WIKIPEDIA
The Free Encyclopedia

Шеллкод

В хакерстве шеллкод — это небольшой фрагмент кода, используемый в качестве полезной нагрузки при эксплуатации уязвимости программного обеспечения . Он называется «шеллкод», потому что обычно запускает командную оболочку , из которой злоумышленник может управлять взломанной машиной, но любой фрагмент кода, который выполняет аналогичную задачу, можно назвать шеллкодом. Поскольку функция полезной нагрузки не ограничивается простым созданием оболочки, некоторые полагают, что название шеллкод недостаточно. [1] Однако попытки заменить этот термин не получили широкого признания. Шеллкод обычно пишется в машинном коде .

При создании шелл-кода обычно желательно сделать его одновременно и небольшим, и исполняемым, что позволяет использовать его в максимально широком спектре ситуаций. [2] В ассемблерном коде одна и та же функция может быть выполнена множеством способов, и существует некоторое разнообразие в длине опкодов, которые могут использоваться для этой цели; хорошие авторы шелл-кода могут использовать эти небольшие опкоды для создания более компактного шелл-кода. [3] Некоторые достигли минимально возможного размера, сохраняя при этом стабильность. [4]

Типы шеллкода

Шелл-код может быть *локальным* или *удаленным* , в зависимости от того, дает ли он злоумышленнику контроль над машиной, на которой он запущен (локальный), или над другой машиной через сеть (удаленный).

Местный

Локальный шеллкод используется злоумышленником, имеющим ограниченный доступ к машине, но способным использовать уязвимость, например, переполнение буфера , в более привилегированном процессе на этой машине. При успешном выполнении шеллкод предоставит злоумышленнику доступ к машине с теми же более высокими привилегиями, что и целевой процесс.

Удаленный

Удаленный шеллкод используется, когда злоумышленник хочет нацелиться на уязвимый процесс, запущенный на другой машине в локальной сети , интрасети или удаленной сети . При успешном выполнении шеллкод может предоставить злоумышленнику доступ к целевой машине по сети. Удаленные шеллкоды обычно используют стандартные соединения сокетов TCP/IP, чтобы позволить злоумышленнику получить доступ к оболочке на целевой машине. Такой шеллкод можно классифицировать на основе того, как это соединение установлено: если шеллкод устанавливает соединение, он называется «обратной оболочкой» или шеллкодом *обратного подключения* , потому что шеллкод *подключается обратно* к машине злоумышленника. С другой стороны, если злоумышленник устанавливает соединение, шеллкод называется *bindshell* , потому что шеллкод *привязывается* к определенному порту на машине жертвы. Существует своеобразный шеллкод, называемый *bindshell random port* , который пропускает часть привязки и прослушивает случайный порт, предоставленный операционной системой . Из-за этого *bindshell random port (https://github.com/geyslan/SLAE/blob/master/improvements/tiny_shell_bind_tcp_random_port_x86_64.asm)* стал

наименьшим стабильным шеллкодом bindshell для x86_64 , доступным на сегодняшний день. Третий, гораздо менее распространенный тип — шеллкод *повторного использования сокета* . Этот тип шеллкода иногда используется, когда эксплойт устанавливает соединение с уязвимым процессом, который не закрыт до запуска шеллкода. Затем шеллкод может *повторно использовать* это соединение для связи с атакующим. Шеллкод повторного использования сокета более сложен, поскольку шеллкоду необходимо выяснить, какое соединение использовать повторно, а на машине может быть открыто много соединений. [5] *(https://github.com/geyslan/SLAE/blob/master/improvements/tiny_shell_bind_tcp_random_port_x86_64.asm)*

Брандмауэр может использоваться для обнаружения исходящих соединений, созданных шеллкодом connect-back , а также входящих соединений, созданных bindshells. Таким образом, они могут обеспечить некоторую защиту от злоумышленника, даже если система уязвима, не давая злоумышленнику подключиться к оболочке, созданной шеллкодом. Одна из причин, по которой иногда используется шеллкод с повторным использованием сокета, заключается в том, что он не создает новых соединений и, следовательно, его сложнее обнаружить и заблокировать.

Загрузите и выполните

Загрузка и выполнение — это тип удаленного шелл-кода, который *загружает и запускает* некоторую форму вредоносного ПО на целевой системе. Этот тип шелл-кода не создает оболочку, а скорее инструктирует машину загрузить определенный исполняемый файл из сети, сохранить его на диск и запустить его. В настоящее время он обычно используется в атаках с использованием drive-by download , когда жертва посещает вредоносную веб-страницу, которая в свою очередь пытается запустить такую загрузку и выполнить шелл-код, чтобы установить программное обеспечение на машине жертвы. Разновидность этого типа шелл-кода загружает и загружает библиотеку . [6] [7] Преимущества этого метода в том, что код может быть меньше, что он не требует, чтобы шелл-код запускал новый процесс на целевой системе, и что шелл-коду не нужен код для очистки целевого процесса, поскольку это может сделать библиотека, загруженная в процесс.

Постановочный

Когда объем данных, которые злоумышленник может внедрить в целевой процесс, слишком ограничен для непосредственного выполнения полезного шелл-кода, его можно выполнить поэтапно. Сначала выполняется небольшой фрагмент шелл-кода (этап 1). Затем этот код загружает в память процесса более крупный фрагмент шелл-кода (этап 2) и выполняет его.

Охота за яйцами

Это еще одна форма *поэтапного* шеллкода, которая используется, если злоумышленник может внедрить в процесс более крупный шеллкод, но не может определить, где в процессе он окажется. Небольшой шеллкод *для поиска яиц* внедряется в процесс в предсказуемом месте и выполняется. Затем этот код ищет в адресном пространстве процесса более крупный шеллкод (*яйцо*) и выполняет его. [8]

Омлет

Этот тип шеллкода похож на шеллкод *охоты за яйцами* , но ищет несколько небольших блоков данных (*яиц*) и объединяет их в один большой блок (*омлет*), который затем выполняется. Это используется, когда злоумышленник может внедрить в процесс только несколько небольших блоков

данных. [9]

Стратегия выполнения шелл-кода

Эксплойт обычно внедряет шелл-код в целевой процесс до или одновременно с эксплуатацией уязвимости для получения контроля над счетчиком программ . Счетчик программ настраивается так, чтобы указывать на шелл-код, после чего он выполняется и выполняет свою задачу. Внедрение шелл-кода часто выполняется путем сохранения шелл-кода в данных, отправляемых по сети уязвимому процессу, путем предоставления его в файле, который считывается уязвимым процессом, или через командную строку или среду в случае локальных эксплойтов.

Кодирование шелл-кода

Поскольку большинство процессов фильтруют или ограничивают данные, которые могут быть введены, часто необходимо писать шелл-код, чтобы учесть эти ограничения. Это включает в себя создание небольшого кода, безнулевого или алфавитно-цифрового . Были найдены различные решения для обхода таких ограничений, в том числе:

- Оптимизация дизайна и реализации для уменьшения размера шелл-кода.
- Изменения реализации для обхода ограничений в диапазоне байтов, используемых в шелл-коде.
- Самоизменяющийся код , который изменяет ряд байтов своего собственного кода перед их выполнением, чтобы заново создать байты, которые обычно невозможно внедрить в процесс.

Поскольку обнаружение вторжений может обнаруживать сигнатуры простых шелл-кодов, отправляемых по сети, их часто кодируют, делают саморасшифровывающимися или полиморфными, чтобы избежать обнаружения.

Процентное кодирование

Эксплойты, нацеленные на браузеры, обычно кодируют шелл-код в строке JavaScript, используя процентное кодирование , кодирование escape-последовательности " `\uXXXX` " или кодирование сущности . [10] Некоторые эксплойты также дополнительно запутывают закодированную строку шелл-кода, чтобы предотвратить обнаружение системой обнаружения вторжений .

Например, на архитектуре IA-32 вот как `NOP`будут выглядеть две (не действующие) инструкции, первая из которых некодирована:

```
90 НОП
90 НОП
```

Закодированные двойные NOP:

процент-кодирование	`unescape("%u9090")`
литерал юникода	`"\u9090"`
HTML/XML сущность	`"&#x9090;"или"&#37008;"`

Эта инструкция используется в слайдах NOP .

Шелл-код без нулей

Большинство шеллкодов пишутся без использования нулевых байтов, поскольку они предназначены для внедрения в целевой процесс через строки с нулевым завершением . При копировании строки с нулевым завершением она будет скопирована до первого нуля включительно, но последующие байты шеллкода не будут обработаны. При внедрении шеллкода, содержащего нули, таким образом будет внедрена только часть шеллкода, что сделает его неспособным к успешному выполнению.

Чтобы создать шеллкод без нулей из шеллкода, содержащего нулевые байты, можно заменить машинные инструкции, содержащие нули, на инструкции, которые имеют тот же эффект, но не содержат нулей. Например, на архитектуре IA-32 можно заменить эту инструкцию:

```
B8 01000000     MOV EAX,1 // Установить регистр EAX в 0x00000001
```

который содержит нули как часть литерала (`1`расширяется до `0x00000001`) с помощью следующих инструкций:

```
33C0            XOR EAX,EAX // Установить регистр EAX в 0x00000000
40              INC EAX // Увеличить EAX до 0x00000001
```

которые имеют тот же эффект, но требуют меньше байтов для кодирования и не содержат нулей.

Буквенно-цифровой и печатный шелл-код

Буквенно **-цифровой шеллкод** — это шеллкод, который состоит из или собирается при выполнении в полностью буквенно-цифровые символы ASCII или Unicode , такие как 0–9, A–Z и a–z. [11] [12] Этот тип кодирования был создан хакерами , чтобы скрыть рабочий машинный код внутри того, что кажется текстом. Это может быть полезно, чтобы избежать обнаружения кода и позволить коду проходить через фильтры, которые очищают строки от не буквенно-цифровых символов (отчасти такие фильтры были ответом на эксплойты не буквенно-цифрового шеллкода). Похожий тип кодирования называется *печатным кодом* и использует все печатные символы (0–9, A–Z, a–z, !@#%^&*() и т. д.). Аналогичным ограниченным вариантом является *код ECHOable,* не содержащий никаких символов, которые не принимаются командой ECHO . Было показано, что можно создать шеллкод, который выглядит как обычный текст на английском языке. [13] Написание алфавитно-цифрового или печатного кода требует хорошего понимания архитектуры набора инструкций машины(машин), на которой(ых) должен(на) выполняться код. Было продемонстрировано, что можно написать алфавитно-цифровой код, который будет выполняться на более чем одной машине, [14] тем самым составляя исполняемый код с несколькими архитектурами .

При определенных обстоятельствах целевой процесс будет фильтровать любой байт из введенного шелл-кода, который не является печатаемым или буквенно-цифровым символом. При таких обстоятельствах диапазон инструкций, которые могут быть использованы для написания шелл-кода, становится очень ограниченным. Решение этой проблемы было опубликовано Риксом в Phrack 57 [11] , в котором он показал, что можно превратить любой код в буквенно-цифровой код. Часто используемая техника заключается в создании самомодифицирующегося кода, поскольку это позволяет коду изменять свои собственные байты, чтобы включать байты за пределами обычно разрешенного диапазона, тем самым расширяя диапазон инструкций, которые он может использовать. Используя этот трюк, можно создать самомодифицирующийся декодер, который изначально использует только байты в разрешенном диапазоне. Основной код шелл-кода кодируется, также используя только байты в разрешенном диапазоне. Когда запускается выходной шелл-код, декодер

может изменить свой собственный код, чтобы иметь возможность использовать любую инструкцию, которая ему требуется для правильной работы, а затем продолжает декодировать исходный шелл-код. После декодирования шелл-кода декодер передает ему управление, поэтому он может быть выполнен как обычно. Было показано, что можно создать произвольно сложный шелл-код, который выглядит как обычный текст на английском языке. [13]

Шеллкод, защищенный Unicode

Современные программы используют строки Unicode для интернационализации текста. Часто эти программы преобразуют входящие строки ASCII в Unicode перед их обработкой. Строки Unicode, закодированные в UTF-16, используют два байта для кодирования каждого символа (или четыре байта для некоторых специальных символов). Когда строка ASCII (в общем случае Latin-1) преобразуется в UTF-16, нулевой байт вставляется после каждого байта в исходной строке. Obscou доказал в Phrack 61 [12] , что можно написать шелл-код, который может успешно работать после этого преобразования. Существуют программы, которые могут автоматически кодировать любой шелл-код в буквенно-цифровой шелл-код, защищенный от UTF-16, основанные на том же принципе небольшого самомодифицирующегося декодера, который декодирует исходный шелл-код.

Платформы

Большая часть шелл-кода написана в машинном коде из-за низкого уровня, на котором эксплуатируемая уязвимость дает злоумышленнику доступ к процессу. Поэтому шелл-код часто создается для нацеливания на одну конкретную комбинацию процессора , операционной системы и пакета обновления , называемую платформой . Для некоторых эксплойтов из-за ограничений, накладываемых на шелл-код целевым процессом, необходимо создать очень конкретный шелл-код. Однако не исключено, что один шелл-код может работать для нескольких эксплойтов, пакетов обновления, операционных систем и даже процессоров. [15] [16] [17] Такая универсальность обычно достигается путем создания нескольких версий шелл-кода, которые нацелены на различные платформы, и создания заголовка, который переходит к правильной версии для платформы, на которой работает код. При выполнении код ведет себя по-разному для разных платформ и выполняет правильную часть шелл-кода для платформы, на которой он работает.

Анализ шелл-кода

Шеллкод не может быть выполнен напрямую. Чтобы проанализировать, что пытается сделать шеллкод, его необходимо загрузить в другой процесс. Одним из распространенных методов анализа является написание небольшой программы на языке C, которая хранит шеллкод в виде байтового буфера, а затем использование указателя на функцию или встроенного ассемблера для передачи ему выполнения. Другой метод заключается в использовании онлайн-инструмента, такого как shellcode_2_exe, для встраивания шеллкода в предварительно созданную исполняемую оболочку, которую затем можно проанализировать в стандартном отладчике. Существуют также специализированные инструменты анализа шеллкода, такие как проект iDefense sclog, который изначально был выпущен в 2005 году как часть пакета Malcode Analyst Pack. Sclog предназначен для загрузки внешних файлов шеллкода и их выполнения в рамках ведения журнала API. Существуют также инструменты анализа шеллкода на основе эмуляции, такие как приложение `sctest` , которое

является частью кроссплатформенного пакета libemu. Другим инструментом анализа шеллкода на основе эмуляции, созданным на основе библиотеки libemu, является `scdbg` , который включает в себя базовую отладочную оболочку и интегрированные функции отчетности.

Смотрите также

- Буквенно-цифровой код
- Компьютерная безопасность
- Переполнение буфера
- Эксплойт (компьютерная безопасность)
- Переполнение кучи
- Проект Metasploit
- Shell (вычисления)
- Сгребание ракушек
- Переполнение буфера стека
- Уязвимость (вычисления)

Ссылки

1. Фостер, Джеймс К.; Прайс, Майк (2005-04-12). *Сокеты, шеллкод, портирование и кодирование: эксплойты обратного проектирования и кодирование инструментов для специалистов по безопасности* (https://books.google.com/books?id=ZNI5dvBSfZoC) . Elsevier Science & Technology Books. ISBN 1-59749-005-9.
2. Энли, Крис; Козиол, Джек (2007). *Справочник шеллкодера: обнаружение и эксплуатация уязвимостей безопасности* (2-е изд.). Индианаполис, Индиана, UA: Wiley. ISBN 978-0-470-19882-7. OCLC 173682537 (https://search.worldcat.org/oclc/173682537) .
3. Фостер, Джеймс С. (2005). *Атаки переполнения буфера: обнаружение, эксплуатация, предотвращение* . Рокленд, Массачусетс, США: Syngress. ISBN 1-59749-022-9. OCLC 57566682 (https://search.worldcat.org/oclc/57566682) .
4. "Tiny Execve sh - язык ассемблера - Linux/x86" (https://github.com/geyslan/SLAE/blob/master/4th.assignment/tiny_execve_sh.asm) . *GitHub* . Получено 2021-02-01 . (https://github.com/geyslan/SLAE/blob/master/4th.assignment/tiny_execve_sh.asm)
5. BHA (2013-06-06). "Shellcode/Socket-reuse" (http://www.blackhatlibrary.net/Shellcode/Socket-reuse). Получено 2013-06-07 . (http://www.blackhatlibrary.net/Shellcode/Socket-reuse)
6. SkyLined (2010-01-11). "Download and LoadLibrary shellcode released" (https://web.archive.org/web/20100123014637/http://skypher.com/index.php/2010/01/11/download-and-loadlibrary-shellcode-released/) . Архивировано из оригинала (http://skypher.com/index.php/2010/01/11/download-and-loadlibrary-shellcode-released/) 2010-01-23 . Получено 2010-01-19 . (https://web.archive.org/web/20100123014637/http://skypher.com/index.php/2010/01/11/download-and-loadlibrary-shellcode-released/) (http://skypher.com/index.php/2010/01/11/download-and-loadlibrary-shellcode-released/)
7. "Загрузка и загрузка шеллкода библиотеки для x86 Windows" (https://code.google.com/p/w32-dl-loadlib-shellcode/) . 2010-01-11 . Получено 2010-01-19 . (https://code.google.com/p/w32-dl-loadlib-shellcode/)
8. Skape (2004-03-09). "Безопасный поиск виртуального адресного пространства процесса" (http://www.hick.org/code/skape/papers/egghunt-shellcode.pdf)(PDF) . nologin . Получено 2009-03-19 . (http://www.hick.org/code/skape/papers/egghunt-shellcode.pdf)

9. SkyLined (2009-03-16). "w32 SEH omelet shellcode" (https://web.archive.org/web/20090323030636/http://skypher.com/wiki/index.php?title=Shellcode%2Fw32_SEH_omelet_shellcode) . Skypher.com. Архивировано из оригинала (http://skypher.com/wiki/index.php?title=Shellcode/w32_SEH_omelet_shellcode) 2009-03-23 . Получено 2009-03-19 . (https://web.archive.org/web/20090323030636/http://skypher.com/wiki/index.php?title=Shellcode%2Fw32_SEH_omelet_shellcode) (http://skypher.com/wiki/index.php?title=Shellcode/w32_SEH_omelet_shellcode)
10. "Обнаружено большое количество неэкранированных шаблонов JavaScript" (https://web.archive.org/web/20150403203325/http://www.iss.net/security_center/reference/vuln/JavaScript_Large_Unescape.htm) . Архивировано из оригинала (http://www.iss.net/security_center/reference/vuln/JavaScript_Large_Unescape.htm) 2015-04-03. (https://web.archive.org/web/20150403203325/http://www.iss.net/security_center/reference/vuln/JavaScript_Large_Unescape.htm) (http://www.iss.net/security_center/reference/vuln/JavaScript_Large_Unescape.htm)
11. rix (2001-08-11)."Написание алфавитно-цифровых шелл-кодов ia32" (http://www.phrack.org/issues.html?issue=57&id=15#article).*Phrack*.**0x0b**(57). Phrack Inc. #0x0f из 0x12.Архивировано (https://web.archive.org/web/20220308045645/http://phrack.org/issues/57/15.html#article)из оригинала 2022-03-08. Получено 2022-05-26 . (http://www.phrack.org/issues.html?issue=57&id=15#article) (https://web.archive.org/web/20220308045645/http://phrack.org/issues/57/15.html#article)
12. obscou (2003-08-13)."Building IA32 'Unicode-Proof' Shellcodes" (http://www.phrack.org/issues.html?issue=61&id=11#article).*Phrack*.**11**(61). Phrack Inc. #0x0b из 0x0f.Архивировано (https://web.archive.org/web/20220526165740/http://phrack.org/issues/61/11.html#article)из оригинала 2022-05-26. Получено 2008-02-29 . (http://www.phrack.org/issues.html?issue=61&id=11#article) (https://web.archive.org/web/20220526165740/http://phrack.org/issues/61/11.html#article)
13. Мейсон, Джошуа; Смолл, Сэм; Монроуз, Фабиан; Макманус, Грег (ноябрь 2009 г.). *Английский шеллкод* (http://www.cs.jhu.edu/~sam/ccs243-mason.pdf) (PDF). Труды 16-й конференции ACM по компьютерной и коммуникационной безопасности. Нью-Йорк, штат Нью-Йорк, США. стр. 524–533.Архивировано (https://web.archive.org/web/20220526164459/https://www.cs.jhu.edu/~sam/ccs243-mason.pdf)(PDF)из оригинала 26.05.2022. Получено 10.01.2010 . (http://www.cs.jhu.edu/~sam/ccs243-mason.pdf) (https://web.archive.org/web/20220526164459/https://www.cs.jhu.edu/~sam/ccs243-mason.pdf) (10 страниц)
14. "Объяснение многоархитектурного (x86) и 64-битного алфавитно-цифрового шеллкода" (https://web.archive.org/web/20120621124443/http://www.blackhatlibrary.net/Alphanumeric_shellcode) . Blackhat Academy. Архивировано из оригинала (http://www.blackhatlibrary.net/Alphanumeric_shellcode) 2012-06-21. (https://web.archive.org/web/20120621124443/http://www.blackhatlibrary.net/Alphanumeric_shellcode) (http://www.blackhatlibrary.net/Alphanumeric_shellcode)
15. eugene (2001-08-11). "Architecture Spanning Shellcode" (http://www.phrack.org/issues.html?issue=57&id=14#article) . *Phrack* . Phrack Inc. #0x0e из 0x12. Архивировано (https://web.archive.org/web/20211109173710/http://phrack.org/issues/57/14.html#article) из оригинала 2021-11-09 . Получено 2008-02-29 . (http://www.phrack.org/issues.html?issue=57&id=14#article) (https://web.archive.org/web/20211109173710/http://phrack.org/issues/57/14.html#article)
16. nemo (2005-11-13). "OSX - Multi arch shellcode" (https://seclists.org/fulldisclosure/2005/Nov/387) . *Полное раскрытие информации* . Архивировано (https://web.archive.org/web/20220526191616/https://seclists.org/fulldisclosure/2005/Nov/387) из оригинала 2022-05-26 . Получено 2022-05-26 . (https://seclists.org/fulldisclosure/2005/Nov/387) (https://web.archive.org/web/20220526191616/https://seclists.org/fulldisclosure/2005/Nov/387)

17. Ча, Санг Кил; Пак, Брайан; Брамли, Дэвид ; Липтон, Ричард Джей (2010-10-08) [2010-10-04]. *Платформонезависимые программы* (https://softsec.kaist.ac.kr/~sangkilc/papers/cha-ccs10.pdf) (PDF) . Труды 17-й конференции ACM по компьютерной и коммуникационной безопасности (CCS'10). Чикаго, Иллинойс, США: Университет Карнеги-Меллона , Питтсбург, Пенсильвания, США / Технологический институт Джорджии , Атланта, Джорджия, США. стр. 547–558 . doi : 10.1145/1866307.1866369 (https://doi.org/10.1145%2F1866307.1866369) . ISBN (https://softsec.kaist.ac.kr/~sangkilc/papers/cha-ccs10.pdf) (https://doi.org/10.1145%2F1866307.1866369) 978-1-4503-0244-9. Архивировано (https://web.archive.org/web/20220526153147/https://softsec.kaist.ac.kr/~sangkilc/papers/cha-ccs10.pdf) (PDF) из оригинала 2022-05-26 . Получено 2022-05-26 . [1] (https://web.archive.org/web/20220526182333/http://users.ece.cmu.edu/~sangkilc/papers/ccs10-cha.pdf) (12 страниц) (См. также: [2] (https://security.ece.cmu.edu/pip/index.html))

Внешние ссылки

- Shell-Storm (http://www.shell-storm.org/shellcode/) База данных шелл-кодов. Мультиплатформенность.
- Введение в переполнение буфера и шеллкод (http://www.phrack.org/issues.html?issue=49&id=14#article)
- Основы шеллкодинга (http://www.infosecwriters.com/text_resources/pdf/basics_of_shellcoding.pdf) (PDF) Обзор шеллкодинга x86 от Анджело Росиелло (http://www.rosiello.org/)
- Введение в разработку шеллкода (https://web.archive.org/web/20120109070051/http://goodfellas.shellcode.com.ar/docz/bof/Writing_shellcode.html)
- Содержит образцы шелл-кода x86 и не-x86, а также онлайн-интерфейс для автоматической генерации и кодирования шелл-кода из проекта Metasploit. (https://web.archive.org/web/20080302111910/http://www.metasploit.com/shellcode/)
- архив шелл-кода, отсортированный по операционной системе (https://web.archive.org/web/20060619025456/http://www.linux-secure.com/endymion/shellcodes/) .
- Учебное пособие по проектированию шелл-кода для Microsoft Windows и Linux: от базового до продвинутого (https://web.archive.org/web/20061112203748/http://www.milw0rm.com/papers/11) .
- Учебник по шелл-коду для Windows и Linux, содержащий пошаговые примеры (http://www.vividmachines.com/shellcode/shellcode.html) .
- Разработка шелл-кода демистифицирована (https://web.archive.org/web/20210322094322/http://www.enderunix.org/docs/en/sc-en.txt)
- ALPHA3 (https://code.google.com/p/alpha3/) Кодер шелл-кода, который может преобразовать любой шелл-код в Unicode и ASCII, в верхнем и смешанном регистре, буквенно-цифровой шелл-код.
- Написание небольшого шелл-кода, автор Дэвид Статтард. (https://web.archive.org/web/20061115040739/http://www.ngssoftware.com/research/papers/WritingSmallShellcode.pdf) Технический документ, объясняющий, как сделать шелл-код максимально компактным, оптимизировав как дизайн, так и реализацию.
- 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) Archived (https://web.archive.org/web/20150403114315/http://skypher.com/wiki/index.php?title=Www.edup.tudelft.nl%2F~bjwever%2Fwhitepaper_shellcode.html.php) 2015-04-03 at the Wayback Machine 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.
- 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.today/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)

Retrieved from "https://en.wikipedia.org/w/index.php?title=Shellcode&oldid=1275596651"