CRYPTOGRAPHY: THE CASE OF CRYPTOGRAPHIC DESIGN IN ASP.NET

Prof. Ganga Gudi

Department of Computer Science, KLE’s S.Nijalingappa College, Bangalore, India

ABSTRACT

This paper discusses about how cryptography is misused in the security design of a large part of the Web. The web application framework developed by Microsoft powers 25% of all Internet web sites. The attackers can abuse multiple cryptographic design flaws to compromise web applications. We describe practical and highly efficient attacks that allow attackers to steal cryptographic secret keys and forge authentication tokens to access sensitive information. The attacks combine decryption oracles, unauthenticated encryptions, and the reuse of keys for different encryption purposes. Finally, we give some reasons why cryptography is often misused in web technologies, and recommend steps to avoid these mistakes.

Keywords: Cryptography, Application Security, Web security, Decryption oracle attack.

[1] INTRODUCTION

This paper shows cryptography is not limited to traditional software, but is highly pervasive in web applications as well. Since HTTP is a stateless protocol, web developers must either manage the user session state data on the server or push it to the client. For performance and scalability reasons, web developers tend to go with the latter method. They want to keep session information secret, so that they correctly turn to cryptography. However, implementing crypto is error-prone. We observe that unauthenticated encryption is often used to encrypt session state data such as HTTP cookies and view states. Unauthenticated encryption is dangerous, particularly when used in an authentication system. The ability to forge a cipher text that decrypts to a desired plaintext allows the attacker to impersonate other users easily. Web developers also tend to use the same keys for different encryption purposes. These Cryptographic errors together make the Web become a gold mine for chosen-cipher text attacks.

[2] AN OVERVIEW OF ASP.NET

In this section, we review some key concepts and terminology for ASP.NET. We then describe how the framework misuses cryptography when attempting to tamper-proof and encrypt sensitive information.

Key Concepts and Terminology
**Machine Key:** The machine key is a pair of global secret keys set in the web application configuration to be used for encryption and authentication. A key named validationKey is used to generate hashed message authentication codes (HMAC) to protect the integrity of authentication tickets and view states. A second key named decryptionKey is used to encrypt and decrypt authentication tickets and view states.

**View State:** An ASP.NET application is a collection of .NET pages, known officially as “web forms”. ASP.NET applications are hosted by a web server and are accessed using the stateless HTTP protocol. As such, if an application uses stateful interaction, it has to implement state management on its own. ASP.NET provides various functions for state management, and view state is one of them.

View state refers to the page-level state management mechanism utilized by the HTML pages emitted by ASP.NET applications to maintain the state of the web form controls and widgets.

**Forms Authentication Tickets:** Since ASP.NET aims to become a rapid web development framework, it provides built-in solutions for many common problems in web development. One of them is user account support. Providing user account support for any site involves the same set of steps: creating a data store, a login page and a register page; defining authentication and authorization mechanisms. Forms authentication uses an authentication ticket that is created when a user logs on to a site; this ticket is then used to track the user throughout the site. Each time a subsequent request is received after authentication, the FormsAuthenticationModule class retrieves the authentication ticket from the authentication cookie or the query string, decrypts it, computes the hash value, and verifies the HMAC value to ensure that the ticket has not been tampered with.

**Web Resources and Script Resources:** In the .NET frame-work, an assembly is a compiled code library used for deployment, versioning and security. An assembly consists of one or more files. These files can be code modules, web resources (e.g., HTML, CSS, or images), or script resources (e.g., JavaScript). Web developers reference these static resources through a standard API.

The request format for both WebResource.axd and ScriptResource.axd is as follows: WebResource.axd?d=encrypted_id & t=timestamp

We observe two interesting things about the d parameter:

1) ASP.NET encrypts this parameter, but does not authenticate the cipher text.
2) Due to a feature in ScriptResource.axd, an attacker can download arbitrary files inside the document root of ASP.NET applications given a valid encrypted d parameter.

**Cryptographic Design Flaws in ASP.NET**

**Insecure Key Management:** There are three issues in how ASP.NET manages cryptographic keys. The first issue is the reuse of keys for different purposes. In the last section, we showed that the framework uses cryptography to authenticate and encrypt view states, forms authentication tickets, web resources and script resources. These are pieces of information with different levels of importance. Forms authentication tickets and view states are critical to the security of ASP.NET, but web resources and script resources identifiers do not include very sensitive information. ASP.NET, however, encrypts all of them with the same cryptographic keys. The second issue is insecure key storage. By default, plaintext cryptographic keys are stored in a file named web.config in the document root of ASP.NET applications. In other words, all it takes to steal these keys in any ASP.NET application is one file disclosure.

**Improper Use of Cryptographic Primitives:** There are two issues in the way ASP.NET uses cryptography. First, the cryptographic API in ASP.NET does not use authenticated encryption by default. Secondly, the framework uses the MAC-then-Encrypt mode for authenticated encryption. As previous work has demonstrated, this mode is vulnerable to chosen-cipher text attacks.
[3] DECRYPTION ORACLE ATTACKS

In this section, we discuss decryption oracle attacks and the CBC-R technique. It is important to stress that the padding oracle is just one kind of decryption oracle and we have found decryption oracles that are easier and faster to exploit in ASP.NET.

The Padding Oracle Attack

The padding oracle attack requires an oracle that, on receipt of a cipher text, decrypts it and replies to the sender whether the padding is valid or invalid. The attack works under the assumption that the attackers can intercept padded messages encrypted in CBC mode and have access to the aforementioned padding oracle. The result is that attackers can recover the plaintext corresponding to any block of cipher text using an average of 128 b oracle calls, where b is the number of bytes in a block.

Turning Decryption Oracles into Encryption Oracles

In this section, we review CBC-R, a technique to turn a decryption oracle into an encryption oracle. When a system assumes that a meaningful message obtained from the decryption of some cipher text implies a trusted origin of it, the CBC-R technique allows attackers to create arbitrary cipher texts to abuse the system.

CBC-R: The CBC mode is defined as follows:

CBC Encryption:
\[ C_1 = CIPH_K(P_1 \oplus IV); \]
\[ C_i = CIPH_K(P_i \oplus C_{i-1}) \text{ for } i=2,\ldots,n. \]

CBC Decryption:
\[ P_1 = CIPH^{-1}(C_1) \oplus IV; \]
\[ P_i = CIPH^{-1}(C_i) \oplus C_{i-1} \text{ for } i=2,\ldots,n. \]

Algorithm: CBC-R.
1) Choose a plaintext message \( P \), pad the message, and divide it into \( n \) blocks of \( b \) bytes denoted by \( P_1, P_2, \ldots, P_n \).
2) Pick a few random bytes \( r_1, r_2, \ldots, r_b \), and set \( C_n = r_1 r_2 \ldots r_b \).
   For \( i = n \) down to 1
   \[ C_{i-1} = P_i \oplus O(C_i) \]
3) Set \( IV = C_0 \).
4) Output $IV$ and $C = C_1 \ldots C_n$

**CBC-R without Controlling:** We have shown that CBC-R allows the attacker to encrypt any message. But, if he cannot set, then the first plaintext block will be random and meaningless. If the victim expects the decrypted message to start with a standard header, and the attacker doesn’t control, then the victim will ignore the forged message constructed by CBC-R. This is what happens with the resource identifiers in ASP.NET, where the first two characters of the decrypted identifiers must be in the limited set of defined options.

[4] CONCLUSIONS

In this paper, we analyze and efficiently exploit several cryptographic flaws in ASP.NET, the widely-used web application framework developed by Microsoft. The most serious vulnerability we discovered is the use of unauthenticated encryption. This vulnerability is exacerbated by the reuse of keys to encrypt data with very different levels of importance. We present two practical and highly efficient attacks that allow an attacker to steal cryptographic secret keys, and impersonate any user account in ASP.NET applications. These attacks are performed by abusing components present in every application developed using the framework. The applications are even more exposed if they use the security features provided by ASP.NET, especially form based authentication.

Cryptography is difficult to implement correctly, and cryptographers often advise non-cryptographers not to develop their own cryptography. But if one looks more closely at the current situation, it is evident that web developers and users do not have much choice. ASP.NET developers still have to figure out on their own how to use cryptographic primitives correctly any time they want to build a secure cryptographic protocol. This is not a problem specific to ASP.NET.

REFERENCES