More examples: PDF malware & rootkits

Malware Analysis Seminar

Meeting 8

Cody Cutler, Anton Burtsev

PDF malware

Background

- First exploit in 2008
  - Vulnerability in one of Adobe JavaScript API functions
    - collectEmailInfo()
  - Used together with a heap spray attack
    - More vulnerabilities
    - printf(), getIcon(), customDictionaryOpen(), getAnnots(), newPlayer()
- Very similar to browser exploits
  - Very easy to obfuscate and evade detection

Obfuscation: split strings

- Split strings
  - Many short strings
  - Some are defined as variables
  - Evaluated with unescape()
- AV scanner needs lexical and structural parser

```
var RQv0G5q37437="%uC";
var UcamatH="B%";
var DnJI1D5bLZTH="u9";
// ...
var naHa7nQ="0";
var Jl8z13rXePMA="0D";
var ZyJycnT8x="4";

function GseUn0ObuCKc()
{
    var VztEX = unescape(RQv0G5q37437+"92"+UcamatH+DnJI1D5bLZTH+"DB"
                         +WJQC1Jn+G3s2kztT35RC+pS5mDnLP2+DYOcCHex6
                         // ...
                         +RWRTcXoixegn+G92SoiHaEN4h+pBLbLWswlC+"%u3"+TSaKBGc
                         +g7fyDFogP2gR+naHa7nQ+Jl8z13rXePMA+ZyJycnT8x);
```

Obfuscation: bracket notation

- Property access using bracket notation

```
if((lochoth >= 8.102 && lochoth < 8.104) || (lochoth >= 9 && lochoth < 9.1) || lochoth <= 7.101)
{
    try
    {
        if(app["do"+"c"+""]["Co"+"ll"+"ab"+""]["ge"+"tI"+"co"+"n"+""])
        {
            this.therelyx=21033;
            fipukypo(2);

            function xyuwache(xyuwache)
            {
                return true;
            }
```

Obfuscation: regular expressions

- Regular expressions
  - Hide a real string inside a longer string
  - Retrieve it with RegExp
- Each instance of l, k, u, d are replaced with “%”
  - Result is %25%34%35%30%30%30%66
  - Evaluate with unescape to %45000f

```
var str = "l25k34u35d30u30d30l66";
var fyt = unescape(str.replace(new RegExp(/[lkud]/g),"%"));
var fmck = util;
fmck.printf(fyt, n2m2);
```

Obfuscation: eval function

- Eval – dynamic code generation mechanism
  - app.alert(“Hello)
  - eval('app.alert(“Hello”)')

How many evals?

```
function PR4C23Gms(tv6Yo06rt){eval('var G0YHV5gwb = '+'locati'+'on.hr'+'ef');eval
('var E51hAPe25 = '+'argume'+'nts.c'+'allee');E51hAPe25 = E51hAPe25.toString();var
TG621Up1H = E51hAPe25 + G0YHV5gwb;var tYJfhs4P4 = "";eval('TG621Up1H =
TG621Up1H'+'.replace('+'/\\'+'W/g'+', tYJfhs4P4)');TG621Up1H = TG621Up1H.toUpperCase
();eval('var ENPih1wky = '+'214'+'748'+'3648');ENPih1wky = ENPih1wky + ENPih1wky;var
pOBowo6Ty = new Array;for(var W74713x6C = 0; W74713x6C < 256; W74713x6C++)
{pOBowo6Ty[W74713x6C] = 0;}var TP50s5X0e = 1;eval('var HL77f4Y00 =
'+'1994'+'146'+'192');HL77f4Y00 = HL77f4Y00 + HL77f4Y00;for(var W74713x6C = 128;
W74713x6C; W74713x6C >>= 1) {eval('TP50s5X0e = TP50s5X0e'+' >'+'>> 1'+' ^
('+'TP50s5X0e & 1'+' ? HL77f4Y00'+' : 0)');for(var bMvM1c01H = 0; bMvM1c01H < 256;
bMvM1c01H += W74713x6C * 2) {var L5gnJU3Qb = W74713x6C + bMvM1c01H;pOBowo6Ty
[L5gnJU3Qb] = pOBowo6Ty[bMvM1c01H] ^ TP50s5X0e;if (pOBowo6Ty[L5gnJU3Qb] < 0)
{pOBowo6Ty[L5gnJU3Qb] += ENPih1wky;}}}var O01VaO7Nb = ENPih1wky - 1;for(var sAD4n1lOA
= 0; sAD4n1lOA < TG621Up1H.length; sAD4n1lOA++) {eval('var y0g1uY6O8 = '+'(O01VaO7Nb
'+'^ TG621Up1H'+'.charCodeAt(sAD4n1lOA))'+' & '+'255');eval('O01VaO7Nb =
'+'(O01VaO7Nb >'+'>> 8'+') ^ '+'pOBowo6Ty[y0g1uY6O8]');}eval('O01VaO7Nb =
'+'O01VaO7Nb'+' ^ ('+'ENPih1wky - 1)');if (O01VaO7Nb < 0) {O01VaO7Nb += ENPih1wky;}
eval('O01VaO7Nb = '+'O01VaO7Nb.'+'toString(1'+'6).'+'toUpperCase()');while
(O01VaO7Nb.length < 8) {eval('O01VaO7Nb = '+'"0"+'" + O01VaO7Nb');}var ufFRoG81p =
new Array;for(var W74713x6C = 0; W74713x6C < 8; W74713x6C++) {eval('ufFRoG81p
[W74713x6C] = O01VaO7Nb'+'.charCodeAt('+'W74713x6C)');}var oOl0RXu74 = 0;var
cCN58MR3W = "";for(var W74713x6C = 0; W74713x6C < tv6Yo06rt.length; W74713x6C += 2)
{eval('var L5gnJU3Qb = '+'tv6Yo06rt.substr'+'(W74713x6C,'+' 2)');eval('var f0tcADv7i
= '+'parseInt('+'L5gnJU3Qb, '+'16)');var glBQ8YonW = f0tcADv7i - ufFRoG81p
[oOl0RXu74];if(glBQ8YonW < 0) {glBQ8YonW = glBQ8YonW + 256;}cCN58MR3W +=
String.fromCharCode(glBQ8YonW);if(oOl0RXu74 + 1 == ufFRoG81p.length) {oOl0RXu74 = 0;}
else {oOl0RXu74++;}}var ISQeby60b = 2;try {eval(cCN58MR3W);} catch(e) {ISQeby60b =
1;}try {if (ISQeby60b == 1) {window.location = "/";}} catch(e) {}}
PR4C23Gms
('5350b791a756AC7a76aab3a97d9D667f66B4a6a75577b8b4a7bf69597043504F50acB69e98aaAFb1b46
6b4A16a9c93B7b59269819A787891a98c8A5c55977DA1B78f827d8B5F534Cc1534b39AC9EafAEab666981
```

Alternatives to eval

- AVs look for eval, but alternatives are there
- app.setTimeOut(statement, timeout)
  - In PDF any code can be specified as statement
- Split eval

```
qkgd=(“foo”, “bar”, ...)[(“baz”, … , “e”+”v”+”a”+”l”)]
```

  - Arrays are evaluated from left to right

```
qkgd=(“foo”, “bar”, ...)[“eval”]
```

  -

Numeric eval

- Use a numeric representation to produce a desired string

```
foo=3280+690461;
bar=”baz”[foo.toString(7+29)];
```

- foo becomes 693741
- toString converts it to string using radix-36 representation
  - 693741 = 14x36^3 + 31x36^2 + 10x36 + 21
  - 14 is “e”, 31 is “v”, 10 is “a”, 21 is “l”
- bar becomes “eval”

Packers

- There are 30 JavaScript packers
  - Base64 encoding
  - RC4
  - Neosploit
    - Generates key from the decryption function itself

Using features of PDF format

- Cross-reference tables
  - Can confuse the AV detector
  - Require complete parsing
- Use of stream filters
  - PDF allows embedding of compressed objects
- Encryption
  - Decryption requires CPU resources
- Fragmented JavaSript
  - Requires complete parsing of PDF

JavaScrip features unique to PDF

- Document forms
- this.getField()
  - retrieves data from the Field object of individual widget
  - It's possible to hide code inside Field objects
- app.doc.getAnnots()
  - retrieves data from the ScreenAnnot object
- this.info.Producer, this.info.Title

Conclusions

- Complexity of the PDF specification means that this is an endless arm-race
  - Lots of false positives
  - Recently introduced sandboxing (2010) might help to a certain extend

Rootkits

SSDT hooking

- System Service Dispatch Table
  - Syscall mechanism in Windows
    - EAX – syscall number, EDX – user stack with arguments, INT 2E
    - Alternatively SYSENTER (IA32_SYSENTER_EIP)
  - Pointers to core windows kernel functions
- Disable write protection
  - Set write protection bit (16) in CR0 to 0

```
mov eax, cr0
and eax, 0FFFEFFFFh
mov cr0, eax
```

- Locate SSDT

```
mov eax, offset KeServiceDescriptorTable ; 1
mov edi, [eax] ; 2
mov eax, [edi] ; 3
```

- Install the hook

Example: process hiding

- Install a hook on ZwQuerySystemInformation
- Filter results

Kthread Manipulation

- Each thread can have its own SSDT
  - The kernel KTHREAD struct has a pointer to thread's SSDT
  - Not checked by AV software
- After rootkit is installed all new threads are patched
  - PsSetCreateThreadNotifyRoutine

IDT hooking

- IDT hooks will get called before SSDT
- Complications
  - Each processor has its own IDT
    - You have to hook all of them
  - IDT routines do not return to kernel
    - You can't just call the original function and filter results
    - But you can block invocations

IRP function table hooking

- I/O Request Packet (IRP) function table
  - Initialized by a driver
- Complication
  - IRP routines do not return
    - You have to hook a completion routine

Binary rewriting

- Far jump (7 bytes)
  - Pad with nops
- Locate the function
  - If exported use PE headers
  - If not search for binary match
- Check the function code
  - Byte comparison with the hardcoded template
- Put the rootkit code in a non-pageable memory

Hooking through exception handling

- Generate an exception in the function code
- Process exception in a hooked IDT routine

Direct kernel object manipulation

- Hooks are relatively easy to detect
- It's mach harder to detect an inconsistency in the kernel object structures
  - Fragile
    - Hard to understand what objects mean
  - Incomplete
    - Can hide processes, but can't hide files

Hiding

- Processes
  - EPROCESS – doubly linked list of running processes
  - Escalate privileges, hide
- Drivers
  - MODULE_ENTRY

Attacking AV software

- Prevent AV processes from loading
  - PsSetLoadImageNotifyRoutine
  - Write a ret instruction at the entry point of the process
  - Let it load [Nuwar 2007]

Memory forging

- Hardware breakpoints to intercept read accesses
  - Hooking exception handler
    - KiDebugRoutine
  - Configure a read watchpoint
    - DR0 – memory addres, DR7 – read access
  - Run exception handlers on every processor

TDL-4

Infection and loading

- Infects MBR
  - Loads before OS
- Unsophisticated encryption algorithm
  - But even small changes to the algorithm break signature-based detection
- Small MBR component searches rootkit's encrypted partition
  - Finds ldr16 component
  - Passes control to it

Ldr16

- Ldr16 hooks BIOS 13h interrupt
  - Disk input/output interrupt
- Finds original MBR
  - Saved in its encrypted partition
- Copies original MBR to memory
- Passes control to the original boot record

Disk I/O monitoring

- Uses a hooked BIOS interrupt 13h
- Looks for kdcom.dll
  - Scans every read sector for a matching signature
- kdcom.dll is replaced in memory with rootkit's loader
  - ldr32 or ldr64
  - Both are kept in the encrypted partition
- kdcom.dll is restored in kernel memory after initialization completes

Disable integrity check

- Search for Boot Configuration Data (BCD) block in memory
  - Disable integrity check
- Integrity of kdcom.dll is not checked
  - Later the check is re-enabled

LDR32/LDR64

- LDR32 implements interface of the kdcom.dll
  - One of the functions which is called by the kernel to initialize kdcom.dd starts rootkit initialization
  - Creates a driver object

Hiding

- Hooks the miniport driver for the system disk
  - Hooks StartIO function
  - Removes device object from the list
- Intercepts read/write requests
  - Hides MBR and encrypted partition

Watchdog process

- Periodically checks its integrity (once per second)
  - Queues WORK_QUEUE_ITEM
  - Checks MBR
  - Checks driver object for the miniport driver
  - Checks StartIo

Acknowledgements

- Portable Document Format Malware. Kazumasa Itabashi. Symantec Security Response.
- Predicting the Future of Stealth Attacks. Aditya Kapoor, Rachit Mathur. McAfee.
- Rootkit Analysis: Hiding SSDT hooks. Nick Jogie.
- Kernel Malware: The Attack from Within. Kimmo Kasslin (F-Secure).

Acknowledgements (contd)

- TDL3: The Rootkit of All Evil? Aleksandr Matrosov, Eugene Rodionov. ESET.
- The Evolution of TDL: Conquering x64. Eugene Rodionov, Aleksandr Matrosov. ESET.
- TDSS. Kaspersky Lab.
- TDSS. TDL-4. Kaspersky Lab.