Descifrado de Ryuk mediante la predictibilidad en la generación de claves AES

Introducción

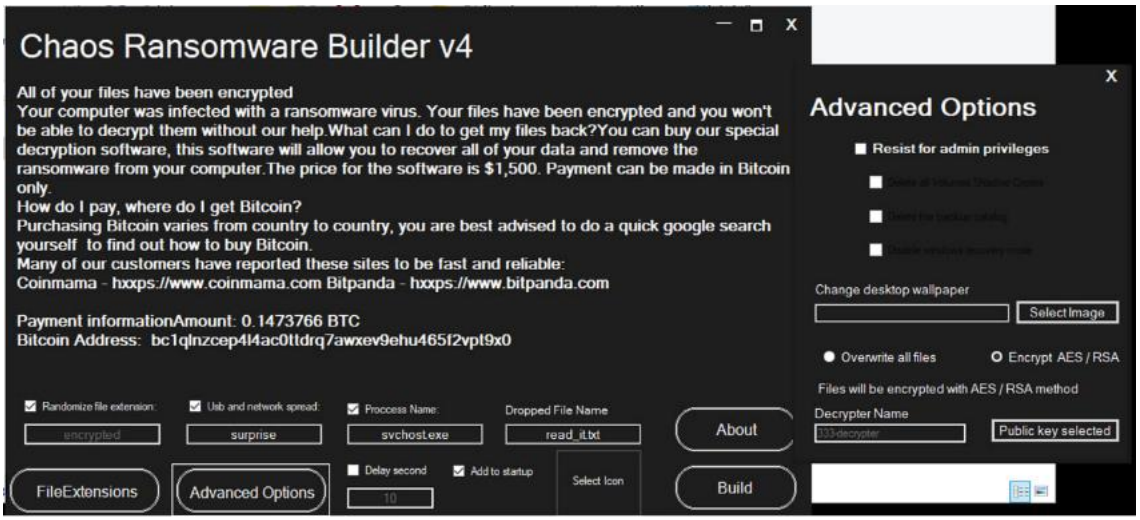
Ryuk es un ransomware dirigido que emplea un esquema criptográfico híbrido basado en **AES + RSA** [1]. Cada archivo es cifrado con una clave AES única, y dicha clave se cifra posteriormente con una clave pública RSA embebida en el binario. En teoría, este diseño impide el descifrado sin la clave privada RSA. Sin embargo, la seguridad real del sistema no depende únicamente del algoritmo elegido, sino también de su **implementación**.

En este documento se analiza un fallo crítico en la implementación de Ryuk: la generación de claves AES mediante `System.Random()` sin semilla explícita. Este detalle introduce una **predictibilidad temporal** que, bajo ciertas condiciones, permite reconstruir las claves AES y descifrar archivos sin necesidad de la clave RSA privada.

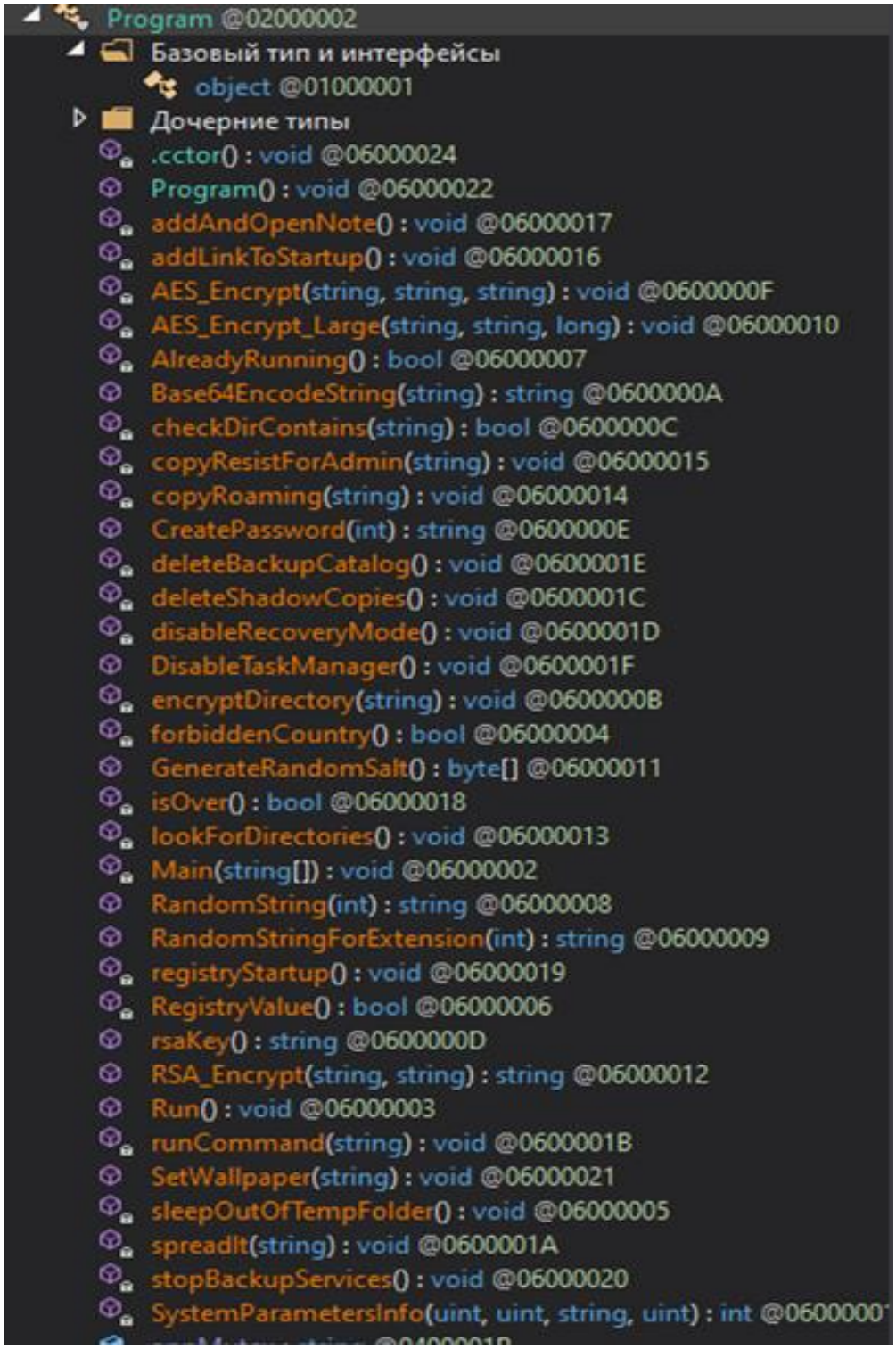
Incidente

En la segunda mitad de 2023, el equipo de investigación de incidentes PT CSIRT (del PT Expert Security Center de *Positive Technologies*) analizó un incidente ocurrido en la infraestructura de uno de sus clientes. Por varios indicios, se determinó que para generar el archivo malicioso probablemente se utilizó el constructor público Chaos Ransomware Builder, una herramienta que permite a los atacantes personalizar programas de cifrado maliciosos.

Con su ayuda, por ejemplo, es posible modificar el nombre del proceso, el nombre del archivo de la nota de rescate, el texto de las demandas, definir una lista de extensiones de archivos a cifrar, crear un descifrador y realizar otras configuraciones.



El propio troyano está escrito en .NET, lo que permite analizar fácilmente sus funciones, por ejemplo, utilizando DNSpy.



Al revisar las funciones principales, los investigadores descubrieron que este codificador cifra los archivos utilizando el algoritmo AES, luego cifra las claves AES con una clave pública RSA integrada y las anexa al final de los archivos cifrados.

Generación de claves AES en Ryuk

En el código descompilado de Ryuk se observa el siguiente fragmento, responsable de generar la clave AES para cada archivo:

```
v
random.Next("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ12345678
90*!=&?&/".Length)
            ]
        );
    }
    return stringBuilder.ToString();
}
```

Cada archivo recibe una clave distinta, pero el generador pseudoaleatorio utilizado es `System.Random`, inicializado **sin semilla explícita**.

El problema de `System.Random()` sin semilla

En .NET Framework 4.x, cuando se instancia `new Random()` sin proporcionar una semilla, el runtime utiliza internamente el valor de [3]: `Environment.TickCount`

`Environment.TickCount` representa el número de milisegundos transcurridos desde el arranque del sistema. Esto implica que:

- La secuencia de valores generados por `Random` es **determinista**.
- Si se conoce el valor de `TickCount` en el momento de la generación de la clave, la secuencia puede reproducirse exactamente.

Por tanto, la clave AES generada por Ryuk **no es verdaderamente aleatoria**, sino dependiente del tiempo de ejecución.

Reproducción de la clave AES

El comportamiento determinista de `Random(seed)` puede demostrarse fácilmente:

```
int seed = 264152453;
string key1 = CreatePassword(40, seed);
string key2 = CreatePassword(40, seed);
// key1 == key2
```

Esto confirma que, conociendo el valor de `seed`, es posible reconstruir exactamente la clave AES utilizada para cifrar un archivo concreto.

Estimación del valor de `Environment.TickCount`

El siguiente paso consiste en estimar el valor de `Environment.TickCount` en el momento del cifrado. Esto puede hacerse utilizando metadatos del archivo cifrado:

1. Se obtiene la marca de tiempo `LastWriteTime` del archivo.
2. Se calcula la diferencia entre la hora actual y dicha marca.
3. Se resta esa diferencia del `Environment.TickCount` actual.

Ejemplo en C#:

```
static int GetTickCountAtFileCreation(string filePath)
{
    FileInfo fileInfo = new FileInfo(filePath);
    DateTime modtime = fileInfo.LastWriteTime;
    int tickCountAtCreation = Environment.TickCount -
        (int)((DateTime.Now - modtime).TotalMilliseconds);
    return tickCountAtCreation;
}
```

Debido a pequeñas diferencias temporales (escritura en disco, planificación del sistema), el valor obtenido suele presentar una desviación de algunos milisegundos.

Fuerza bruta acotada del espacio de semillas

Para compensar esta desviación, se realiza una búsqueda en un rango reducido de valores alrededor del `TickCount` estimado:

```
int tick = GetTickCountAtFileCreation(encryptedFile);
for (int i = tick - 100; i <= tick + 100; i++)
{
    string candidateKey = CreatePassword(40, i);
    if (AES_Decrypt(encryptedFile, candidateKey, outputFile))
    {
        Console.WriteLine("Clave encontrada: " + candidateKey);
        break;
    }
}
```

El espacio de búsqueda es pequeño (centenas de valores), lo que hace viable el ataque en términos computacionales.

Verificación mediante encabezados mágicos

Para determinar si una clave candidata es correcta, se emplea la verificación de **magic bytes**. Tras descifrar el archivo en memoria, se comprueban los primeros bytes:

```
static bool IsValidFileHeader(byte[] data)
{
    return MatchHeader(data, new byte[] { 0x50, 0x4B, 0x03, 0x04 }) || // ZIP / DOCX
           MatchHeader(data, new byte[] { 0x4D, 0x5A }) ||             // EXE
           MatchHeader(data, new byte[] { 0xD0, 0xCF, 0x11, 0xE0 });   // OLE
}
```

Si el encabezado coincide, se asume que la clave AES es correcta y el archivo se escribe en disco.

Prueba de concepto

En un entorno controlado, este método permite:

- Reconstruir claves AES generadas por Ryuk.
- Descifrar archivos sin acceso a la clave privada RSA.
- Verificar automáticamente el éxito del descifrado.

El funcionamiento del programa implementado se puede ver en el siguiente video: https://www.youtube.com/watch?v=WjGb05mq4S0&feature=youtu.be

La prueba de concepto demuestra que el cifrado de Ryuk es **criptográficamente sólido en diseño**, pero **débil en implementación**.

Limitaciones del enfoque

Este método no es universal y presenta varias limitaciones importantes:

- Si el sistema se reinicia tras el cifrado, `Environment.TickCount` se reinicia.
- Si no se dispone de timestamps fiables, la estimación del seed puede fallar.
- En cifrados concurrentes o multihilo, el margen de error aumenta.
- El contador `TickCount` se reinicia aproximadamente cada 49,7 días.

Implicaciones de seguridad

Desde el punto de vista defensivo, este análisis demuestra que:

- El uso de RNG no criptográficos invalida esquemas de cifrado fuertes.
- La implementación es tan crítica como el algoritmo.
- El análisis forense temporal puede ser clave para la recuperación de datos.

Desde el punto de vista del desarrollo seguro, la lección es clara:

> Nunca deben utilizarse generadores pseudoaleatorios generales (`System.Random`) para fines criptográficos.

Conclusión

Ryuk emplea algoritmos criptográficos robustos, pero introduce una debilidad crítica al generar claves AES mediante un generador pseudoaleatorio dependiente del tiempo. Esta decisión permite, en escenarios favorables, reconstruir las claves y descifrar archivos sin pagar el rescate.

Este caso subraya cómo pequeños errores de implementación pueden tener consecuencias significativas incluso en malware avanzado, y constituye un ejemplo práctico de la importancia de la criptografía aplicada correctamente en ciberseguridad [2][5].

Bibliografía

[1] CrowdStrike. (2021). *Ryuk Ransomware: Targeted Attacks Against Enterprises*. CrowdStrike Intelligence Report.

[2] IBM Security X-Force. (2021). *Cryptographic Flaws in Prometheus and Ryuk Ransomware*. IBM Security Blog.

[3] Microsoft. (n.d.). *System.Random Class*. Microsoft Learn. https://learn.microsoft.com/dotnet/api/system.random

[4] Microsoft. (n.d.). *Environment.TickCount Property*. Microsoft Learn. https://learn.microsoft.com/dotnet/api/system.environment.tickcount

[5] Microsoft. (n.d.). *RandomNumberGenerator Class*. Microsoft Learn. https://learn.microsoft.com/dotnet/api/system.security.cryptography.randomnumbergenerator

[6] MITRE ATT&CK®. (n.d.). *Software: Ryuk*. https://attack.mitre.org/software/S0446/

[7] Positive Technologies. (2019). *Ryuk ransomware cryptographic weaknesses*. Xakep Magazine. https://xakep.ru/2019/