Analysing meterpreter payload with Ghidra

Remco

21 POSTS

ISC HANDLER

Yesterday I found a powershell script using urlscan.io which can be found. I didn't (and still don't) have any idea about the origins, being benign or malicious. Spoiler, it is (just) a meterpreter reverse-https payload being delivered using Metasploit's Web Delivery.

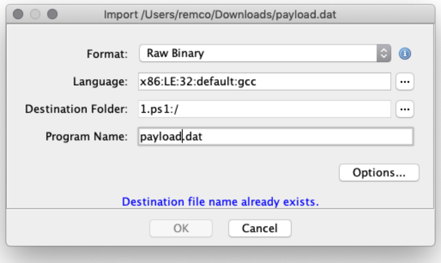
Urlscan is a great and powerfull tool to analyse webpages. It delivers reports about how the page loads, creates screenshots, stores interesting files and extracts all kind of indicators. Urls can be scanned manually or by the api. There are many automated submissions, like links that have been included in emails or are suspicious. The service helps to find other domains running from the same ip, similar pages and campaigns.

Searching for 1.ps1 using urlscan delivers all kind of powershell scripts (many malicious), as also the one I found. Just to add some context, I searched for other occurences of the ip address and file hash delivers, but found just one single result.

The powershell contains a base64 encoded payload which will be executed by starting a new powershell session with the script as argument. Using Cyberchef it is easy to decode the base64 payload as can be shown here. Multiple of my dear handler colleagues have written about this useful service already. Cyberchef (runs client side only) makes it easy to create recipes, that will transform the data by just dropping new operations (which are many predefined) to the recipe. This step only base64 decodes the payload, but the next step deflates the payload also.

Step 2 contains the encoded reverse-https Meterpreter payload that will be loaded and executed in memory. If we now extract the payload and extract it using another recipe we have the shellcode and we'll load this into Ghidra. Ghidra is the Software Reverse Engineering (SRE) suite of tools which are developed (and now opened) by the NSA. Currently the github repository contains only a placeholder, but it says it will be made available opensource. There has been tweeted a lot about Ghidra and overall reactions are positive. Positive reactions are about the decompiler, the ability for collaborating with colleagues, support for adding multiple binaries to a single project, ability to undo, diffing, many supported processors and the analysis. Negative reactions are that it is based on java, supports no debugging and (steep) learning curve. A more thorough overview can be found in this article of Joxean Koret.

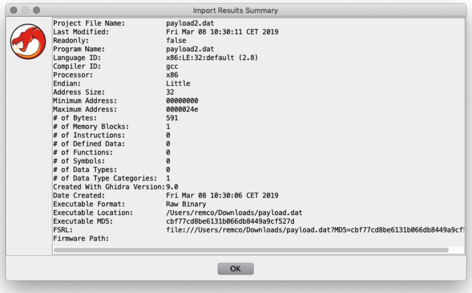
Just to highlight a few features of Ghidra, we'll load the binary. After loading the file we have to pick the language, which is x86 32 bits and the binary can be analysed.



Format: Raw Binary
Language: x86:LE:32:default:gcc
Destination Folder: 1.ps1:/
Program Name: payload.dat

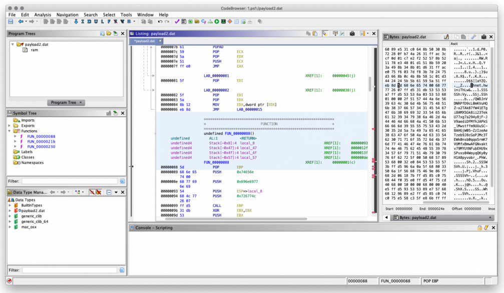
Destination file name already exists.

After importing it will show a small summary about the results.

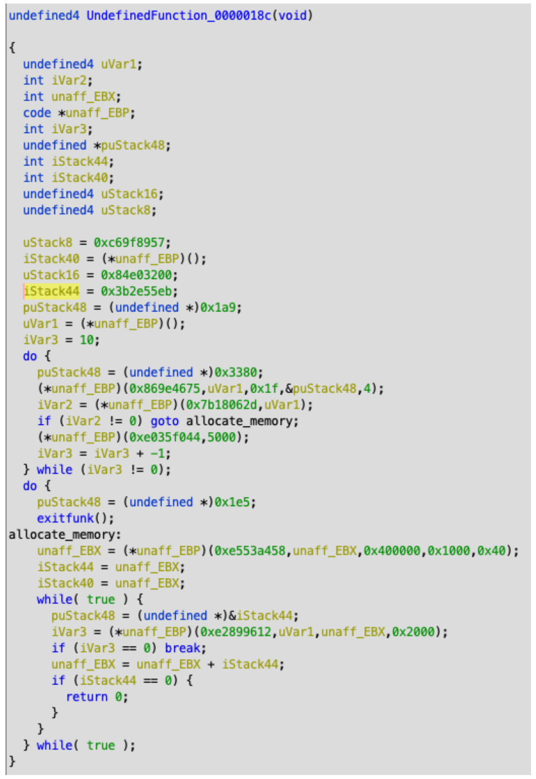


```
Project File Name:          payload2.dat
Last Modified:              Fri Mar 08 10:30:11 CET 2019
Readonly:                   false
Program Name:               payload2.dat
Language ID:                x86:LE:32:default (2.8)
Compiler ID:                gcc
Processor:                  x86
Endian:                     Little
Address Size:               32
Minimum Address:            00000000
Maximum Address:            0000024e
# of Bytes:                 591
# of Memory Blocks:         1
# of Instructions:          0
# of Defined Data:          0
# of Functions:             0
# of Symbols:               0
# of Data Types:            0
# of Data Type Categories:  1
Created With Ghidra Version:9.0
Date Created:               Fri Mar 08 10:30:06 CET 2019
Executable Format:          Raw Binary
Executable Location:        /Users/remco/Downloads/payload.dat
Executable MD5:             cbf77cd8be6131b066db8449a9cf527d
FSRL:                       file:///Users/remco/Downloads/payload.dat?MD5=cbf77cd8be6131b066db8449a9cf
Firmware Path:
```

The payload start address (0x0) needs to be disassembled manually, as it doesn't recognise the code. After disassembling the first bytes, the other code will following and you'll get the this screen. The code can be annotated, functions created, diffed etc.



Ghidra will show the decompiled function next to the assembly view, a sample of decompilated function (the http request and response part) looks like this.



```
undefined4 UndefinedFunction_0000018c(void)

{
  undefined4 uVar1;
  int iVar2;
  int unaff_EBX;
  code *unaff_EBP;
  int iVar3;
  undefined *puStack48;
  int iStack44;
  int iStack40;
  undefined4 uStack16;
  undefined4 uStack8;

  uStack8 = 0xc69f8957;
  iStack40 = (*unaff_EBP)();
  uStack16 = 0x84e03200;
  iStack44 = 0x3b2e55eb;
  puStack48 = (undefined *)0x1a9;
  uVar1 = (*unaff_EBP)();
  iVar3 = 10;
  do {
    puStack48 = (undefined *)0x3380;
    (*unaff_EBP)(0x869e4675,uVar1,0x1f,&puStack48,4);
    iVar2 = (*unaff_EBP)(0x7b18062d,uVar1);
    if (iVar2 != 0) goto allocate_memory;
    (*unaff_EBP)(0xe035f044,5000);
    iVar3 = iVar3 + -1;
  } while (iVar3 != 0);
  do {
    puStack48 = (undefined *)0x1e5;
    exitfunk();
allocate_memory:
    unaff_EBX = (*unaff_EBP)(0xe553a458,unaff_EBX,0x400000,0x1000,0x40);
    iStack44 = unaff_EBX;
    iStack40 = unaff_EBX;
    while( true ) {
      puStack48 = (undefined *)&iStack44;
      iVar3 = (*unaff_EBP)(0xe2899612,uVar1,unaff_EBX,0x2000);
      if (iVar3 == 0) break;
      unaff_EBX = unaff_EBX + iStack44;
      if (iStack44 == 0) {
        return 0;
      }
    }
  } while( true );
}
```

The payload uses a hashed functions to prevent presence of strings within the payload containing the system functions, which makes it less readable.

After analyses this is just a default Meterpreter payload where a reverse https shell will be opened to a Meterpreter handler.

Meterpreter http(s) handlers will use the default "It works" page we know from Apache, but only a bit different. As the default Apache index.html contains an extra **\n** (sha256: f2dcc96deec8bca2facba9ad0db55c89f3c493 7cd6d2d28e5c4869216ffa81cf and 45 bytes), where meterpreter doesn't (sha256: 8f3ff2e2482468f3b9315a433b383f0cc0f9eb5 25889a34d4703b7681330a3fb and 44 bytes). If we search the meterpreter hash for Censys we'll find over two thousand suspected meterpreter servers. Maybe something to blacklist?

Remco Verhoef (@remco_verhoef)
ISC Handler - Founder of DutchSec
PGP Key

Subscribe

Mar 8th 2019
6 months ago

Great write-up, thank you for providing!

I think the Censys results show more servers however, over 53 thousand if I read correctly.

Quote

Anonymous

Mar 8th 2019
6 months ago

← Next Thread Previous Thread →

Sign Up for Free or Log In to start participating in the conversation!