SECURITY DRIVEN .NET

.NET/ASP.NET 4.5

STAN DRAPKIN. OCTOBER 2013.

SECURITY DRIVEN .NET

by Stan Drapkin

Last updated: 29 October 2013.

“The world needs security engineers even more than it needs cryptographers. We're great at mathematically secure cryptography, and terrible at using those tools to engineer secure systems.”

Bruce Schneier

TABLE OF CONTENTS

PREFACE

Who Is This Book For?

This book is intended for architects, developers, and other information technology professionals who design and build security components/layers of .NET solutions.

Why Is This Book Relevant Today?

Three main problems that plague developers writing security-focused .NET code today are:

- Many security developers do not know most of the material in this book, regardless of seniority or job title.
- What many security developers believe to be the "right-thing-to-do" is often wrong and/or insecure.
- Even the most knowledgeable and aware security developers are often unable to produce quality .NET implementations that are sane, simple, reusable, and secure.

This book addresses many widespread security knowledge gaps and provides practical guidance & code samples.

What Makes This Book Different From Other ".NET Security" Books?

- Original & relevant content. We have not seen the topics we cover addressed properly in other books.
- Absence of lengthy introductions. There are many excellent books and other learning materials dedicated to introductory and foundational topics. We get to the point right away and have neither time nor patience for handholding – we've got work to do and so do you.
- The only math required is arithmetic. We promise.
- Our advice is based on real-world experience implementing .NET security. Sometimes we "bend the spoon", other times we bend ourselves, but we pick our battles and compromises carefully.
- DRM-free PDF format with content-focused layout that does not waste pages. "More content in fewer pages" was one of our goals for this book. There is no index – we assume you know how to search within PDF.
- Simple license that is not even a "license". We do not "license" this book – we "sell" it. If you have purchased this book, it is "yours" – you can back it up, resell every copy you have purchased, etc. The only restriction is "do not violate our copyright".
- This book is self-published. We sell it for a small fraction of typical "publisher" price to help you afford it.

Source Code Samples

Source code for this book can be downloaded from http://SecurityDriven.NET.

CRYPTOGRAPHIC PRIMITIVES

.NET framework security foundation is made out of various cryptographic building blocks, which, in turn, rely on various cryptographic primitives. These primitives are supposed to be well-defined and well-understood cryptographic algorithms and services that follow a "Single Responsibility Principle" – i.e. they should do one thing only, and do it well. Cryptographic primitives are typically combined together to form more complex cryptographic building blocks, which can be further combined to form security protocols. You will see later that not all primitives provided by Microsoft are well-defined or well-understood.

The 1st rule of sane security architecture is "never design your own cryptographic primitives or protocols". The 2nd rule of sane security architecture is "never implement cryptographic primitives or protocols – even when you follow an established design", since your implementation will be flawed. Implementation flaws are a much more common cause of practical vulnerabilities than actual cryptographic weaknesses. There are no exceptions to the 1st rule – it is absolute. There are some exceptions to the 2nd rule, however. One acceptable reason to break the 2nd rule (outside of learning purposes) is to deal with situations when reliable implementations are either not available, or the available ones are sorely lacking.

You will be breaking the 2nd rule in upcoming chapters, but first you need the required understanding of what primitives are available in .NET out-of-the-box, and the strengths and deficiencies of their implementations. So let's get on with it…