bind TCP random port - 57 bytes](http://shell-storm.org/shellcode/files/shellcode-859.php) *by Geyslan G. Bem*
* [Linux/x86-64 - TCP bind shell - 150 bytes](http://shell-storm.org/shellcode/files/shellcode-858.php) *by Russell Willis*
* [Linux/x86-64 - Reverse TCP shell - 118 bytes](http://shell-storm.org/shellcode/files/shellcode-857.php) *by Russell Willis*
* [Linux/x86-64 - add user with passwd - 189 bytes](http://shell-storm.org/shellcode/files/shellcode-801.php) *by 0\_o*
* [Linux/x86-64 - execve(/sbin/iptables, [/sbin/iptables, -F], NULL) - 49 bytes](http://shell-storm.org/shellcode/files/shellcode-683.php) *by 10n1z3d*
* [Linux/x86-64 - Execute /bin/sh - 27 bytes](http://shell-storm.org/shellcode/files/shellcode-806.php) *by Dad`*
* [Linux/x86-64 - bind-shell with netcat - 131 bytes](http://shell-storm.org/shellcode/files/shellcode-822.php) *by Gaussillusion*
* [Linux/x86-64 - connect back shell with netcat - 109 bytes](http://shell-storm.org/shellcode/files/shellcode-823.php) *by Gaussillusion*
* [Linux/x86-64 - setreuid(0,0) execve(/bin/ash,NULL,NULL) + XOR - 85 bytes](http://shell-storm.org/shellcode/files/shellcode-815.php) *by egeektronic*
* [Linux/x86-64 - setreuid(0,0) execve(/bin/csh, [/bin/csh, NULL]) + XOR - 87 bytes](http://shell-storm.org/shellcode/files/shellcode-816.php) *by egeektronic*
* [Linux/x86-64 - setreuid(0,0) execve(/bin/ksh, [/bin/ksh, NULL]) + XOR - 87 bytes](http://shell-storm.org/shellcode/files/shellcode-817.php) *by egeektronic*
* [Linux/x86-64 - setreuid(0,0) execve(/bin/zsh, [/bin/zsh, NULL]) + XOR - 87 bytes](http://shell-storm.org/shellcode/files/shellcode-818.php) *by egeektronic*
* [Linux/x86-64 - bindshell port:4444 shellcode - 132 bytes](http://shell-storm.org/shellcode/files/shellcode-78.php) *by evil.xi4oyu*
* [Linux/x86-64 - setuid(0) + execve(/bin/sh) 49 bytes](http://shell-storm.org/shellcode/files/shellcode-77.php) *by evil.xi4oyu*
* [Linux/x86-64 - execve(/bin/sh, [/bin/sh], NULL) - 33 bytes](http://shell-storm.org/shellcode/files/shellcode-76.php) *by hophet*
* [Linux/x86-64 - execve(/bin/sh); - 30 bytes](http://shell-storm.org/shellcode/files/shellcode-603.php) *by zbt*
* [Linux/x86-64 - reboot(POWER\_OFF) - 19 bytes](http://shell-storm.org/shellcode/files/shellcode-602.php) *by zbt*
* [Linux/x86-64 - sethostname() & killall - 33 bytes](http://shell-storm.org/shellcode/files/shellcode-605.php) *by zbt*

**Intel x86**

* [Linux/x86 - setuid + setgid + stdin re-open + exceve - 71 bytes](http://shell-storm.org/shellcode/files/shellcode-906.php) *by Andres C. Rodriguez (acamro)*
* [Linux/x86 - Followtheleader custom execve-shellcode Encoder/Decoder - 136 bytes](http://shell-storm.org/shellcode/files/shellcode-902.php) *by Konstantinos Alexiou*
* [Linux/x86 - ROT-7 Decoder execve - 74 bytes](http://shell-storm.org/shellcode/files/shellcode-900.php) *by Stavros Metzidakis*
* [Linux/x86 - Add map in /etc/hosts file - 77 bytes](http://shell-storm.org/shellcode/files/shellcode-893.php) *by Javier Tejedor*
* [Linux/x86 - Obfuscated - chmod({passwd,shadow}) - add new root user - exec /bin/sh - 512 bytes](http://shell-storm.org/shellcode/files/shellcode-887.php) *by Ali Razmjoo*
* [Linux/x86 - setreuid() + exec /usr/bin/python - 54 bytes](http://shell-storm.org/shellcode/files/shellcode-886.php) *by Ali Razmjoo*
* [Linux/x86 - chmod + Add new root user with password + exec sh - 378 bytes](http://shell-storm.org/shellcode/files/shellcode-885.php) *by Ali Razmjoo*
* [Linux/x86 - Shell Reverse TCP Shellcode - 74 bytes](http://shell-storm.org/shellcode/files/shellcode-883.php) *by Julien Ahrens*
* [Linux/x86 - Shell Bind TCP Shellcode Port 1337 - 89 bytes](http://shell-storm.org/shellcode/files/shellcode-882.php) *by Julien Ahrens*
* [Linux/x86 - sockfd trick + dup2(0,0),dup2(0,1),dup2(0,2) + execve /bin/sh - 50 bytes](http://shell-storm.org/shellcode/files/shellcode-881.php) *by ZadYree*
* [Linux/x86 - shutdown -h now - 56 bytes](http://shell-storm.org/shellcode/files/shellcode-876.php) *by Osanda Malith Jayathissa*
* [Linux/x86 - chmod 0777 /etc/shadow (a bit obfuscated) Shellcode - 51 bytes](http://shell-storm.org/shellcode/files/shellcode-875.php) *by Osanda Malith Jayathissa*
* [Linux/x86 - /bin/nc -le /bin/sh -vp 17771 - 58 bytes](http://shell-storm.org/shellcode/files/shellcode-872.php) *by Oleg Boytsev*
* [Linux/x86 - JMP-FSTENV execve shell - 67 bytes](http://shell-storm.org/shellcode/files/shellcode-869.php) *by Paolo Stivanin*
* [Linux/x86 - shift-bit-encoder execve - 114 bytes](http://shell-storm.org/shellcode/files/shellcode-868.php) *by Shihao Song*
* [Linux/x86 - Copy /etc/passwd to /tmp/outfile - 97 bytes](http://shell-storm.org/shellcode/files/shellcode-864.php) *by Paolo Stivanin*
* [Linux/x86 - jump-call-pop execve shell - 52 bytes](http://shell-storm.org/shellcode/files/shellcode-863.php) *by Paolo Stivanin*
* [Linux/x86 - Download + chmod + exec - 108 bytes](http://shell-storm.org/shellcode/files/shellcode-862.php) *by Daniel Sauder*
* [Linux/x86 - reads /etc/passwd and sends the content to 127.1.1.1 port 12345 - 111 bytes](http://shell-storm.org/shellcode/files/shellcode-861.php) *by Daniel Sauder*
* [Linux/x86 - Multi-Egghunter](http://shell-storm.org/shellcode/files/shellcode-856.php) *by Ryan Fenno*
* [Linux/x86 - Obfuscated tcp bind shell - 112 bytes](http://shell-storm.org/shellcode/files/shellcode-852.php) *by Russell Willis*
* [Linux/x86 - Obfuscated execve /bin/sh - 30 bytes](http://shell-storm.org/shellcode/files/shellcode-851.php) *by Russell Willis*
* [Linux/x86 - egghunter shellcode](http://shell-storm.org/shellcode/files/shellcode-850.php) *by Russell Willis*
* [Linux/x86 - Reverse TCP bind shell - 92 bytes](http://shell-storm.org/shellcode/files/shellcode-849.php) *by Russell Willis*
* [Linux/x86 - Set /proc/sys/net/ipv4/ip\_forward to 0 & exit() - 83 bytes](http://shell-storm.org/shellcode/files/shellcode-848.php) *by Hamid Zamani*
* [Linux/x86 - TCP bind shell - 108 bytes](http://shell-storm.org/shellcode/files/shellcode-847.php) *by Russell Willis*
* [Linux/x86 - Encrypted execve /bin/sh with uzumaki algorithm - 50 bytes](http://shell-storm.org/shellcode/files/shellcode-846.php) *by Geyslan G. Bem*
* [Linux/x86 - Mutated Execve Wget - 96 bytes](http://shell-storm.org/shellcode/files/shellcode-845.php) *by Geyslan G. Bem*
* [Linux/x86 - Mutated Fork Bomb - 15 bytes](http://shell-storm.org/shellcode/files/shellcode-844.php) *by Geyslan G. Bem*
* [Linux/x86 - Mutated Reboot - 55 bytes](http://shell-storm.org/shellcode/files/shellcode-843.php) *by Geyslan G. Bem*
* [Linux/x86 - Tiny read /etc/passwd file - 51 bytes](http://shell-storm.org/shellcode/files/shellcode-842.php) *by Geyslan G. Bem*
* [Linux/x86 - Tiny Execve sh Shellcode - 21 bytes](http://shell-storm.org/shellcode/files/shellcode-841.php) *by Geyslan G. Bem*
* [Linux/x86 - Insertion Decoder Shellcode - 33+ bytes](http://shell-storm.org/shellcode/files/shellcode-840.php) *by Geyslan G. Bem*
* [Linux/x86 - Egg Hunter Shellcode - 38 bytes](http://shell-storm.org/shellcode/files/shellcode-839.php) *by Geyslan G. Bem*
* [Linux/x86 - Tiny Shell Reverse TCP - 67 bytes](http://shell-storm.org/shellcode/files/shellcode-838.php) *by Geyslan G. Bem*
* [Linux/x86 - Tiny Shell Bind TCP Random Port - 57 bytes](http://shell-storm.org/shellcode/files/shellcode-837.php) *by Geyslan G. Bem*
* [Linux/x86 - Tiny Shell Bind TCP - 73 bytes](http://shell-storm.org/shellcode/files/shellcode-836.php) *by Geyslan G. Bem*
* [Linux/x86 - Shell Bind TCP (GetPC/Call/Ret Method) - 89 bytes](http://shell-storm.org/shellcode/files/shellcode-835.php) *by Geyslan G. Bem*
* [Linux/x86 - append /etc/passwd & exit() - 107 bytes](http://shell-storm.org/shellcode/files/shellcode-561.php) *by $andman*
* [Linux/x86 - unlink(/etc/passwd) & exit() - 35 bytes](http://shell-storm.org/shellcode/files/shellcode-560.php) *by $andman*
* [Linux/x86 - connect back&send&exit /etc/shadow - 155 bytes](http://shell-storm.org/shellcode/files/shellcode-207.php) *by 0in*
* [Linux/x86 - execve read shellcode - 92 bytes](http://shell-storm.org/shellcode/files/shellcode-204.php) *by 0ut0fbound*
* [Linux/x86 - egghunt shellcode - 29 bytes](http://shell-storm.org/shellcode/files/shellcode-784.php) *by Ali Raheem*
* [Linux/x86 - nc -lvve/bin/sh -p13377 - 62 bytes](http://shell-storm.org/shellcode/files/shellcode-804.php) *by Anonymous*
* [Linux/x86 - /bin/sh Null-Free Polymorphic - 46 bytes](http://shell-storm.org/shellcode/files/shellcode-690.php) *by Aodrulez*
* [Linux/x86 - execve() Diassembly Obfuscation Shellcode - 32 bytes](http://shell-storm.org/shellcode/files/shellcode-237.php) *by BaCkSpAcE*
* [Linux/x86 - SET\_IP() Connectback Shellcode - 82 bytes](http://shell-storm.org/shellcode/files/shellcode-239.php) *by Benjamin Orozco*
* [Linux/x86 - SET\_PORT() portbind - 100 bytes](http://shell-storm.org/shellcode/files/shellcode-238.php) *by Benjamin Orozco*
* [Linux/x86 - netcat bindshell port 8080 - 75 bytes](http://shell-storm.org/shellcode/files/shellcode-684.php) *by Blake*
* [Linux/x86 - netcat connect back port 8080 - 76 bytes](http://shell-storm.org/shellcode/files/shellcode-685.php) *by Blake*
* [Linux/x86 - adds a root user no-passwd to /etc/passwd - 83 bytes](http://shell-storm.org/shellcode/files/shellcode-548.php) *by Bob [Dtors.net]*
* [Linux/x86 - chmod(//bin/sh ,04775); set sh +s - 31 bytes](http://shell-storm.org/shellcode/files/shellcode-550.php) *by Bob [Dtors.net]*
* [Linux/x86 - execve()/bin/ash; exit; - 34 bytes](http://shell-storm.org/shellcode/files/shellcode-547.php) *by Bob [Dtors.net]*
* [Linux/x86 - setuid(); execve(); exit(); - 44 bytes](http://shell-storm.org/shellcode/files/shellcode-549.php) *by Bob [Dtors.net]*
* [Linux/x86 - setreuid(0, 0) + execve(/bin//sh, [/bin//sh, -c, cmd], NULL);](http://shell-storm.org/shellcode/files/shellcode-216.php) *by Bunker*
* [Linux/x86 - dup2(0,0); dup2(0,1); dup2(0,2); 15 bytes](http://shell-storm.org/shellcode/files/shellcode-259.php) *by Charles Stevenson*
* [Linux/x86 - exit(1) - 7 bytes](http://shell-storm.org/shellcode/files/shellcode-55.php) *by Charles Stevenson*
* [Linux/x86 - if(read(fd,buf,512)<=2) \_exit(1) else buf(); - 29 bytes](http://shell-storm.org/shellcode/files/shellcode-258.php) *by Charles Stevenson*
* [Linux/x86 - read(0,buf,2541); chmod(buf,4755); - 23 bytes](http://shell-storm.org/shellcode/files/shellcode-257.php) *by Charles Stevenson*
* [Linux/x86 - execve(/bin/dash) - 49 bytes](http://shell-storm.org/shellcode/files/shellcode-756.php) *by Chroniccommand*
* [Linux/x86 - Audio (knock knock knock) via /dev/dsp+setreuid(0,0)+execve() - 566 bytes](http://shell-storm.org/shellcode/files/shellcode-541.php) *by Cody Tubbs*
* [Linux/x86 - Surprise ! ! ! - 361 bytes](http://shell-storm.org/shellcode/files/shellcode-775.php) *by Florian Gaultier*
* [Linux/x86 - Write FS PHP Connect Back Utility Shellcode - 508 bytes](http://shell-storm.org/shellcode/files/shellcode-208.php) *by GS2008*
* [Linux/x86 - Bind TCP Port - with SO\_REUSEADDR set (Avoiding SIGSEGV) - 103 bytes](http://shell-storm.org/shellcode/files/shellcode-832.php) *by Geyslan G. Bem*
* [Linux/x86 - Shell Bind TCP Random Port - 65 bytes](http://shell-storm.org/shellcode/files/shellcode-834.php) *by Geyslan G. Bem*
* [Linux/x86 - Shell Reverse TCP Shellcode - 72 bytes](http://shell-storm.org/shellcode/files/shellcode-833.php) *by Geyslan G. Bem*
* [Linux/x86 - Password Authentication portbind port 64713/tcp - 166 bytes](http://shell-storm.org/shellcode/files/shellcode-253.php) *by Gotfault Security*
* [Linux/x86 - portbind port 64713 - 86 bytes](http://shell-storm.org/shellcode/files/shellcode-252.php) *by Gotfault Security*
* [Linux/x86 - setreuid(0,0) + execve(/bin/sh, [/bin/sh, NULL]) - 33 bytes](http://shell-storm.org/shellcode/files/shellcode-250.php) *by Gotfault Security*
* [Linux/x86 - setuid(0) setgid(0) execve("/bin/sh", ["/bin/sh", NULL]) - 37 bytes](http://shell-storm.org/shellcode/files/shellcode-251.php) *by Gotfault Security*
* [Linux/x86 - Force Reboot shellcode 36 bytes](http://shell-storm.org/shellcode/files/shellcode-831.php) *by Hamza Megahed*
* [Linux/x86 - Remote Port forwarding - 87 bytes](http://shell-storm.org/shellcode/files/shellcode-829.php) *by Hamza Megahed*
* [Linux/x86 - execve /bin/sh shellcode - 23 bytes](http://shell-storm.org/shellcode/files/shellcode-827.php) *by Hamza Megahed*
* [Linux/x86 - execve-chmod 0777 /etc/shadow - 57 bytes](http://shell-storm.org/shellcode/files/shellcode-828.php) *by Hamza Megahed*
* [Linux/x86 - iptables --flush - 43 bytes](http://shell-storm.org/shellcode/files/shellcode-825.php) *by Hamza Megahed*
* [Linux/x86 - ASLR deactivation - 83 bytes](http://shell-storm.org/shellcode/files/shellcode-813.php) *by Jean Pascal Pereira*
* [Linux/x86 - chmod 666 /etc/passwd & /etc/shadow - 57 bytes](http://shell-storm.org/shellcode/files/shellcode-812.php) *by Jean Pascal Pereira*
* [Linux/x86 - execve(/bin/sh) - 28 bytes](http://shell-storm.org/shellcode/files/shellcode-811.php) *by Jean Pascal Pereira*
* [Linux/x86 - ///sbin/iptables -POUTPUT DROP - 60 bytes](http://shell-storm.org/shellcode/files/shellcode-740.php) *by John Babio*
* [Linux/x86 - /etc/init.d/apparmor teardown - 53 bytes](http://shell-storm.org/shellcode/files/shellcode-765.php) *by John Babio*
* [Linux/x86 - /usr/bin/killall snort - 46 bytes](http://shell-storm.org/shellcode/files/shellcode-741.php) *by John Babio*
* [Linux/x86 - /bin/sh polymorphic shellcode - 48 bytes](http://shell-storm.org/shellcode/files/shellcode-491.php) *by Jonathan Salwan*
* [Linux/x86 - ConnectBack with SSL connection - 422 bytes](http://shell-storm.org/shellcode/files/shellcode-770.php) *by Jonathan Salwan*
* [Linux/x86 - Remote file Download - 42 bytes](http://shell-storm.org/shellcode/files/shellcode-611.php) *by Jonathan Salwan*
* [Linux/x86 - execve(/bin/bash, [/bin/sh, -p], NULL) - 33 bytes](http://shell-storm.org/shellcode/files/shellcode-606.php) *by Jonathan Salwan*
* [Linux/x86 - polymorphic execve(/bin/bash, [/bin/sh, -p], NULL) - 57 bytes](http://shell-storm.org/shellcode/files/shellcode-607.php) *by Jonathan Salwan*
* [Linux/x86 - /bin/sh - 8 bytes](http://shell-storm.org/shellcode/files/shellcode-574.php) *by JungHoon Shin*
* [Linux/x86 - add root user (r00t) with no password to /etc/passwd](http://shell-storm.org/shellcode/files/shellcode-211.php) *by Kris Katterjohn*
* [Linux/x86 - chmod(/etc/shadow, 0666) & exit()](http://shell-storm.org/shellcode/files/shellcode-210.php) *by Kris Katterjohn*
* [Linux/x86 - execve(rm -rf /) - 45 bytes](http://shell-storm.org/shellcode/files/shellcode-57.php) *by Kris Katterjohn*
* [Linux/x86 - forkbomb - 7 bytes](http://shell-storm.org/shellcode/files/shellcode-214.php) *by Kris Katterjohn*
* [Linux/x86 - ipchains -F - 40 bytes](http://shell-storm.org/shellcode/files/shellcode-65.php) *by Kris Katterjohn*
* [Linux/x86 - kill all processes - 11 bytes](http://shell-storm.org/shellcode/files/shellcode-212.php) *by Kris Katterjohn*
* [Linux/x86 - set system time to 0 & exit](http://shell-storm.org/shellcode/files/shellcode-213.php) *by Kris Katterjohn*
* [Linux/x86 - setuid(0) setgid(0) execve(echo 0 > /proc/sys/kernel/randomize\_va\_space) - 79 bytes](http://shell-storm.org/shellcode/files/shellcode-222.php) *by LiquidWorm*
* [Linux/x86 - DoS-Badger-Game - 6 bytes](http://shell-storm.org/shellcode/files/shellcode-586.php) *by Magnefikko*
* [Linux/x86 - SLoc-DoS shellcode - 55 bytes](http://shell-storm.org/shellcode/files/shellcode-587.php)*by Magnefikko*
* [Linux/x86 - bind sh@64533 - 97 bytes](http://shell-storm.org/shellcode/files/shellcode-672.php) *by Magnefikko*
* [Linux/x86 - chmod(/etc/shadow, 0666) - 36 bytes](http://shell-storm.org/shellcode/files/shellcode-584.php) *by Magnefikko*
* [Linux/x86 - chmod(/etc/shadow, 0777) - 29 bytes](http://shell-storm.org/shellcode/files/shellcode-593.php) *by Magnefikko*
* [Linux/x86 - execve(/bin/sh) - 25 bytes](http://shell-storm.org/shellcode/files/shellcode-585.php) *by Magnefikko*
* [Linux/x86 - execve(a->/bin/sh) - 14 bytes](http://shell-storm.org/shellcode/files/shellcode-589.php) *by Magnefikko*
* [Linux/x86 - setreud(getuid(), getuid()) & execve(/bin/sh) - 34 bytes](http://shell-storm.org/shellcode/files/shellcode-597.php) *by Magnefikko*
* [Linux/x86 - setuid(0) ^ execve(/bin/sh, 0, 0) - 27 bytes](http://shell-storm.org/shellcode/files/shellcode-599.php) *by Magnefikko*
* [Linux/x86 - setuid(0) + execve(/bin/sh,...) - 29 bytes](http://shell-storm.org/shellcode/files/shellcode-598.php) *by Marcin Ulikowski*
* [Linux/x86 - re-use of (/bin/sh) string in .rodata - 16 bytes](http://shell-storm.org/shellcode/files/shellcode-218.php) *by Marco Ivaldi*
* [Linux/x86 - setuid/portbind port 31337 TCP - 96 bytes](http://shell-storm.org/shellcode/files/shellcode-217.php) *by Marco Ivaldi*
* [Linux/x86 - stdin re-open and /bin/sh execute](http://shell-storm.org/shellcode/files/shellcode-219.php) *by Marco Ivaldi*
* [Linux/x86 - add user t00r ENCRYPT - 116 bytes](http://shell-storm.org/shellcode/files/shellcode-353.php) *by Matias Sedalo*
* [Linux/x86 - chmod 666 /etc/shadow - 41 bytes](http://shell-storm.org/shellcode/files/shellcode-355.php) *by Matias Sedalo*
* [Linux/x86 - chmod 666 shadow ENCRYPT - 75 bytes](http://shell-storm.org/shellcode/files/shellcode-354.php) *by Matias Sedalo*
* [Linux/x86 - execve /bin/sh encrypted - 58 bytes](http://shell-storm.org/shellcode/files/shellcode-358.php) *by Matias Sedalo*
* [Linux/x86 - portbind a shell in port 5074 - 92 bytes](http://shell-storm.org/shellcode/files/shellcode-357.php) *by Matias Sedalo*
* [Linux/x86 - execve /bin/sh anti-ids 40 bytes](http://shell-storm.org/shellcode/files/shellcode-256.php) *by NicatiN*
* [Linux/x86 - /bin/cp /bin/sh /tmp/katy & chmod 4555 - 126 bytes](http://shell-storm.org/shellcode/files/shellcode-51.php) *by RaiSe*
* [Linux/x86 - execve(/bin//sh/,[/bin//sh],NULL) - 22 bytes](http://shell-storm.org/shellcode/files/shellcode-58.php) *by Revenge*
* [Linux/x86 - setuid(0) + execve(/bin//sh, [/bin//sh], NULL) - 28 bytes](http://shell-storm.org/shellcode/files/shellcode-215.php) *by Revenge*
* [Linux/x86 - Port Bind 4444 ( xor-encoded ) - 152 bytes](http://shell-storm.org/shellcode/files/shellcode-481.php) *by Rick*
* [Linux/x86 - edit /etc/sudoers for full access - 86 bytes](http://shell-storm.org/shellcode/files/shellcode-62.php) *by Rick*
* [Linux/x86 - Connect Back shellcode - 90 bytes](http://shell-storm.org/shellcode/files/shellcode-242.php) *by Russell Sanford*
* [Linux/x86 - socket-proxy - 372 bytes](http://shell-storm.org/shellcode/files/shellcode-56.php) *by Russell Sanford*
* [Linux/x86 - [setreuid()] -> [/sbin/iptables -F] -> [exit(0)] - 76 bytes](http://shell-storm.org/shellcode/files/shellcode-805.php) *by Sh3llc0d3*
* [Linux/x86 - Add root user /etc/passwd - 104 bytes](http://shell-storm.org/shellcode/files/shellcode-407.php) *by Shok*
* [Linux/x86 - iptables -F - 49 bytes](http://shell-storm.org/shellcode/files/shellcode-365.php) *by Sp4rK*
* [Linux/x86 - execve(/sbin/halt,/sbin/halt) - 27 bytes](http://shell-storm.org/shellcode/files/shellcode-483.php) *by TheWorm*
* [Linux/x86 - execve(/sbin/reboot,/sbin/reboot) - 28 bytes](http://shell-storm.org/shellcode/files/shellcode-477.php) *by TheWorm*
* [Linux/x86 - execve(/sbin/shutdown,/sbin/shutdown 0) - 36 bytes](http://shell-storm.org/shellcode/files/shellcode-476.php) *by TheWorm*
* [Linux/x86 - exit(0) 3 bytes or exit(1) 4 bytes](http://shell-storm.org/shellcode/files/shellcode-470.php) *by TheWorm*
* [Linux/x86 - setuid(0) & execve(/bin/sh,0) - 25 bytes](http://shell-storm.org/shellcode/files/shellcode-472.php) *by TheWorm*
* [Linux/x86 - setuid(0), setgid(0) & execve(/bin/sh,[/bin/sh,NULL]) - 33 bytes](http://shell-storm.org/shellcode/files/shellcode-473.php) *by TheWorm*
* [Linux/x86 - System Beep - 45 bytes](http://shell-storm.org/shellcode/files/shellcode-60.php) *by Thomas Rinsma*
* [Linux/x86 - Bindshell TCP/5074 - 226 bytes](http://shell-storm.org/shellcode/files/shellcode-370.php) *by Tora*
* [Linux/x86 - iptables -F - 45 bytes](http://shell-storm.org/shellcode/files/shellcode-368.php) *by UnboundeD*
* [Linux/x86 - Connect-Back port UDP/54321 - 151 bytes](http://shell-storm.org/shellcode/files/shellcode-64.php) *by XenoMuta*
* [Linux/x86 - append rsa key to /root/.ssh/authorized\_keys2 - 295 bytes](http://shell-storm.org/shellcode/files/shellcode-63.php) *by XenoMuta*
* [Linux/x86 - listens for shellcode on tcp/5555 and jumps to it - 83 bytes](http://shell-storm.org/shellcode/files/shellcode-501.php) *by XenoMuta*
* [Linux/x86 - Self-modifying ShellCode for IDS evasion - 64 bytes](http://shell-storm.org/shellcode/files/shellcode-505.php) *by Xenomuta*
* [Linux/x86 - shellcode that forks a HTTP Server on port tcp/8800 - 166 bytes](http://shell-storm.org/shellcode/files/shellcode-506.php) *by Xenomuta*
* [Linux/x86 - stagger that reads second stage shellcode (127 bytes maximum) from stdin - 14 bytes](http://shell-storm.org/shellcode/files/shellcode-824.php) *by \_fkz*
* [Linux/x86 - alphanumeric Bomb FORK Shellcode - 117 Bytes](http://shell-storm.org/shellcode/files/shellcode-619.php) *by agix*
* [Linux/x86 - chmod(/etc/shadow, 0666) ASCII - 443 bytes](http://shell-storm.org/shellcode/files/shellcode-650.php) *by agix*
* [Linux/x86 - pwrite(/etc/shadow, hash, 32, 8) - 89 Bytes](http://shell-storm.org/shellcode/files/shellcode-610.php) *by agix*
* [Linux/x86 - Polymorphic - setuid(0) + chmod(/etc/shadow, 0666) - 61 Bytes](http://shell-storm.org/shellcode/files/shellcode-624.php) *by antrhacks*
* [Linux/x86 - execve(/bin/cat, /etc/shadow, NULL) - 42 bytes](http://shell-storm.org/shellcode/files/shellcode-758.php) *by antrhacks*
* [Linux/x86 - setuid(0) + chmod(/etc/shadow, 0666) - 37 Bytes](http://shell-storm.org/shellcode/files/shellcode-608.php) *by antrhacks*
* [Linux/x86 - setreuid(geteuid(),geteuid()),execve(/bin/sh,0,0) - 34bytes](http://shell-storm.org/shellcode/files/shellcode-399.php) *by blue9057*
* [Linux/x86 - /bin/sh sysenter Opcode Array Payload - 23 Bytes](http://shell-storm.org/shellcode/files/shellcode-236.php) *by c0ntex & BaCkSpAcE*
* [Linux/x86 - File Reader /etc/passwd - 65 bytes](http://shell-storm.org/shellcode/files/shellcode-73.php) *by certaindeath*
* [Linux/x86 - sends Phuck3d! to all terminals - 60 bytes](http://shell-storm.org/shellcode/files/shellcode-604.php) *by condis*
* [Linux/x86 - upload & exec - 189 bytes](http://shell-storm.org/shellcode/files/shellcode-54.php) *by cybertronic*
* [Linux/x86 - File unlinker 18 bytes + file path length](http://shell-storm.org/shellcode/files/shellcode-75.php) *by darkjoker*
* [Linux/x86 - Perl script execution 99 bytes + script length](http://shell-storm.org/shellcode/files/shellcode-74.php) *by darkjoker*
* [Linux/x86 - iptables -F - 58 bytes](http://shell-storm.org/shellcode/files/shellcode-361.php) *by dev0id*
* [Linux/x86 - symlink /bin/sh xoring - 56 bytes](http://shell-storm.org/shellcode/files/shellcode-53.php) *by dev0id*
* [Linux/x86 - iopl(3); asm(cli); while(1){} - 12 bytes](http://shell-storm.org/shellcode/files/shellcode-205.php) *by dun*
* [Linux/x86 - SWAP restore - 109 bytes](http://shell-storm.org/shellcode/files/shellcode-255.php) *by dx & spud*
* [Linux/x86 - SWAP store - 99 bytes](http://shell-storm.org/shellcode/files/shellcode-254.php) *by dx & spud*
* [Linux/x86 - /sbin/iptables --flush - 69 bytes](http://shell-storm.org/shellcode/files/shellcode-554.php) *by eSDee [Netric .org]*
* [Linux/x86 - connect back shellcode (port=0xb0ef) - 131 bytes](http://shell-storm.org/shellcode/files/shellcode-552.php) *by eSDee [Netric .org]*
* [Linux/x86 - forking portbind shellcode - port=0xb0ef(45295) - 200 bytes](http://shell-storm.org/shellcode/files/shellcode-553.php) *by eSDee [Netric .org]*
* [Linux/x86 - setreuid(0,0) execve("/bin/zsh", [/bin/zsh, NULL]) + XOR - 53 bytes](http://shell-storm.org/shellcode/files/shellcode-810.php) *by egeektronic*
* [Linux/x86 - setreuid(0,0) execve("/bin/csh", [/bin/csh, NULL]) + XOR - 53 bytes](http://shell-storm.org/shellcode/files/shellcode-808.php) *by egeektronic*
* [Linux/x86 - setreuid(0,0) execve("/bin/ksh", [/bin/ksh, NULL]) + XOR - 53 bytes](http://shell-storm.org/shellcode/files/shellcode-809.php) *by egeektronic*
* [Linux/x86 - setreuid(0,0) execve(/bin/ash,NULL,NULL) + XOR - 58 bytes](http://shell-storm.org/shellcode/files/shellcode-807.php) *by egeektronic*
* [Linux/x86 - bin/cat /etc/passwd - 43 bytes](http://shell-storm.org/shellcode/files/shellcode-571.php) *by fb1h2s*
* [Linux/x86 - execve() - 51bytes](http://shell-storm.org/shellcode/files/shellcode-555.php) *by fl0 fl0w*
* [Linux/x86 - Find all writeable folder in filesystem linux polymorphic shellcode](http://shell-storm.org/shellcode/files/shellcode-682.php) *by gunslinger\_*
* [Linux/x86 - Polymorphic bindport to 13123 - 125 bytes](http://shell-storm.org/shellcode/files/shellcode-656.php) *by gunslinger\_*
* [Linux/x86 - Polymorphic bindport to 31337 with setreuid (0,0) - 131 bytes](http://shell-storm.org/shellcode/files/shellcode-655.php) *by gunslinger\_*
* [Linux/x86 - bind port to 6678 XOR encoded polymorphic - 125 bytes](http://shell-storm.org/shellcode/files/shellcode-678.php) *by gunslinger\_*
* [Linux/x86 - cdrom ejecting shellcode - 46 bytes](http://shell-storm.org/shellcode/files/shellcode-621.php) *by gunslinger\_*
* [Linux/x86 - chown root:root /bin/sh - 48 bytes](http://shell-storm.org/shellcode/files/shellcode-642.php) *by gunslinger\_*
* [Linux/x86 - force unmount /media/disk - 33 bytes](http://shell-storm.org/shellcode/files/shellcode-641.php) *by gunslinger\_*
* [Linux/x86 - give all user root access when execute /bin/sh - 45 bytes](http://shell-storm.org/shellcode/files/shellcode-643.php) *by gunslinger\_*
* [Linux/x86 - hard reboot (without any message) and data not lost - 33 bytes](http://shell-storm.org/shellcode/files/shellcode-639.php) *by gunslinger\_*
* [Linux/x86 - hard reboot (without any message) and data will be lost - 29 bytes](http://shell-storm.org/shellcode/files/shellcode-638.php) *by gunslinger\_*
* [Linux/x86 - nc -lp 31337 -e /bin//sh polymorphic - 91 bytes](http://shell-storm.org/shellcode/files/shellcode-679.php) *by gunslinger\_*
* [Linux/x86 - polymorphic cdrom ejecting - 74 bytes](http://shell-storm.org/shellcode/files/shellcode-653.php) *by gunslinger\_*
* [Linux/x86 - setdomainname to (th1s s3rv3r h4s b33n h1j4ck3d !!)](http://shell-storm.org/shellcode/files/shellcode-636.php) *by gunslinger\_*
* [Linux/x86 - sys\_chmod(/etc/shadow, 599) - 39 bytes](http://shell-storm.org/shellcode/files/shellcode-625.php) *by gunslinger\_*
* [Linux/x86 - sys\_execve(/bin/sh, -c, ping localhost) - 55 bytes](http://shell-storm.org/shellcode/files/shellcode-632.php) *by gunslinger\_*
* [Linux/x86 - sys\_exit(0) - 8 bytes](http://shell-storm.org/shellcode/files/shellcode-623.php) *by gunslinger\_*
* [Linux/x86 - sys\_kill(-1,9) - 11 bytes](http://shell-storm.org/shellcode/files/shellcode-626.php) *by gunslinger\_*
* [Linux/x86 - sys\_rmdir(/tmp/willdeleted) - 41 bytes](http://shell-storm.org/shellcode/files/shellcode-633.php) *by gunslinger\_*
* [Linux/x86 - sys\_sethostname(PwNeD !!, 8) - 32 bytes](http://shell-storm.org/shellcode/files/shellcode-622.php) *by gunslinger\_*
* [Linux/x86 - sys\_setuid(0) & sys\_setgid(0) & execve (/bin/sh) - 39 bytes](http://shell-storm.org/shellcode/files/shellcode-631.php) *by gunslinger\_*
* [Linux/x86 - sys\_sync - 6 bytes](http://shell-storm.org/shellcode/files/shellcode-630.php) *by gunslinger\_*
* [Linux/x86 - unlink /etc/shadow - 33 bytes](http://shell-storm.org/shellcode/files/shellcode-634.php) *by gunslinger\_*
* [Linux/x86 - Reverse Telnet](http://shell-storm.org/shellcode/files/shellcode-367.php) *by hts*
* [Linux/x86 - execve /bin/sh - 21 bytes](http://shell-storm.org/shellcode/files/shellcode-575.php) *by ipv*
* [Linux/x86 - HTTP/1.x GET, Downloads & execve() - 111 bytes+](http://shell-storm.org/shellcode/files/shellcode-59.php) *by izik*
* [Linux/x86 - HTTP/1.x GET, Downloads and JMP - 68 bytes+](http://shell-storm.org/shellcode/files/shellcode-227.php) *by izik*
* [Linux/x86 - anti-debug trick (INT 3h trap) execve(/bin/sh, [/bin/sh, NULL], NULL) - 39 bytes](http://shell-storm.org/shellcode/files/shellcode-230.php) *by izik*
* [Linux/x86 - cat /dev/urandom > /dev/console, no real profit just for kicks - 63 bytes](http://shell-storm.org/shellcode/files/shellcode-234.php) *by izik*
* [Linux/x86 - eject & close cd-rom frenzy loop (follows /dev/cdrom symlink) - 45 bytes](http://shell-storm.org/shellcode/files/shellcode-232.php) *by izik*
* [Linux/x86 - execve /bin/sh xored for Intel x86 CPUID 41 bytes](http://shell-storm.org/shellcode/files/shellcode-228.php) *by izik*
* [Linux/x86 - execve(/bin/sh, [/bin/sh, NULL]) + Bitmap - 27 bytes](http://shell-storm.org/shellcode/files/shellcode-226.php) *by izik*
* [Linux/x86 - execve(/bin/sh, [/bin/sh, NULL]) + RIFF Header - 28 bytes](http://shell-storm.org/shellcode/files/shellcode-225.php) *by izik*
* [Linux/x86 - execve(/bin/sh, [/bin/sh, NULL]) + RTF header - 30 bytes](http://shell-storm.org/shellcode/files/shellcode-224.php) *by izik*
* [Linux/x86 - execve(/bin/sh, [/bin/sh, NULL]) + ZIP Header - 28 bytes](http://shell-storm.org/shellcode/files/shellcode-223.php) *by izik*
* [Linux/x86 - execve(/bin/sh, [/bin/sh], NULL) / encoded by +1 - 39 bytes](http://shell-storm.org/shellcode/files/shellcode-229.php) *by izik*
* [Linux/x86 - open cd-rom loop (follows /dev/cdrom symlink) - 39 bytes](http://shell-storm.org/shellcode/files/shellcode-231.php) *by izik*
* [Linux/x86 - quick (yet conditional, eax != 0 and edx == 0) exit - 4 bytes](http://shell-storm.org/shellcode/files/shellcode-233.php) *by izik*
* [Linux/x86 - chmod(/etc/shadow, 0666) & exit() - 33 bytes](http://shell-storm.org/shellcode/files/shellcode-556.php) *by ka0x*
* [Linux/x86 - setuid(0) & execve(/bin/cat /etc/shadow) - 49 bytes](http://shell-storm.org/shellcode/files/shellcode-557.php) *by ka0x*
* [Linux/x86 - setuid(0) & execve(/sbin/poweroff -f) - 47 bytes](http://shell-storm.org/shellcode/files/shellcode-558.php) *by ka0x*
* [Linux/x86 - execve (/bin/sh) - 21 Bytes](http://shell-storm.org/shellcode/files/shellcode-752.php) *by kernel\_panik*
* [Linux/x86 - Bindport TCP/3879](http://shell-storm.org/shellcode/files/shellcode-366.php) *by lamagra*
* [Linux/x86 - connect back, download a file and execute - 149 bytes](http://shell-storm.org/shellcode/files/shellcode-206.php) *by militan*
* [Linux/x86 - raw-socket ICMP/checksum shell - 235 bytes](http://shell-storm.org/shellcode/files/shellcode-209.php) *by mu-b*
* [Linux/x86 - hence dropping a SUID root shell in /tmp - 126 bytes](http://shell-storm.org/shellcode/files/shellcode-540.php) *by n/a*
* [Linux/x86 - kill snort - 151 bytes](http://shell-storm.org/shellcode/files/shellcode-52.php) *by nob0dy*
* [Linux/x86 - setreuid & execve - 31 bytes](http://shell-storm.org/shellcode/files/shellcode-261.php) *by oc192*
* [Linux/x86 - rm -rf / which attempts to block the process from being stopped - 132 bytes](http://shell-storm.org/shellcode/files/shellcode-221.php) *by onionring*
* [Linux/x86 - portbind (define your own port) - 84 bytes](http://shell-storm.org/shellcode/files/shellcode-235.php) *by oveRet*
* [Linux/x86 - setuid(0)+setgid(0)+add user iph without password - 124 bytes](http://shell-storm.org/shellcode/files/shellcode-798.php) *by pentesters.ir*
* [Linux/x86 - break chroot execve /bin/sh - 80 bytes](http://shell-storm.org/shellcode/files/shellcode-363.php) *by preedator*
* [Linux/x86 - Search php,html writable files and add your code - 380+ bytes](http://shell-storm.org/shellcode/files/shellcode-799.php) *by rigan*
* [Linux/x86 - chmod 666 /etc/shadow - 27 bytes](http://shell-storm.org/shellcode/files/shellcode-566.php) *by root@thegibson*
* [Linux/x86 - eject /dev/cdrom - 42 bytes](http://shell-storm.org/shellcode/files/shellcode-563.php) *by root@thegibson*
* [Linux/x86 - kill all processes - 9 bytes](http://shell-storm.org/shellcode/files/shellcode-564.php) *by root@thegibson*
* [Linux/x86 - overwrite MBR on /dev/sda with LOL! - 43 bytes](http://shell-storm.org/shellcode/files/shellcode-565.php) *by root@thegibson*
* [Linux/x86 - execve(/bin/sh,0,0) - 21 bytes](http://shell-storm.org/shellcode/files/shellcode-517.php) *by sToRm*
* [Linux/x86 - portbind /bin/sh (port 64713) - 83 bytes](http://shell-storm.org/shellcode/files/shellcode-515.php) *by sToRm*
* [Linux/x86 - setuid(0) & execve(/bin/sh,0,0) - 28 bytes](http://shell-storm.org/shellcode/files/shellcode-516.php) *by sToRm*
* [Linux/x86 - setresuid(0,0,0); execve /bin/sh; exit; - 41 bytes](http://shell-storm.org/shellcode/files/shellcode-551.php) *by sacrine*
* [Linux/x86 - setuid(0) & execve(/bin/sh,0,0) - 28 bytes](http://shell-storm.org/shellcode/files/shellcode-61.php) *by sch3m4*
* [Linux/x86 - disabled modsecurity - 64 bytes](http://shell-storm.org/shellcode/files/shellcode-578.php) *by sekfault*
* [Linux/x86 - shared memory exec - 50 bytes](http://shell-storm.org/shellcode/files/shellcode-369.php) *by sloth*
* [Linux/x86 - chmod(/etc/shadow, 0777) - 33 bytes](http://shell-storm.org/shellcode/files/shellcode-590.php) *by sm0k*
* [Linux/x86 - setresuid(0,0,0)-/bin/sh - 35 bytes](http://shell-storm.org/shellcode/files/shellcode-220.php) *by sorrow*
* [Linux/x86 - Add User USER=t00r PASS=t00r - Encoder PexFnstenvSub - 116 bytes](http://shell-storm.org/shellcode/files/shellcode-311.php) *by vlad902*
* [Linux/x86 - disables shadowing - 42 bytes](http://shell-storm.org/shellcode/files/shellcode-465.php) *by vlan7*
* [Linux/x86 - setuid() & execve() - 27 bytes](http://shell-storm.org/shellcode/files/shellcode-466.php) *by vlan7*
* [Linux/x86 - examples of long-term payloads hide-wait-change - 187 bytes+](http://shell-storm.org/shellcode/files/shellcode-244.php) *by xort & izik*
* [Linux/x86 - Alpha-Numeric using IMUL Method - 88 bytes](http://shell-storm.org/shellcode/files/shellcode-247.php) *by xort*
* [Linux/x86 - Magic Byte Self Modifying Code for surviving - execve() \_exit() - 76 bytes](http://shell-storm.org/shellcode/files/shellcode-249.php) *by xort*
* [Linux/x86 - Radically Self Modifying Code - execve & \_exit() - 70 bytes](http://shell-storm.org/shellcode/files/shellcode-248.php) *by xort*
* [Linux/x86 - alpha-numeric - 64 bytes](http://shell-storm.org/shellcode/files/shellcode-246.php) *by xort*
* [Linux/x86 - examples of long-term payloads hide-wait-change (.s)](http://shell-storm.org/shellcode/files/shellcode-245.php) *by xort*
* [Linux/x86 - add a passwordless local root account w000t - 177 bytes](http://shell-storm.org/shellcode/files/shellcode-543.php) *by zillion*
* [Linux/x86 - execve of /bin/sh /tmp/p00p - 70 bytes](http://shell-storm.org/shellcode/files/shellcode-546.php) *by zillion*
* [Linux/x86 - execve of /sbin/ipchains -F - 70 bytes](http://shell-storm.org/shellcode/files/shellcode-544.php) *by zillion*
* [Linux/x86 - execve() of /sbin/iptables -F - 70 bytes](http://shell-storm.org/shellcode/files/shellcode-545.php) *by zillion*
* [Linux/x86 - mkdir() & exit() - 36 bytes](http://shell-storm.org/shellcode/files/shellcode-542.php) *by zillion*

**NetBSD**

* [NetBSD/x86 - kill all processes shellcode - 23 bytes](http://shell-storm.org/shellcode/files/shellcode-803.php) *by Anonymous*
* [NetBSD/x86 - execve(/bin/sh) - 68 bytes](http://shell-storm.org/shellcode/files/shellcode-108.php) *by humble*
* [NetBSD/x86 - callback (port 6666) - 83 bytes](http://shell-storm.org/shellcode/files/shellcode-110.php) *by minervini*
* [NetBSD/x86 - setreuid(0, 0); execve(/bin//sh, ..., NULL); - 29 bytes](http://shell-storm.org/shellcode/files/shellcode-109.php) *by minervini*

**OpenBSD**

* [OpenBSD/x86 - reboot() - 15 bytes](http://shell-storm.org/shellcode/files/shellcode-600.php) *by beosroot*
* [OpenBSD/x86 - execve(/bin/sh) - 23 bytes](http://shell-storm.org/shellcode/files/shellcode-163.php) *by hophet*
* [OpenBSD/x86 - add user w00w00 - 112 bytes](http://shell-storm.org/shellcode/files/shellcode-165.php) *by n/a*
* [OpenBSD/x86 - portbind port 6969 - 148 bytes](http://shell-storm.org/shellcode/files/shellcode-164.php) *by noir*

**OSX**

**PPC**

* [Osx/ppc - Add user r00t - 219 bytes](http://shell-storm.org/shellcode/files/shellcode-128.php) *by B-r00t*
* [Osx/ppc - add inetd backdoor - 222 bytes](http://shell-storm.org/shellcode/files/shellcode-127.php) *by B-r00t*
* [Osx/ppc - create /tmp/suid - 122 bytes](http://shell-storm.org/shellcode/files/shellcode-126.php) *by B-r00t*
* [Osx/ppc - remote findsock by recv() key shellcode](http://shell-storm.org/shellcode/files/shellcode-121.php) *by Dino Dai Zovi*
* [Osx/ppc - Single Reverse TCP](http://shell-storm.org/shellcode/files/shellcode-122.php) *by H D Moore*
* [Osx/ppc - stager sock find peek](http://shell-storm.org/shellcode/files/shellcode-123.php) *by H D Moore*
* [Osx/ppc - stager sock find](http://shell-storm.org/shellcode/files/shellcode-124.php) *by H D Moore*
* [Osx/ppc - stager sock reverse](http://shell-storm.org/shellcode/files/shellcode-125.php) *by H D Moore*
* [Osx/ppc - Bind Shell PORT TCP/8000 - encoder OSXPPCLongXOR - 300 bytes](http://shell-storm.org/shellcode/files/shellcode-312.php) *by H D moore*
* [Osx/ppc - shellcode execve(/bin/sh)](http://shell-storm.org/shellcode/files/shellcode-120.php) *by ghandi*
* [Osx/ppc - execve(/bin/sh,[/bin/sh],NULL)& exit() - 72 bytes](http://shell-storm.org/shellcode/files/shellcode-129.php) *by haphet*
* [Osx/ppc - sync(), reboot() - 32 bytes](http://shell-storm.org/shellcode/files/shellcode-130.php) *by haphet*

**Intel x86-64**

* [Osx/x86-64 - setuid shell x86\_64 - 51 bytes](http://shell-storm.org/shellcode/files/shellcode-736.php) *by Dustin Schultz*
* [Osx/x86-64 - reverse tcp shellcode - 131 bytes](http://shell-storm.org/shellcode/files/shellcode-761.php) *by Jacob Hammack*
* [Osx/x86-64 - universal OSX dyld ROP shellcode](http://shell-storm.org/shellcode/files/shellcode-786.php) *by pa\_kt*

**Intel x86**

* [Osx/x86 - execve(/bin/sh) - 24 bytes](http://shell-storm.org/shellcode/files/shellcode-692.php) *by Simon Derouineau*

**Solaris**

**MIPS**

* [Solaris/mips - connect-back (with XNOR encoded session) - 600 bytes](http://shell-storm.org/shellcode/files/shellcode-241.php) *by Russell Sanford*
* [Solaris/mips - download and execute - 278 bytes](http://shell-storm.org/shellcode/files/shellcode-240.php) *by Russell Sanford*

**SPARC**

* [Solaris/sparc - setreuid(geteuid()), setregid(getegid()), execve /bin/sh](http://shell-storm.org/shellcode/files/shellcode-385.php) *by Claes M. Nyberg*
* [Solaris/sparc - Bind /bin/sh TCP port 2001](http://shell-storm.org/shellcode/files/shellcode-115.php) *by ghandi*
* [Solaris/sparc - portbind | port 6666 - 240 bytes](http://shell-storm.org/shellcode/files/shellcode-117.php) *by lhall*
* [Solaris/sparc - setreuid - 56 bytes](http://shell-storm.org/shellcode/files/shellcode-118.php) *by lhall*
* [Solaris/sparc - execve(/bin/sh) - 52 bytes](http://shell-storm.org/shellcode/files/shellcode-116.php) *by LSD*
* [Solaris/sparc - Single bind TCP shell](http://shell-storm.org/shellcode/files/shellcode-313.php) *by vlad902*

**Intel x86**

* [Solaris/x86 - setuid(0) /bin/cat //etc/shadow - 61 bytes](http://shell-storm.org/shellcode/files/shellcode-769.php) *by John Babio*
* [Solaris/x86 - Remote Download file - 79 bytes](http://shell-storm.org/shellcode/files/shellcode-616.php) *by Jonathan Salwan*
* [Solaris/x86 - execve(/bin/sh, /bin/sh, NULL) - 27 bytes](http://shell-storm.org/shellcode/files/shellcode-613.php) *by Jonathan Salwan*
* [Solaris/x86 - add services and execve inetd - 201 bytes](http://shell-storm.org/shellcode/files/shellcode-114.php) *by n/a*
* [Solaris/x86 - execve /bin/sh toupper evasion - 84 bytes](http://shell-storm.org/shellcode/files/shellcode-113.php) *by n/a*
* [Solaris/x86 - execve /bin/sh - 43 bytes](http://shell-storm.org/shellcode/files/shellcode-386.php) *by shellcode.com.ar*
* [Solaris/x86 - setuid(0)&execve(//bin/sh)&exit(0) - 39 bytes](http://shell-storm.org/shellcode/files/shellcode-112.php) *by sm4x*
* [Solaris/x86 - setuid(0)&execve(/bin/cat, /etc/shadow)&exit(0) - 59 bytes](http://shell-storm.org/shellcode/files/shellcode-111.php) *by sm4x*

**Windows**

* [Windows/64 - Obfuscated Shellcode x86/x64 Download And Execute [Use PowerShell] - Generator](http://shell-storm.org/shellcode/files/shellcode-899.php) *by Ali Razmjoo*
* [Windows/64 - Add Admin, enable RDP, stop firewall and start terminal service - 1218 bytes](http://shell-storm.org/shellcode/files/shellcode-898.php) *by Ali Razmjoo*
* [Windows/64 - (URLDownloadToFileA) download and execute - 218+ bytes](http://shell-storm.org/shellcode/files/shellcode-150.php) *by Weiss*
* [Windows/64 - Windows Seven x64 (cmd) - 61 bytes](http://shell-storm.org/shellcode/files/shellcode-627.php) *by agix*
* [Windows - Add Admin, enable RDP, stop firewall and start terminal service - 1218 bytes](http://shell-storm.org/shellcode/files/shellcode-897.php) *by Ali Razmjoo*
* [Windows - Add Admin User Shellcode - 194 bytes](http://shell-storm.org/shellcode/files/shellcode-874.php) *by Giuseppe D'Amore*
* [Windows - Safari JS JITed shellcode - exec calc (ASLR/DEP bypass)](http://shell-storm.org/shellcode/files/shellcode-673.php) *by Alexey Sintsov*
* [Windows - Vista/7/2008 - download and execute file via reverse DNS channel](http://shell-storm.org/shellcode/files/shellcode-767.php) *by Alexey Sintsov*
* [Windows - sp2 (En + Ar) cmd.exe - 23 bytes](http://shell-storm.org/shellcode/files/shellcode-568.php) *by AnTi SeCuRe*
* [Windows - add new local administrator - 326 bytes](http://shell-storm.org/shellcode/files/shellcode-714.php) *by Anastasios Monachos*
* [Windows - pro sp3 (EN) - add new local administrator 113 bytes](http://shell-storm.org/shellcode/files/shellcode-715.php) *by Anastasios Monachos*
* [Windows - xp sp2 PEB ISbeingdebugged shellcode - 56 bytes](http://shell-storm.org/shellcode/files/shellcode-802.php) *by Anonymous*
* [Windows - XP Pro Sp2 English Message-Box Shellcode - 16 Bytes](http://shell-storm.org/shellcode/files/shellcode-526.php) *by Aodrulez*
* [Windows - XP Pro Sp2 English Wordpad Shellcode - 15 bytes](http://shell-storm.org/shellcode/files/shellcode-513.php) *by Aodrulez*
* [Windows - Write-to-file Shellcode](http://shell-storm.org/shellcode/files/shellcode-681.php) *by Brett Gervasoni*
* [Windows - telnetbind by winexec - 111 bytes](http://shell-storm.org/shellcode/files/shellcode-148.php) *by DATA\_SNIPER*
* [Windows - useradd shellcode for russian systems - 318 bytes](http://shell-storm.org/shellcode/files/shellcode-390.php) *by Darkeagle*
* [Windows - XP SP3 English MessageBoxA - 87 bytes](http://shell-storm.org/shellcode/files/shellcode-691.php) *by Glafkos Charalambous*
* [Windows - SP2 english ( calc.exe ) - 37 bytes](http://shell-storm.org/shellcode/files/shellcode-576.php) *by Hazem mofeed*
* [Windows - SP3 english ( calc.exe ) - 37 bytes](http://shell-storm.org/shellcode/files/shellcode-577.php) *by Hazem mofeed*
* [Windows - Shellcode (cmd.exe) for XP SP2 Turkish - 26 Bytes](http://shell-storm.org/shellcode/files/shellcode-573.php) *by Hellcode*
* [Windows - Shellcode (cmd.exe) for XP SP3 English - 26 Bytes](http://shell-storm.org/shellcode/files/shellcode-572.php) *by Hellcode*
* [Windows - XP SP3 EN Calc Shellcode - 16 Bytes](http://shell-storm.org/shellcode/files/shellcode-739.php) *by John Leitch*
* [Windows - win32/PerfectXp-pc1/sp3 (Tr) Add Admin Shellcode - 112 bytes](http://shell-storm.org/shellcode/files/shellcode-776.php) *by KaHPeSeSe*
* [Windows - PEB Kernel32.dll ImageBase Finder - 49 Bytes](http://shell-storm.org/shellcode/files/shellcode-393.php) *by Koshi*
* [Windows - PEB Kernel32.dll ImageBase Finder Alphanumeric - 67 bytes](http://shell-storm.org/shellcode/files/shellcode-394.php) *by Koshi*
* [Windows - PEB!NtGlobalFlags shellcode - 14 bytes](http://shell-storm.org/shellcode/files/shellcode-156.php) *by Koshi*
* [Windows - XP sp3 (Ru) WinExec+ExitProcess cmd shellcode - 12 bytes](http://shell-storm.org/shellcode/files/shellcode-581.php) *by Lord Kelvin*
* [Windows - Reverse Generic Shellcode w/o Loader - 249 bytes](http://shell-storm.org/shellcode/files/shellcode-389.php) *by Matthieu Suiche*
* [Windows - Pop up message box (XP/SP2) - 110 bytes](http://shell-storm.org/shellcode/files/shellcode-161.php) *by Omega7*
* [Windows - sp3 (FR) Sleep - 14 bytes](http://shell-storm.org/shellcode/files/shellcode-494.php) *by Optix*
* [Windows - XP download and exec source](http://shell-storm.org/shellcode/files/shellcode-146.php) *by Peter Winter-Smith*
* [Windows - Allwin MessageBoxA - 238 bytes](http://shell-storm.org/shellcode/files/shellcode-648.php) *by RubberDuck*
* [Windows - Allwin WinExec add new local administrator + ExitProcess Shellcode - 272 bytes](http://shell-storm.org/shellcode/files/shellcode-766.php) *by RubberDuck*
* [Windows - Allwin WinExec cmd.exe + ExitProcess Shellcode - 195 bytes](http://shell-storm.org/shellcode/files/shellcode-662.php) *by RubberDuck*
* [Windows - Shellcode Collection - (calc) 19 bytes](http://shell-storm.org/shellcode/files/shellcode-567.php) *by SkuLL-HacKeR*
* [Windows - null-free 32-bit Windows download and LoadLibrary shellcode - 164 bytes](http://shell-storm.org/shellcode/files/shellcode-700.php) *by SkyLined*
* [Windows - null-free 32-bit Windows shellcode that executes calc.exe - 100 bytes](http://shell-storm.org/shellcode/files/shellcode-701.php) *by SkyLined*
* [Windows - null-free 32-bit Windows shellcode that shows a message box - 140 bytes](http://shell-storm.org/shellcode/files/shellcode-702.php) *by SkyLined*
* [Windows - null-free bindshell for Windows 5.0-6.0 all service packs](http://shell-storm.org/shellcode/files/shellcode-484.php) *by SkyLined*
* [Windows - XP sp2 (FR) Sellcode cmd.exe - 32 bytes](http://shell-storm.org/shellcode/files/shellcode-149.php) *by Stack*
* [Windows - XP/sp2 (EN) cmd.exe - 23 bytes](http://shell-storm.org/shellcode/files/shellcode-482.php) *by Stack*
* [Windows - XP Professional SP2 ita calc.exe - 36 bytes](http://shell-storm.org/shellcode/files/shellcode-580.php) *by Stoke*
* [Windows - WinExec() Command Parameter - 104 bytes](http://shell-storm.org/shellcode/files/shellcode-391.php) *by Weiss*
* [Windows - download and execute - 124 bytes](http://shell-storm.org/shellcode/files/shellcode-392.php) *by Weiss*
* [Windows - Download and Execute Shellcode Generator](http://shell-storm.org/shellcode/files/shellcode-159.php) *by YAG KOHHA*
* [Windows - sp3 (Tr) Add Admin Account Shellcode - 127 bytes](http://shell-storm.org/shellcode/files/shellcode-706.php) *by ZoRLu*
* [Windows - sp3 (Tr) MessageBoxA Shellcode - 109 bytes](http://shell-storm.org/shellcode/files/shellcode-703.php) *by ZoRLu*
* [Windows - sp3 (Tr) calc.exe Shellcode 53 bytes](http://shell-storm.org/shellcode/files/shellcode-704.php) *by ZoRLu*
* [Windows - sp3 (Tr) cmd.exe Shellcode - 42 bytes](http://shell-storm.org/shellcode/files/shellcode-710.php) *by ZoRLu*
* [Windows - sp3 (Tr) cmd.exe Shellcode 52 bytes](http://shell-storm.org/shellcode/files/shellcode-705.php) *by ZoRLu*
* [Windows - Xp Pro SP3 Fr (calc.exe) - 31 Bytes](http://shell-storm.org/shellcode/files/shellcode-612.php) *by agix*
* [Windows - XP PRO SP3 - Full ROP calc shellcode](http://shell-storm.org/shellcode/files/shellcode-830.php) *by b33f*
* [Windows - xp pro sp3 (calc) - 57 bytes](http://shell-storm.org/shellcode/files/shellcode-579.php) *by cr4wl3r*
* [Windows - win32/xp pro sp3 MessageBox shellcode - 11 bytes](http://shell-storm.org/shellcode/files/shellcode-751.php) *by d3c0der*
* [Windows - download & exec shellcode - 226 bytes+](http://shell-storm.org/shellcode/files/shellcode-162.php) *by darkeagle*
* [Windows - Shellcode Checksum Routine](http://shell-storm.org/shellcode/files/shellcode-697.php) *by dijital1*
* [Windows - IsDebuggerPresent ShellCode (NT/XP) - 39 bytes](http://shell-storm.org/shellcode/files/shellcode-153.php) *by ex-pb*
* [Windows - PEB method (9x/NT/2k/XP) - 29 bytes](http://shell-storm.org/shellcode/files/shellcode-388.php) *by loco*
* [Windows - connectback, receive, save and execute shellcode](http://shell-storm.org/shellcode/files/shellcode-157.php) *by loco*
* [Windows - Bind Shell (NT/XP/2000/2003) - 356 bytes](http://shell-storm.org/shellcode/files/shellcode-173.php)*by metasploit*
* [Windows - Create Admin User Account (NT/XP/2000) - 304 bytes](http://shell-storm.org/shellcode/files/shellcode-175.php) *by metasploit*
* [Windows - Vampiric Import Reverse Connect - 179 bytes](http://shell-storm.org/shellcode/files/shellcode-174.php)*by metasploit*
* [Windows - PEB method (9x/NT/2k/XP)](http://shell-storm.org/shellcode/files/shellcode-260.php) *by oc192*
* [Windows - eggsearch shellcode - 33 bytes](http://shell-storm.org/shellcode/files/shellcode-755.php) *by oxff*
* [Windows - XP-sp1 portshell on port 58821 - 116 bytes](http://shell-storm.org/shellcode/files/shellcode-147.php) *by silicon*
* [Windows - XP SP3 addFirewallRule](http://shell-storm.org/shellcode/files/shellcode-569.php) *by sinn3r*
* [Windows - PEB method (9x/NT/2k/XP) - 31 bytes](http://shell-storm.org/shellcode/files/shellcode-387.php) *by twoci*
* [Windows - Beep Shellcode (SP1/SP2) - 35 bytes](http://shell-storm.org/shellcode/files/shellcode-152.php) *by xnull*

How to Prevent Shellcode Injection

When you first hear ‘**shellcode**,’ you might think of shell scripting. Surprisingly enough, neither shellcode nor **shellcode injection** have anything to do with **shell scripting**. Keep reading to learn more!

Let’s say the network of your organization is a big, beautiful garden surrounded with thick high walls, and has a fortified gate. In this analogy, the walls represent your **security measures** that keep intruders out, and the gate represents**secure access points** through which your employees can get in. If your walls are thick and high enough, they can keep the hackers away but unfortunately, even **the safest security systems** have vulnerable spots. Most hackers try to find these spots and exploit them instead of loudly banging the front door or hitting the walls with a sledgehammer.

Shellcode injection is a sophisticated way of finding a vulnerable spot on the **cyber security layer** of an organization and exploiting it for malicious purposes. In this article, we will explain **what shellcode injection is** and how you can protect your organization from it.

**What is a shellcode?**

In the context of hacking, a shellcode refers to a small piece of code that is used for exploiting a vulnerability of a software. The name is a node to shellcode’s ability to start a command shell through which the hacker gains the control of the compromised device. In time, the definition of the ‘shellcode’ expanded to encompass any piece of code that can carry out a similar task.

Often written in machine code, a shellcode can be local or remote. A **local shellcode** is often opted for by a hacker who has little to no access to the machine but can successfully exploit a vulnerability like a buffer overflow. On the other hand, a **remote shellcode** is preferred by a hacker who desires to attack the target machine through a local network, remote network or intranet. A vast majority of remote shellcodes employ **TCP/IP** socket connections.

**What is shellcode injection?**

Simply put, shellcode injection is a hacking technique where the hacker exploits **vulnerable programs**. The hacker infiltrates into the vulnerable programs and makes it execute their own code.

The shellcode injection process consists of three steps:

1. Crafting the shellcode
2. Injecting the shellcode
3. Modifying the execution flow and/or running the shellcode

To **craft the shellcode**, the hacker needs to craft a compiled machine code through writing and assembling the code, and extracting bytes from the machine code. For injecting the shellcode, the program is manipulated to take the input and read the external files.

**How can I protect my organization from shellcode injection?**

Often, hackers try to reverse engineer programs to find their vulnerable spots. You can start by making sure that all the vulnerabilities of the software you use are alleviated. In addition, you can also address **buffer overflows** to make sure that your organization is safe from shellcode injection.

If you want to make sure that your organization is protected 24/7 and all the vulnerabilities on your security posture are addressed properly, you should take a closer look at our**SIEM** and**SOAR** products. They offer tailor made solutions for your **cyber security** needs and make sure your team’s efforts and actions on security is automated against possible attacks.

<https://www.sentinelone.com/blog/malicious-input-how-hackers-use-shellcode/>

<https://www.slideshare.net/DhavalKapil/shellcode-injection>

<https://en.wikipedia.org/wiki/Shellcode>

<https://www.httpcs.com/en/php-shell-code-injection-vulnerability>

<https://dhavalkapil.com/blogs/Shellcode-Injection/>

A Beginner’s Guide to Windows Shellcode Execution Techniques

This blog post is aimed to cover basic techniques of how to execute shellcode within the memory space of a process. The background idea for this post is simple: New techniques to achieve stealthy code execution appear every day and it’s not always trivial to break these new concepts into their basic parts to understand how they work. By explaining basic concepts of In-Memory code execution this blog post aims to improve everyone’s ability to do this.

By Carsten Sandker

Security Consultant

24 JUL 2019

[Vulnerabilities And Exploits](https://www.contextis.com/blog/category/vulnerabilities-and-exploits)

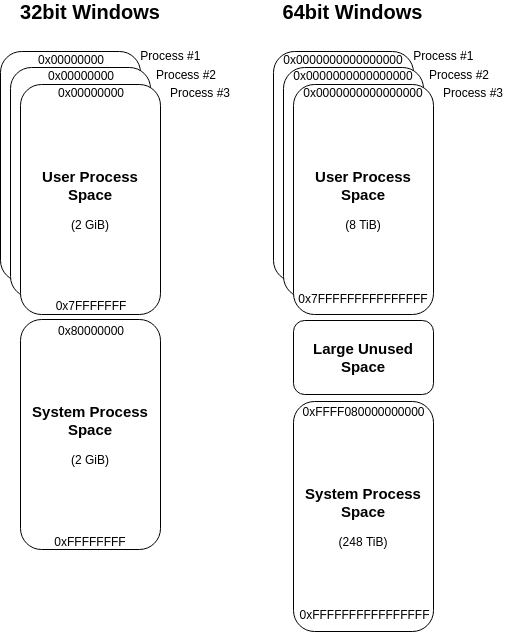
In essence the following four execution techniques will be covered:

* Dynamic Allocation of Memory
* Function Pointer Execution
* .TEXT-Segment Execution
* RWX-Hunter Execution

Especially the first two techniques are very widely known and most should be familiar with these, however, the latter two might be new to some.   
Each of these techniques describes a way of executing code in a different memory section, therefore it is necessary to review a processes memory layout as a first step.

A Processes Memory Layout

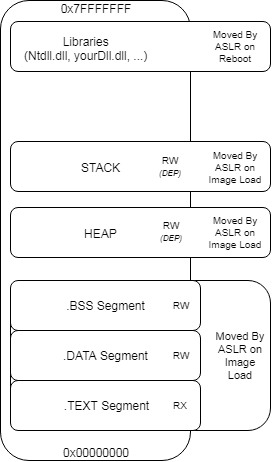
The first concept that needs to be understood is that the entire virtual memory space is split into two relevant parts: Virtual memory space reserved for user processes (user space) and virtual memory space reserved for system processes (kernel space), as shown below:



This visual representation is based on Microsoft’s description given here: <https://docs.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/virtual-address-spaces>.

The first takeaway from this is that each process gets its own, private virtual address space, where the “kernel space” is kind of a “shared environment”, meaning each kernel process can read/write to virtual memory anywhere it wants to. Please note the latter is only true for environments without Virtualization-based Security (VBS), but that’s a different topic.

The representation above shows what the global virtual address space looks like, let’s break this down for a single process:



A single processes virtual memory space consists of multiple sections that are placed somewhere within the available space boundaries by Address Space Layout Randomization (ASLR). Most of these sections should be familiar, but to keep everyone on the same page, here is a quick rundown of these sections:

**.TEXT Segment**: This is where the executable process image is placed. In this area you will find the main entry of the executable, where the execution flow starts.

**.DATA Segment:** The .DATA section contains globally initialized or static variables. Any variable that is not bound to a specific function is stored here.

**.BSS Segment:** Similar to the .DATA segment, this section holds any uninitialized global or static variables.

**HEAP:** This is where all your dynamic local variables are stored. Every time you create an object for which the space that is needed is determined at run time, the required address space is dynamically assigned within the HEAP (usually using alloc() or similar system calls).

**STACK:** The stack is the place every static local variable is assigned to. If you initialize a variable locally within a function, this variable will be placed on the STACK.

Dynamically Allocate Memory

After defining the basics, let’s have a look on what is needed to execute shellcode within your process memory space. In order to execute your shellcode you need to complete the following three checks:

1. You need virtual address space that is marked as executable (otherwise DEP will throw an exception)
2. You need to get your shellcode into that address space
3. You need to direct the code flow to that memory region

The text book method to complete these three steps is to use WinAPI calls to dynamically allocate readable, writeable and executable (RWX) memory and start a thread pointing to the freshly allocated memory region. Coding this in C would look like this:

#include <windows.h>

int main()

{

char shellcode[] = "\xcc\xcc\xcc\xcc\x41\x41\x41\x41";

// Alloc memory

LPVOID addressPointer = VirtualAlloc(NULL, sizeof(shellcode), 0x3000, 0x40);

// Copy shellcode

RtlMoveMemory(addressPointer, shellcode, sizeof(shellcode));

// Create thread pointing to shellcode address

CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)addressPointer, NULL, 0, 0);

// Sleep for a second to wait for the thread

Sleep(1000);

return 0;

}

As it will be shown in the following screenshots, when compiling and executing the above code, the shellcode will be executed from the heap, which is by default protected by the system wide Data Execution Prevention (DEP) policy that has been introduced in Windows XP (for details on this see: <https://docs.microsoft.com/en-us/windows/desktop/memory/data-execution-prevention>).  For DEP enabled processes this would prevent code execution in this memory region. To overcome this burden we ask the system to mark the required memory region as RWX. This is done by specifying the last argument to VirtualAlloc to be 0x40, which is equivalent to PAGE\_EXECUTE\_READWRITE, as specified in <https://docs.microsoft.com/en-us/windows/desktop/memory/memory-protection-constants>.

So far so good, but how would that code behave in memory? To analyse this we’ll use WinDbg (<https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/debugger-download-tools>). If you have never set up WinDbg before, refer to the following screenshot to get an idea of how to point WinDbg to your source code, list all loaded modules, set a break point and run your program:

After entering “g” in the WinDbg’s command line the program will break into the main function of your executable. If you then step through your code to the point after *RtlMoveMemory* is called, you will face something like the following in WinDbg:

As indicated by the violet line we are currently right after the call to *RtlMoveMemory*. If we refer to the code above, *RtlMoveMemory* takes a Pointer from *VirtualAlloc* to write our shellcode to the given location. As the pointer returned from *VirtualAlloc* is the first argument to *RtlMoveMemory*, it will be pushed on stack last (within register ecx) before calling the function, as function parameters get pushed on the stack in reverse order. If we would have stopped right before the call to *RtlMoveMemory* the ecx register would show the address location to be ‘*0x420000’*, which in the above screenshot has been placed into the eax register after the WinAPI call.

Inspecting the memory location at address 0x*420000* in the screenshot above, shows that our shellcode has been placed at this address. Furthermore, note that the stack base address (ebp) is shown as *0x5afa34* and the stack pointer (esp – the top address of the stack) is pointing to *0x5af938*, spanning the stack across the addresses in this range. As the memory location of the shellcode is not within the stack range we can safely conclude it has been placed on the heap instead.

|  |  |
| --- | --- |
| **The key takeaway parts:** WinAPI system calls are used to dynamically allocate RWX memory within the heap, move the shellcode into the newly allocated memory region and start a new execution thread. | |
| **The PROs** Using WinAPI calls is the textbook method to execute code and very reliable.  The allocated memory region is not only executable, but also writeable and readable, which allows modification of the shellcode within this memory region. This allows shellcode encoding/encryption. | **The CONs** The usage of WinAPI calls is very easily detectable by mature AV/EDR systems. |

Function Pointer Execution

In contrast to the vanilla approach above, another technique to execute shellcode within memory is by the use of function pointers, as shown in the code snippet below:

#include <windows.h>

int main()

{

char buf[] = "\xcc\xcc\xcc\xcc";

// One way to do it

int (\*func)();

func = (int (\*)()) (void\*)buf;

(int)(\*func)();

// Shortcut way to do it

// (\*(int(\*)()) buf)();

// sleep for a second

Sleep(1000);

return 0;

}

The way this code works is as follows:

* A pointer to a function is declared, in the above code snippet that function pointer is named ‘func’
* The declared function pointer is than assigned the address of the code to execute (as any variable would be assigned with a value, the func pointer is assigned with an address)
* Finally the function pointer is called, meaning the execution flow is directed to the assigned address.

Applying the same steps as above we can analyse this in memory with WinDbg, which takes us to the following:

The key steps that lead to code execution in this case are the following:

* The shellcode, contained in a local variable, is pushed onto the stack during initialization (relatively close the ebp, as this is one of the first things to happen in the main-method)
* The shellcode is loaded from the stack into eax as shown at address 0x00fd1753
* The shellcode is executed by calling eax as shown at address 0x00fd1758

Referring back to the virtual memory layout of a single process shown above, it is stated that the stack is only marked as RW memory section with regards to DEP. The same problem occurred before with dynamic allocation of heap memory, in which case a WinAPI function (VirtualAlloc) was used to mark the memory section as executable. In this case we’re not using any WinAPI functions, but luckily we can simply disable DEP for the compiled executable by setting the /NXCOMPAT:NO flag (for VisualStudio this can be set within the advanced Linker options). The result is happily executing shellcode.

|  |  |
| --- | --- |
| **The key takeaway parts:** A function pointer is used to call shellcode, allocated as local variable on the stack. | |
| **The PROs** No WinAPI calls are used, which could be used to avoid AV/EDR detection.  The stack is writeable and readable, which allows modification of the shellcode within this memory region. This allows shellcode encoding/encryption. | **The CONs** By default DEP prevents code execution within the stack, which requires to compile the code without DEP support. A system wide DEP enforcement would prevent the code execution. |

.TEXT Segment Execution

So far we have achieved code execution within the heap and the stack, which are both not executable by default and therefore we were required to use WinAPI functions and disabling DEP respectively to overcome this.  
We could avoid using such methods with code execution in a memory region that is already marked as executable.  
A quick reference back to the memory layout above shows that the .TEXT segment is such a memory region.

The .TEXT segment needs to be executable, because this is the section that contains your executable code, such as your main-function.  
Sounds like a suitable place for shellcode execution, but how can we place and execute shellcode in this section. We can’t use WinAPI functions to simply move our shellcode into here, because the .TEXT segment is not writable and we can’t use function pointers as we don’t have a reference in here to point at.

The solution here is Inline-Assembly (<https://docs.microsoft.com/en-us/cpp/assembler/inline/inline-assembler?view=vs-2019>), which can be used to embed our shellcode within our main-method.

Shoutout to [@MrUn1k0d3r](https://twitter.com/mrun1k0d3r?lang=en) at this point, who showed an implementation of this technique here: <https://github.com/Mr-Un1k0d3r/Shellcoding>. A slightly shortened version of his code shown below:

#include <Windows.h>

int main() {

asm(".byte 0xde,0xad,0xbe,0xef,0x00\n\t"

"ret\n\t");

return 0;

}

To compile this code the GCC compiler is required, due to the use of the “.byte” directive. Luckily there is a GCC compiler contained in the MinGW project and we can easily compile this as follows:

*mingw32-gcc.exe -c Main.c -o Main.o*

*mingw32-g++.exe -o Main.exe Main.o*

Viewing this in IDA reveals that our shellcode has been embed into the .TEXT segment (IDA is just a bit more visual than WinDbg here):

The defined shellcode ‘0xdeadbeef’ has been placed within the assembled code right after the call to *\_\_main*, which is used as initialization routine. As soon as the *\_\_main* function finishes the initialization our shellcode is executed right away.

|  |  |
| --- | --- |
| **The key takeaway parts:** Inline Assembly is used to embed shellcode right within the .TEXT segment of the executable program. | |
| **The PROs** No WinAPI calls are used, which could be used to avoid AV/EDR detection. | **The CONs** The .TEXT segment is not writeable, therefore no shellcode encoders/encrypters can be used.  As such malicious shellcode is easily detectable by AVs/EDRs if not customized. |

RWX-Hunter Execution

Last, but not least, after using the default executable .TEXT segment for code execution and creating non-default executable memory sections with WinAPI functions and by disabling DEP, there is one last path to go, which is: Searching for memory sections that have already been marked as read (R), write (W) and executable (X) – which i stumbled across reading [@subTee](https://twitter.com/subTee) post on InstallUtil’s help-functionality code exec.

The basic idea for the RWX-Hunter is running through your processes virtual memory space searching for a memory section that is marked as RWX.  
The attentive reader will now notice that this only fulfils only 1/3 of the defined steps for code execution, that i set up initially, which is: Finding executable memory.  The task of how to get your shellcode into this memory region and how to direct the code flow to there is not covered with this approach. However, the concept still fits well in this guide and is therefore worth mentioning.

The first question that needs to be answered is the range of where to search for RWX memory sections. Once again referring back to the initial description of a processes private virtual memory space it is stated that a processes memory space spans from 0x00000000 to 0x7FFFFFFFF, so this should be the search range.

The Code-Snippet, which I’ve ported to C from [@subTee](https://twitter.com/subTee) C# gist [here](https://gist.github.com/caseysmithrc/0b40f1ec0340edd5efe54f1111bba325), to implement this could look like the following (honestly i prefer this in C#, but since all of the above code is in C i stick to consistency):

long MaxAddress = 0x7fffffff;

long address = 0;

do

{

MEMORY\_BASIC\_INFORMATION m;

int result = VirtualQueryEx(process, (LPVOID)address, &m, sizeof(MEMORY\_BASIC\_INFORMATION));

if (m.AllocationProtect == PAGE\_EXECUTE\_READWRITE)

{

printf("YAAY - RWX found at 0x%x\n", m.BaseAddress);

return m.BaseAddress;

}

if (address == (long)m.BaseAddress + (long)m.RegionSize)

break;

address = (long)m.BaseAddress + (long)m.RegionSize;

} while (address <= MaxAddress);

This implementation is pretty much straight forward for what we want to achieve. A processes private virtual memory space (the user land virtual memory space) is searched for a memory section that is marked with PAGE\_EXECUTE\_READWRITE, which again maps to 0x40 as seen in previous examples. If that space is found it is returned, if not the next search address is set the next memory region (BaseAddress + Memory Region).

To complete this into code execution your shellcode needs then to be moved to that found memory region and executed. An easy way to do this would to fall back to WinAPI calls as shown in the first technique, but the CONs of that approach should be considered as stated above. At the end of this post I’ll share usable PoCs for references of how this could be implemented (for the RWX-Hunter you might also want to check out [@subTee’s](https://twitter.com/subTee) implementation linked above).

For the creative minds: There are also other techniques (some of them are surely still to be uncovered) to achieve steps 2. & 3.. To get shellcode into the found memory region (Step 2.) a Write-What-Where condition could become useful, as for example used in the AtomBombing  technique that came up a few years back (the technique was initially published [here](https://blog.ensilo.com/atombombing-brand-new-code-injection-for-windows)). To finally execute the placed shellcode (Step 3.) ROP-gadgets might become useful… (a good introduction to ROP gadgets can be found [here](https://resources.infosecinstitute.com/return-oriented-programming-rop-attacks/) or on [Wikipedia](https://en.wikipedia.org/wiki/Return-oriented_programming)).

|  |  |
| --- | --- |
| **The key takeaway parts:** A readable, writeable and executable (RWX) memory section is searched within a processes memory space to avoid dynamic creation of such. | |
| **The PROs** A call to VirtuallAlloc/VirtuallAllocEx is avoided and no RWX memory is dynamically created by the exploiting process. | **The CONs** Advanced knowledge is needed to avoid WinAPI calls to place shellcode and redirection of code flow to the placed shellcode. |

**And Finally:**

A complete set of working PoCs is published here: <https://github.com/csandker/inMemoryShellcode>