Creating User-Friendly Exploits

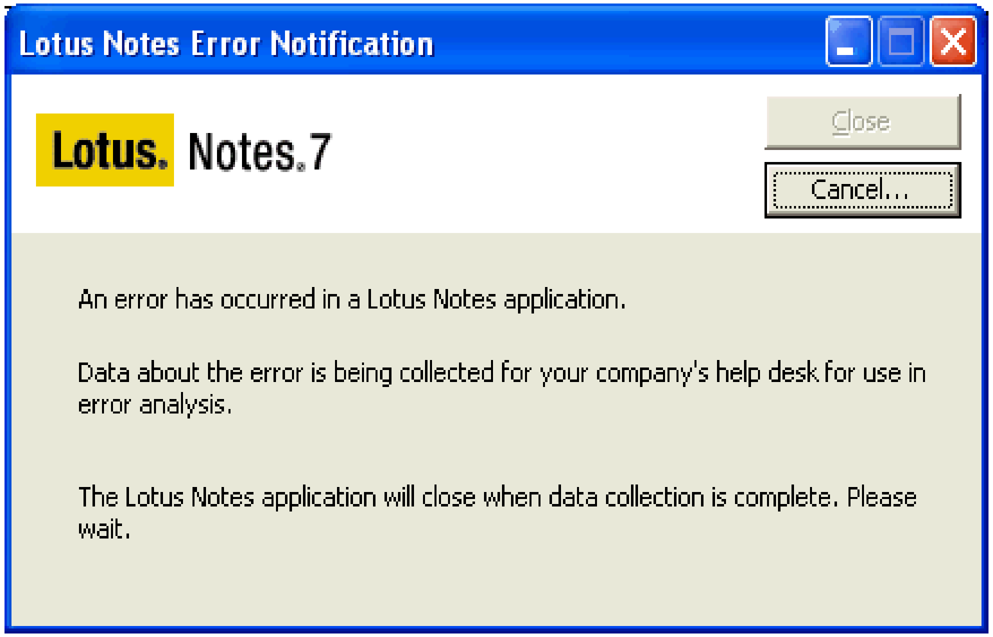
Skylar Rampersaud
skylar@immunityinc.com

IMMUNITY Security Research

What is a User-Friendly Exploit?

- An exploit that causes no distress to the user of the exploited program
  - i.e., signs or symptoms

Why Should You Care?

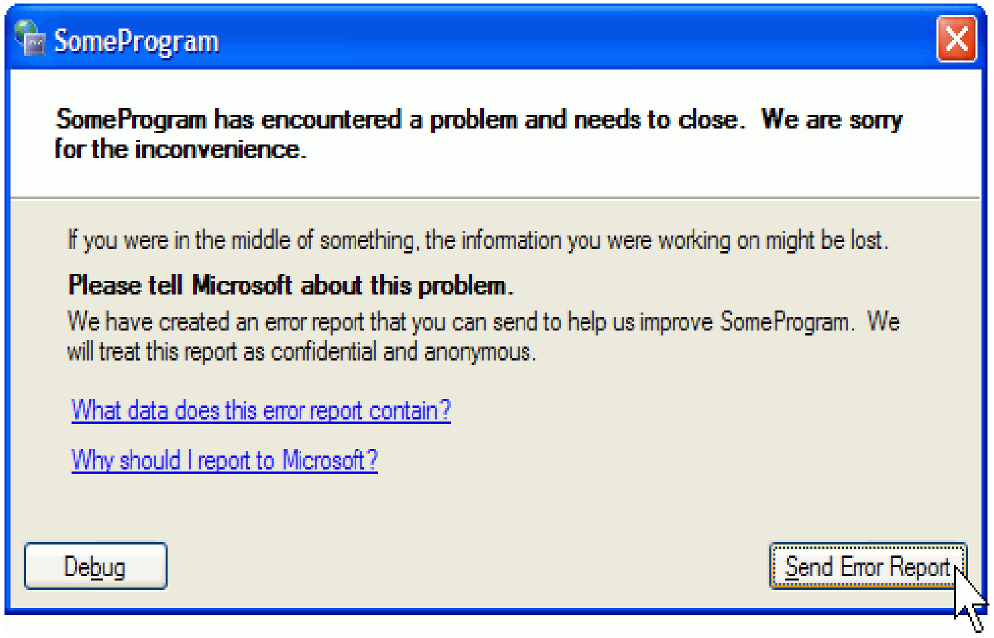


Lotus Notes Error Notification

An error has occurred in a Lotus Notes application.

Data about the error is being collected for your company's help desk for use in error analysis.

The Lotus Notes application will close when data collection is complete. Please wait.



SomeProgram

SomeProgram has encountered a problem and needs to close. We are sorry for the inconvenience.

If you were in the middle of something, the information you were working on might be lost.

Please tell Microsoft about this problem.
We have created an error report that you can send to help us improve SomeProgram. We will treat this report as confidential and anonymous.

What data does this error report contain?

Why should I report to Microsoft?

Problem Statement

- Given an exploitable bug in a Windows application
- How can you execute arbitrary code such that:
  - The application continues to run
  - The application appears to run normally?

Some General Things

- Do as little as possible before returning control to the program
- Prevent the exploit from running again
- Logging
- Visual Cues

The Big Deal: Recovery

- Recovery is the shellcode we write to make the exploited process continue as if nothing unusual happened

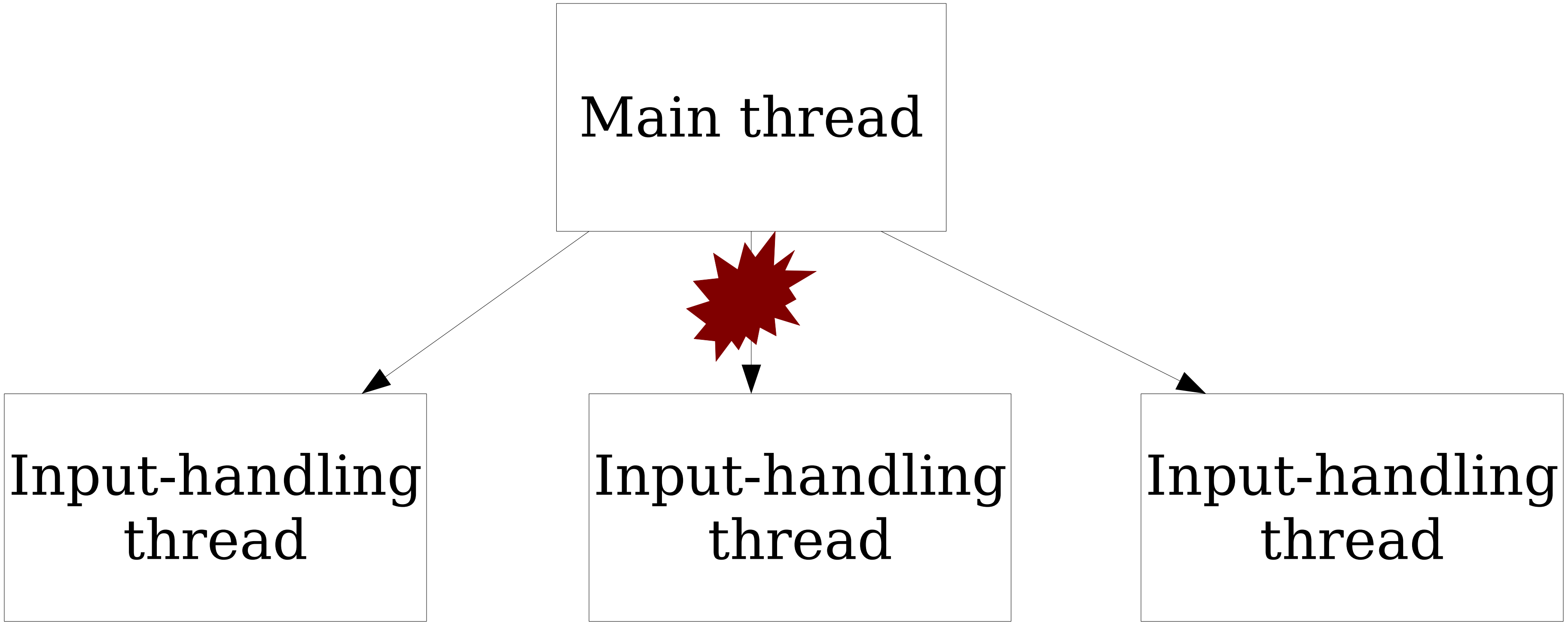
- It's more than calling ExitThread()

Single-Threaded Processes

Main thread

- Dead thread == Dead process

Multithreaded Process



Multi-threaded, take 2

- Multiple threads, but each one has specific functions
- Killing a thread won't kill the process
  - But it will be severely disabled

KNOWING YOU'RE SECURE

Restarting/Replacing the Thread

- May not be feasible
  - Thread creates windows
  - Other threads holding handle to current thread

The Plan

- Identify a place to return control
- Release shared resources
- Find/fix data structures

Observe and Emulate

- Get cozy with your debugger

IMMUNITY
DEBUGGER

Example: Complete Stack Overwrite

- Upon gaining execution, the thread's stack is in bad shape
- Having the thread continue execution seems impossible

Challenges

- A (possibly large) number of functions have not completed due to exploitation
- What were those functions supposed to do?
- What resources are held that haven't been released?
- Where can we return control to the program?

Automating Cleanup

- We're looking at a labour-intensive, manual process
- Some elements can be automated

Immunity Debugger

- Since we'll be spending a lot of time with the debugger, an extensible framework is ideal
- Immunity Debugger allows you to create custom scripts
  - and is freely available
- It uses the Python scripting language since it's flexible and easy to use

Finding the Message Pump

- There will be a place where the thread loops, waiting for indications that it has some work to do
- May be its own function
- May be in the thread's initial function

Automating Message Pump Finding

- We can do this manually
  - Read the code in the thread's call stack
  - Test hypotheses by setting breakpoints
- We can write an ID script
  - Hook calls to PeekMessage or other communication functions

Finding C++ objects

- What objects does the message pump use?
- Where are they located (heap, stack, .data?)
- How does the message pump reference them?

KNOWING YOU'RE SECURE

Make It or Fake It

- A C++ object pointer was on the stack
  - Can't locate it
- Allocate some memory
  - Use a pointer to the actual function
  - Or make your own

OS vs. Application Synchronization

- Windows provides a variety of objects:
  - Mutexes, Semaphores
  - Processes, Threads
  - Input, Events, Notifications
  - Waitable timers
- Applications can implement their own synchronization mechanisms

Recognizing Synchronization

- The process doesn't crash
  - But it doesn't exactly work, either
- Variable checking at the beginning of functions
  - Especially “end if non-zero” checks

Finding Synchronization Issues

- We can do this manually
  - Read code that is executed by all other threads
- We can write an ID script
  - Keep track of any objects being waited on
- Demo later!

Versioning

- Techniques described makes exploit more “brittle”
  - Easier to break if something changes
- Remember, at this point we have code execution.
  - Easy to check for exact versions of DLLs, etc.

Demo!

Conclusions

- Code execution is not the end of the story!
- ExitThread() and ExitProcess() aren't your only options
- Cleanup requires in-depth process knowledge
- Immunity Debugger offers tools to improve your shellcode-writing experience

KNOWING YOU'RE SECURE

Thank you for your time!

Questions?

Get Immunity Debugger at:

http://www.immunityinc.com/products-immdbg.shtml

IMMUNITY

Security Research Team