GrdCRC

Функция(метод) **GrdCRC** позволяет подсчитать 32-битный CRC участка памяти. Это удобно для экспресс-анализа и контроля целостности кода или данных.

C

```
DWORD GRD_API GrdCRC (
  void *pData,
  DWORD dwLng,
  DWORD dwPrevCRC
);
```

pData	адрес участка памяти для подсчета CRC
dwLng	длина (в байтах) участка памяти
dwPrevCRC	CRC предыдущего участка памяти, при последовательном вычислении CRC для нескольких буферов

32-битный (4 байта) CRC участка памяти.
Функция **GrdCRC** позволяет подсчитать CRC (циклический избыточный код) участка памяти или буфера. CRC обычно используется как контрольная сумма каких-либо данных (изменение хотя бы одного бита данных, с приемлемой в большинстве случаев вероятностью, приведет к существенному изменению значения CRC). Параметр *pData* задает адрес участка памяти, а *dwLng* - длину этого участка в байтах. Если необходимо подсчитать общий CRC нескольких участков, в параметр *dwPrevCRC* при подсчете CRC каждого следующего участка нужно занести CRC, подсчитанный на предыдущем этапе. Иначе параметр *dwPrevCRC* должен содержать -1 (константа Grd_StartCRC). Функция GrdCRC не возвращает никакого кода ошибки.

Данная функция удобна, например, для контроля результатов расшифровки блоков данных без непосредственного сравнения с эталоном. Если эталон дешифрованых данных присутствует в коде приложения, то хакер может найти его и легко восстановить оригинальный вид данных. Однако надо понимать, что и по значению CRC небольших блоков данных, за счет содержащейся в нем избыточности, возможно его реверсирование (т.е. восстановление оригинальных данных по их CRC). Поэтому в критичных для этого случаях лучше использовать функцию GrdHash / GrdHashEx для подсчета 256-битного SHA-хэша, который лучше защищен от подобных атак.

C#

```
public static unsafe uint GrdCRC(byte[] data)
public static unsafe uint GrdCRC(byte[] data, uint prevCRC)
public static unsafe uint GrdCRC(short[] data)
public static unsafe uint GrdCRC(short[] data, uint prevCRC)
public static unsafe uint GrdCRC(ushort[] data)
public static unsafe uint GrdCRC(ushort[] data, uint prevCRC)
public static unsafe uint GrdCRC(int[] data)
public static unsafe uint GrdCRC(int[] data, uint prevCRC)
public static unsafe uint GrdCRC(uint[] data)
public static unsafe uint GrdCRC(uint[] data, uint prevCRC)
public static unsafe uint GrdCRC(long[] data)
public static unsafe uint GrdCRC(long[] data, uint prevCRC)
public static unsafe uint GrdCRC(ulong[] data)
public static unsafe uint GrdCRC(ulong[] data, uint prevCRC)
```

data [in]

Типы: byte [ ], short [ ], ushort [ ], int [ ], uint [ ], long [ ], ulong [ ]

Адрес участка памяти для вычисления CRC.

prevCRC [in]

Тип: uint

CRC для предыдущего участка памяти, при последовательном вычислении CRC для нескольких буферов.
32-битный (4 байта) CRC участка памяти.

Метод **GrdCRC** позволяет подсчитать CRC (циклический избыточный код) участка памяти или буфера. CRC обычно используется как контрольная сумма каких-либо данных (изменение хотя бы одного бита данных, с приемлемой в большинстве случаев вероятностью, приведет к существенному изменению значения CRC). Параметр *data* задает адрес участка памяти. Если необходимо подсчитать общий CRC нескольких участков, в параметр *prevCRC* при подсчете CRC каждого следующего участка нужно занести CRC, подсчитанный на предыдущем этапе. Иначе параметр *prevCRC* должен содержать -1 (константа Grd_StartCRC). Метод GrdCRC не возвращает никакого кода ошибки.

Данный метод удобен, например, для контроля результатов расшифровки блоков данных без непосредственного сравнения с эталоном. Если эталон дешифрованых данных присутствует в коде приложения, то хакер может найти его и легко восстановить оригинальный вид данных. Однако надо понимать, что и по значению CRC небольших блоков данных, за счет содержащейся в нем избыточности, возможно его реверсирование (т.е. восстановление оригинальных данных по их CRC). Поэтому в критичных для этого случаях лучше использовать метод GrdHash / GrdHashEx для подсчета 256-битного SHA-хэша, который лучше защищен от подобных атак.

Java

```
public static int GrdCRC(byte[] data)
public static int GrdCRC(byte[] data, int prevCRC)
public static int GrdCRC(short[] data)
public static int GrdCRC(short[] data, int prevCRC)
public static int GrdCRC(int[] data)
public static int GrdCRC(int[] data, int prevCRC)
public static int GrdCRC(long[] data)
public static int GrdCRC(long[] data, int prevCRC)
```

data [in]

Типы: byte [ ], short [ ], int [ ], long [ ]

Адрес участка памяти для вычисления CRC.

prevCRC [in]

Тип: int

CRC для предыдущего участка памяти, при последовательном вычислении CRC для нескольких буферов.
32-битный (4 байта) CRC участка памяти.
Метод GrdCRC позволяет подсчитать CRC (циклический избыточный код) участка памяти или буфера. CRC обычно используется как контрольная сумма каких-либо данных (изменение хотя бы одного бита данных, с приемлемой в большинстве случаев вероятностью, приведет к существенному изменению значения CRC). Параметр *data* задает адрес участка памяти. Если необходимо подсчитать общий CRC нескольких участков, в параметр *prevCRC* при подсчете CRC каждого следующего участка нужно занести CRC, подсчитанный на предыдущем этапе. Иначе параметр *prevCRC* должен содержать -1 (константа Grd_StartCRC). Метод GrdCRC не возвращает никакого кода ошибки.

Данный метод удобен, например, для контроля результатов расшифровки блоков данных без непосредственного сравнения с эталоном. Если эталон дешифрованых данных присутствует в коде приложения, то хакер может найти его и легко восстановить оригинальный вид данных. Однако надо понимать, что и по значению CRC небольших блоков данных, за счет содержащейся в нем избыточности, возможно его реверсирование (т.е. восстановление оригинальных данных по их CRC). Поэтому в критичных для этого случаях лучше использовать метод GrdHash / GrdHashEx для подсчета 256-битного SHA-хэша, который лучше защищен от подобных атак.