Reversed Buffer Overflow

Cross Stack Attacks

Kris Kaspersky
Endeavor Security, Inc.

Who am I?

- journalist, reversing as a hobby;
- currently working for:
  - **XAKEP** magazine (www.xakep.ru)
  - **Endeavor Security, Inc** (www.endeavorsecurity.com)
- telephone:
  - +7 (918) 348-92-93 (mobile)
  - +7 (86-140) 5-82-69 (office)
  - +7 (86-140) 5-51-18 (home)
- email:
  - poldhiir#gmail.com

History

- **for years hackers overflow stack buffers in the same direction:**
  - from lower to higher addresses;
  - stack grows from top to bottom;
- **compilers and OS developers have provided a lot of protections:**
  - stack guard, pro-police, not executable stack, address space randomization (ASLR), etc;
- **it's too hard to perform buffer overflow today!**
  - the new approach has been found:reversed buffer overflow;
  - we're going to overflow the stack from top to bottom!

Safe and unsafe memory allocation

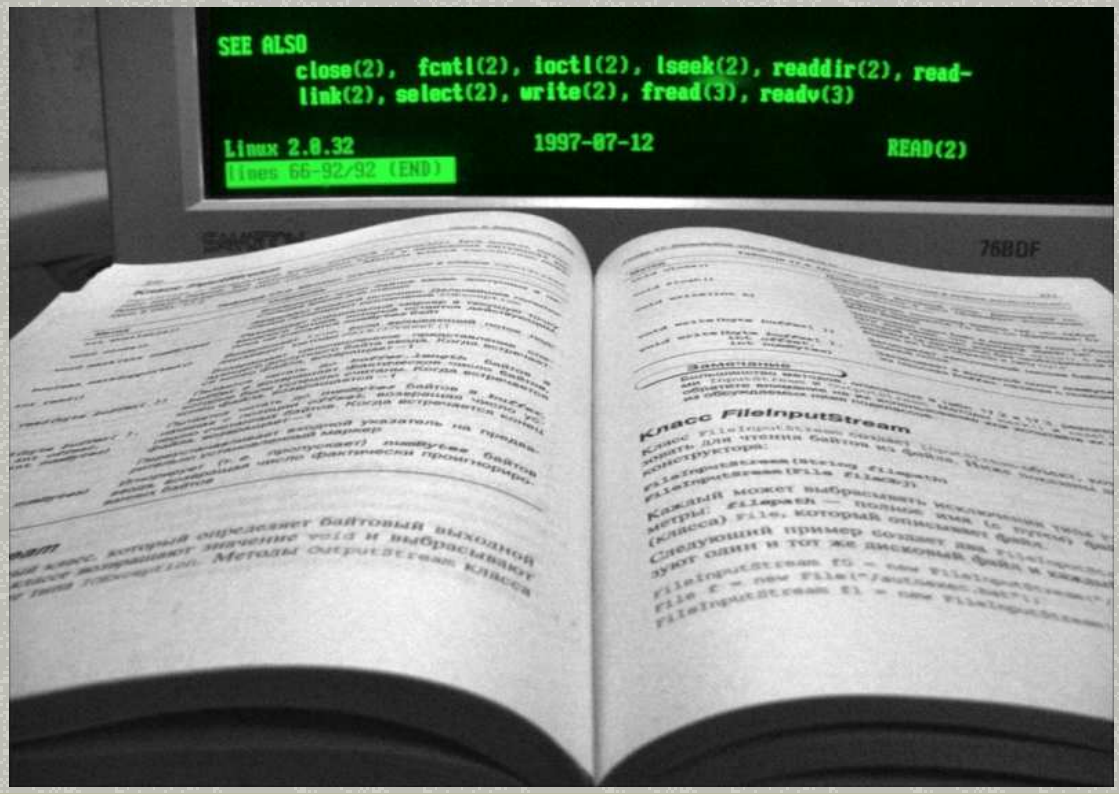
- malloc without checking the return address is a great sin;
- do you think the following code is safe?
  - ```
    foo(int a, int b)
    {
        return a + b;
    }
    ```
  - ```
    bar()
    {
        return foo(6, 9);
    }
    ```

Safe and unsafe memory allocation

- **this is unsafe and unreliable code!**
  - **there is no easy way to determine how much stack memory left;**
  - **during page file increasing:**
    - **malloc() returns zero;**
    - **attempts to allocate a new stack page causes STACK_OVERFLOW exception;**
  - **recursion is a bad idea:**
    - **especially if there is no depth control;**
    - **almost ANY recursive app is vulnerable;**
  - **using local arrays is a bad idea:**
    - **stack is too small, use heap or PE file image;**
    - **most of apps using big local arrays are vulnerable;**

Safe and unsafe memory allocation

- just a few programmers care to handle the stack overflow exceptions!
- almost nobody does it right!
- why? coz nobody really knows how stack works!

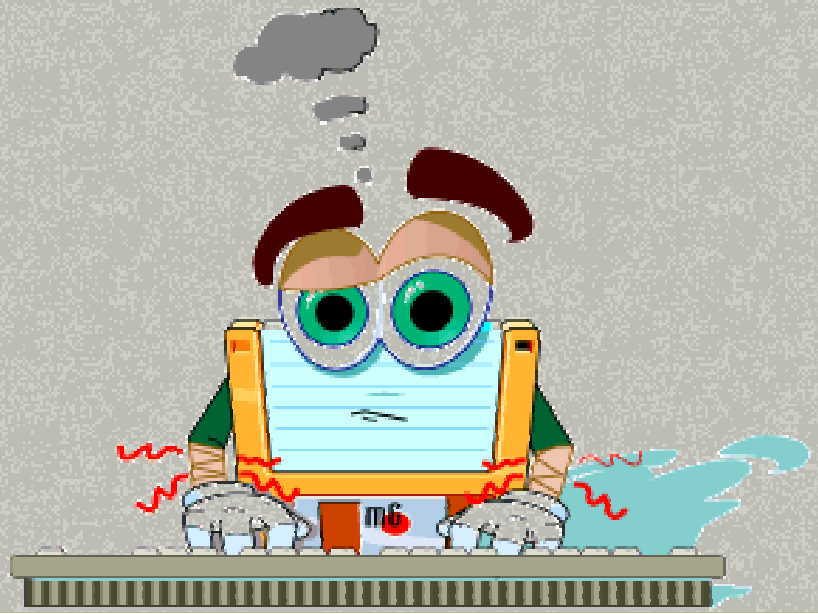


Stack overflow exception internals

- NT doesn't commit stack pages in advance;
  - MSDN says:
    - there is a guard page on the top of stack;
    - accessing to the guard page causes the exception;
      - OS catches the exception;
      - automatically commits the page;
      - moves the guard up;
  - MSDN doesn't clearly say:
    - there is two types of guard pages:
    - red and yellow, speaking in DEC's terms;
    - a yellow page is a normal guard page;
    - when a yellow page reaches the top of the stack it turns to red;
    - accessing to the red guard page:
      - removes PAGE_GUARD flag;
      - causes STACK_OVERFLOW exception;
      - OS passes control to a custom SEH handler (if there is one);

Stack overflow exception internals

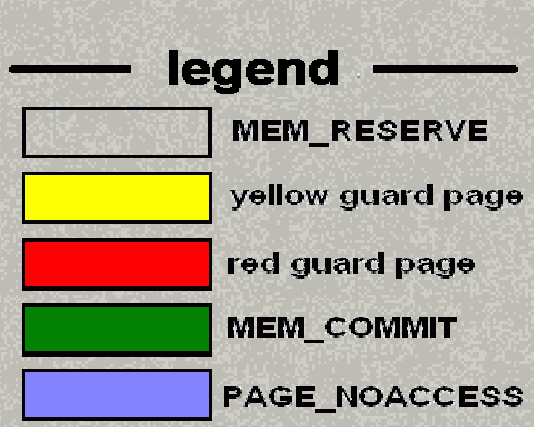
- the point is:
  - STACK_OVERFLOW exception is generated only once!
  - OS doesn't restore PAGE_GUARD flag automatically when the red page has been reached;
  - a programmer is supposed to do it manually;
  - who does it? nobody!



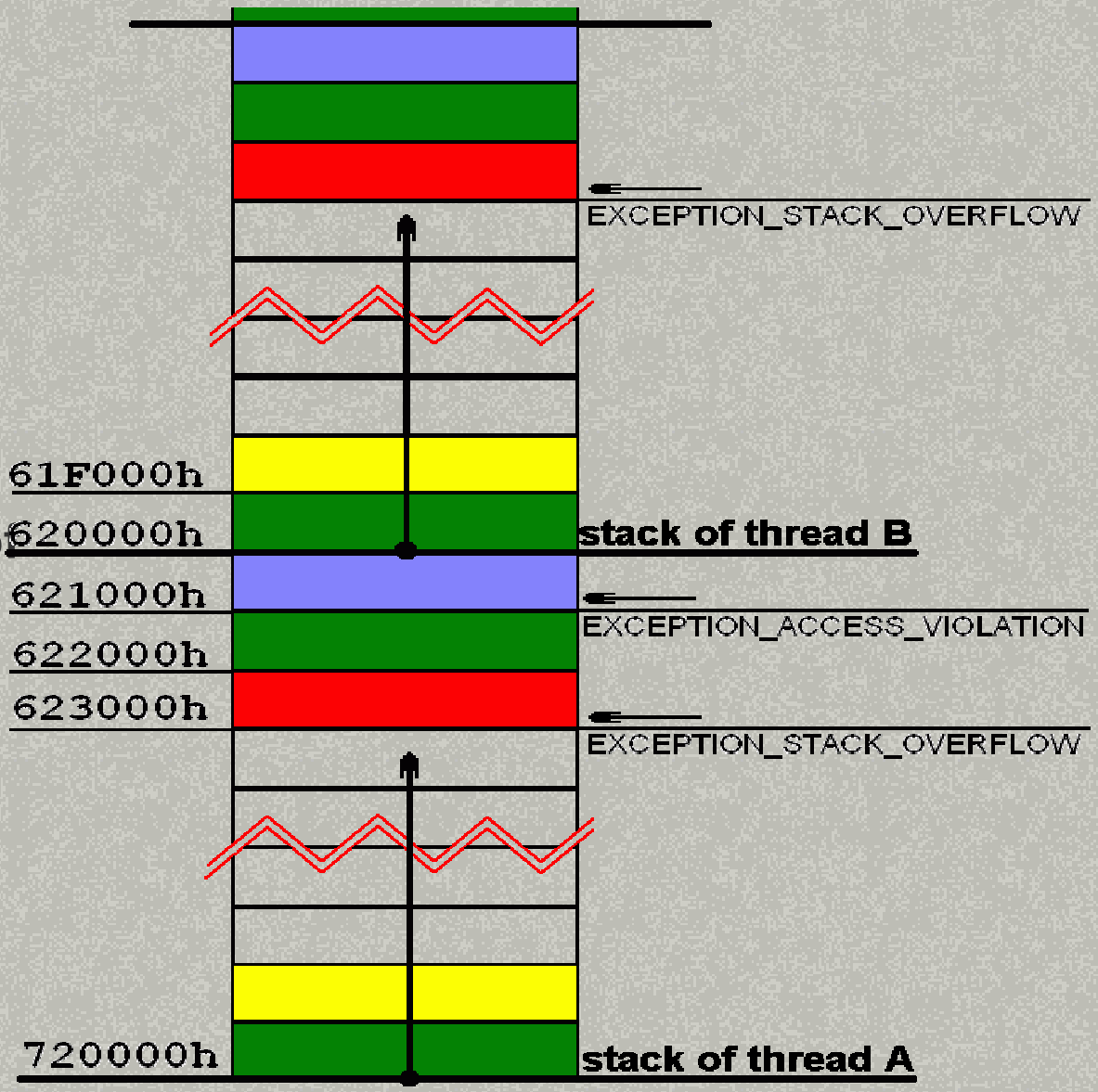
Stack overflow exception internals

- typical stack overflow exception handler:
  - just a few programmers care to catch stack overflow exception;
  - most of those who handle it don't restore PAGE_GUARD flag;
- what happens if the stack will be overflowed again?
  - stack overflow exception will NOT be generated;
  - access violation exception will be generated;
  - why?! there is a page with PAGE_NOACCESS flag on the top of the stack;
- moral:
  - stack is much more complicated than it seems to be;

How the stack is organized?



- STACK_OVERFLOW:
  - 3 pages left at the top of the stack!
  - 2 pages are allocated and can be used by SEH-handler;
  - the last page is marked as PAGE_NOACCESS to prevent out-of-stack memory access;



The back of beyond

- what is above the stack?
  - unallocated memory;
  - heap memory block;
  - the bottom of the another thread stack;
- demonstration of "stack-allocate-strategy";
  - source code;
  - the result;
  - comparing different OS;
- is it possible to overwrite heap or another stack?
  - yes!!!
  - thousands vulnerable applications!!!

Attack scenarios

- **DoS or application termination:**
  - all applications share the same system memory and page file;
  - one app consumes memory - another app crashes;
  - it's easy to force browser (IE, FireFox or Opera) to consume a lot of memory;
  - forcing a victim to allocate stack memory and increase page files to raise STACK_OVERFLOW exception;
  - the victim crashes;
  - it might be personal firewall, AV or something else;
  - yes! it's possible to terminate firewall/AV remotely via browser;

Attack scenarios

- it's very hard to develop safe-n-secure code:
  - STACK_OVERFLOW exception is raised;
  - stack is NOT overflowed, but no memory for SEH;
  - OS kernel is unable to allocate a stack page for SEH;
  - OS kernel terminates the application;
  - the application can't prevent this;
- survival guide:
  - allocate memory in advance, before executing critical code;
  - set min. page file size equal to max;

Attack scenarios

- a few optimization tricks:
  - standard stack allocator is too slow:
    - allocates only one page at a time;
    - an exception costs a lot of CPU ticks;
  - it's possible to replace the system stack allocator:
    - to allocate more than a page per time;
    - o reserve a couple pages for SEH;
    - to restore GUARD_PAGE flag automatically;
    - to generate only STACK_OVERFLOW exception;
    - NOT unexpected ACCESS_VIOLATION exception;

Attack scenarios

- another typical bug occurs during program installation:
  - check free disk space;
  - begin the installation process if it's enough;
  - oops! NT & Linux/BSD is multi-tasking OS;
  - any app can consume disk space during the installation;
  - the installer has to reserve requested disk space before installation;
  - the same situation with system memory;
    - allocate memory for critical needs at startup;
    - this guarantees the app will be able to save all data when there is no free memory left;

Accessing out-of-stack memory

- exploiting optimized compilers/assembly in-lines bugs:
  - normally, stack prologue code reads every page of the stack frame to avoid "jump-over-guard" situation;
  - if accessed pages were allocated by previous functions, this procedure isn't necessary;
  - MOV [ESP-JMP_OVER], XX (where JMP_OVER >= 3000h) allows overwriting memory outside of the current thread stack by jumping over the stack and the guard page;
    - ACCESS_VIOLATION exception only if the stack frame crosses the guard page;
    - STACK_OVERFLOW exception is raised when 3 stack pages is left;
  - it's hard to reproduce the "jump-over-guard" bug;
  - a buggy app has all chances to work well for years;
  - Sun Java as an example of the buggy app;

Questions?