Corrupting Memory In Microsoft Office Protected-View Sandbox

Yong Chuan Koh

08th Nov 2017
++

Outline

- Introduction
- Inter-Process Communication (IPC) Mechanism
- Fuzzer Implementation
- Results
- Conclusion and Future Work
• Yong Chuan, Koh (@yongchuank)
• Security Researcher @ MWR Infosecurity (SG) since 2014

• Work:
  – Desktop Applications Security Assessments
  – Web + Mobile Applications Security Assessments
  – Malware RE + Analysis

• Interests:
  – Vulnerability Research
  – Reverse Engineering
MS Office 2016 Protected-View Sandbox

Introduction

• Motivation

• Goals
Introduction

Motivation

• Microsoft Office Bounty Program
  – 15 Mar to 15 Jun, extended to 31 Dec 2017
  – $6000 to $15000, for Protected-View EOP
• Received $0 bounty

Announcing the New Bug Insider Builds on Windows

Extending the Microsoft Office Bounty Program

Microsoft announces the extension of the Microsoft Office Bounty Program through December 31, 2017. This extension is retroactive for any cases submitted during the interim.

The Office Bug Bounty Program complements our continuous threat modeling and security in code reviews, security audits. The Microsoft Cloud and Online Services Bounty Program has individuals actively partnership to protect our customers. Rewarding deeper thought in improving the security of Office.

Office Insider Builds give users early access to the latest Office capabilities and security innovation. By testing against these early builds, you can potentially find prior to public release. This helps improve quality and protect customers.

- Types of vulnerabilities awarded and their details can be found in the Microsoft Office Insider Builds on Windows Bounty Program Terms, including:
  - Elevation of privilege via Office Protected View
  - Macro execution by bypassing security policies to block macros
  - Code execution by bypassing Outlook automatic attachment block policies
- The program duration is from March 15 to December 31, 2017
- Bounty payout ranges during this period will be $6000 to $15000 USD
Introduction

++

Motivation

- Missing puzzle piece in “sandbox–everything” trend
  - Edge, Chrome, IE, Adobe Reader

- Increasing popularity in hacking contest
  - New categories in Pwn2Own 2017, Zer0Fest 2017
  - MS Office with Protected–View
Motivation

• No advisories (AFAIK) on Protected-View memory corruptions
  – 100% secure code?

• Experiment fuzzing technique on different targets
  – “Fuzzing the Windows Kernel”, HITB GSEC 2016
Introduction

++

Goals

• Find all Protected-View memory corruptions
  – And fix them!

• Hopefully increase cost for attackers
  – And hacking contest participants? 😊
  – New and creative bypass techniques!

• Learn about vulnerabilities trends

• Have fun!
Introduction

++

Baseline

• Fuzzing and analysis of MS Office 16.0.4266.1001 (vanilla)
  – Reuse existing .idb with notes + comments
  – *Assumed* no new IPC messages introduced
MS Office 2016 Protected-View Sandbox

Inter-Process Communication (IPC) Mechanism

• Attack Surfaces
• IPC Internals
• IPC Message Format
Attack Surfaces

- Unlike any other sandboxes
  - Aims to provide text-view of file content
  - Does not provide full features of application
  - Reduced attack surfaces (eg: no COM)
  - Only for MOTW files
IPC Mechanism

++

Attack Surfaces

- Protected-View boundary
  - AppContainer SID
  - ALL_APP_PACKAGES SID (S-1-15-2-1)
  - Unknown MSOffice Capability SID (S-1-15-3-2929230137-1657469040)

- Accessible securable objects
  - %UserProfile%\AppData\Local\Packages\<AC>\*
  - HKCU\Software\Microsoft\Office\* (KEY_READ)
  - HKLM\Software\Microsoft\MSLicensing\Store\* (KEY_READ|KEY|WRITE)

- Sensitive data exfiltration (if any) requires...
  - PV has no internet capabilities
  - IPC for broker to send data out as WER
  - Requires a compromised WER server, which doesn’t meet MSRC servicing bar
IPC Mechanism

++

Attack Surfaces

- AppContainer

- System-Calls: “Generic” Escapes

- Logic Quirks: “Trendy” Escapes

- IPC: “Old-School” Escapes

[*] https://medium.com/@mxatone/how-bad-design-decisions-created-the-least-secure-driver-on-windows-33e662a502fe
IPC Mechanism

+++ IPC Internals

- All files are rendered in same PV process
- Broker has to differentiate IPC msgs from each MOTW file
IPC Mechanism

IPC Internals

*Only relevant object fields are illustrated*
**IPC Mechanism**

++

**Protected-View Scenario: App Broker**

- **Broker**
  - “Core” Office App: Winword, Excel or Powerpoint
- **Sandbox**
  - “Core” Office App: Winword, Excel or Powerpoint

![Diagram showing the IPC Mechanism with the following data structures:
- PVFile:
  - ulViewID
  - hWnd
  - lpwzFileName
  - lpwzTempFileName
  - lpPVCoreAttr
  - bSessionHyperlinks

- PVCOREAttr:
  - lpPVAppAttr
  - hWnd
  - lpDRMStream
  - lpTaskList

- PVAppAttr:
  - ...

Serviced in MSO.DLL

Servicing DLL depends on broker

<table>
<thead>
<tr>
<th>Process</th>
<th>Integrity</th>
</tr>
</thead>
<tbody>
<tr>
<td>WINWORD.EXE</td>
<td>Medium</td>
</tr>
<tr>
<td>EXCEL.EXE</td>
<td>Medium</td>
</tr>
<tr>
<td>POWERPNT.EXE</td>
<td>Medium</td>
</tr>
</tbody>
</table>
Protected-View Scenario: Outlook Broker

- Broker
  - Outlook
- Sandbox
  - “Core” Office App: Winword, Excel, Powerpoint or Visio Previewer

IPC Mechanism

Serviced in MSO.DLL
Servicing DLL depends on broker

PVFile
- ulViewID
- hOPHWnd
- lpwzFileName
- lpwzTempFileName
- lpPVCoreAttr
- bSessionHyperlinks

PVCoreAttr
- lpPVAppAttr
- hOPHParentWnd
- lpDRMStream
- lpTaskList

PVAppAttr
**Named-Pipe Properties**

- Name "\\pipe\OfficeUser_OICE_16_974FA576_32C1D314_{ID}"
  - {ID} = rand_s() & 3FFFh (16k possible values)
- Bidirectional (PIPE_ACCESS_DUPLEX)
- Local Machine Client (PIPE_REJECT_REMOTE_CLIENTS)
- Message Stream (PIPE_TYPE_MESSAGE)
- Max In/Out Buffer Size 2000h
**IPC Mechanism**

**Message Format**

- **Protected-View Core (PVCore)**
  - Independent of broker application
  - 16-bytes IPC message header
  - MSO.DLL

- **Protected-View App (PVApp)**
  - Dependent on broker application (Word, Excel or Powerpoint)
  - 20-bytes IPC message header
  - WWLIB.DLL, EXCEL.EXE or PPCORE.DLL
## IPC Mechanism

### Message Format

**Message Header**

- **ulVirtualKey**: Subset of vkey codes, to query state `GetAsyncKeyState()`
- **ulMsgTag**: Type of IPC message
- **ulMsgID**: Matches IPC request to response; can be random
- **ulMsgSize**: Total size of IPC message, including header

**Message Body**

- Body Format:
  - Dependent on type of IPC message (MsgTag)
  - May be optional
  - May be fixed or varied size
  - Some PVCORE messages also have ViewID in body

**Body Format**

- **ulViewID**: Ordered sequence of file opened in Protected-View

---

**PVCore Header Format**

- **ulVirtualKey**: Subset of vkey codes, to query state `GetAsyncKeyState()`
- **ulMsgTag**: Type of IPC message
- **ulMsgID**: Matches IPC request to response; can be random
- **ulMsgSize**: Total size of IPC message, including header

**PVApp Header Format**

- **ulVirtualKey**: Subset of vkey codes, to query state `GetAsyncKeyState()`
- **ulMsgTag**: Type of IPC message
- **ulMsgID**: Matches IPC request to response; can be random
- **ulMsgSize**: Total size of IPC message, including header
- **ulViewID**: Ordered sequence of file opened in Protected-View
**IPC Mechanism**

++

**Message Format**

- 23 MSO (PVC) messages (0x001000 – 0x161000; +0x10000)
- 15 WINWORD messages (0x001100 – 0x0E1100; +0x10000)
- 17 MSO messages (0x001200 – 0x101200; +0x10000)
- 23 MSO (PVC) messages (0x001000 – 0x161000; +0x10000)
- 17 POWERPNT messages (0x001300 – 0x101300; +0x10000)

* IPC Sanity Check Functions
* IPC Service Request Functions
IPC Mechanism

++

Message Format

• Broker does general checks on IPC message header
  – NumberOfBytesRead of ReadFile() <= 2000h
  – Header.MsgSize == NumberOfBytesRead

• Broker does sanity-checks on IPC message body
  – Header.MsgSize == IPC static msg size, or
  – Header.MsgSize >= IPC min (dynamic) msg size
  – Update (WCHAR*, USTRING*, BYTE*, Array[], etc) pointers, if not NULL

```c
typedef struct _USTRING
{
    UINT16 Len; //len of Buffer[], excl NULL
    WCHAR Buffer[N]; //wide-character string
} USTRING;
```
### IPC Mechanism

#### Message Format

- Broker does sanity-checks on IPC message body
  - $$\text{Ptr}_1 + \text{Header.MsgSize} = \text{Ptr}_2 + \text{sizeof(s.fields)} + \text{sizeof(d.fields)}$$

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULONG</td>
<td>ulVirtualKey</td>
</tr>
<tr>
<td>ULONG</td>
<td>ulMsgTag</td>
</tr>
<tr>
<td>ULONG</td>
<td>ulMsgID</td>
</tr>
<tr>
<td>ULONG</td>
<td>ulMsgSize</td>
</tr>
<tr>
<td>ULONG</td>
<td>ulViewID</td>
</tr>
<tr>
<td>WCHAR*</td>
<td>UStrInChars.Buffer</td>
</tr>
<tr>
<td>UINT8</td>
<td>ui8Unknown_18</td>
</tr>
<tr>
<td>UINT8</td>
<td>ui8Unknown_19</td>
</tr>
<tr>
<td>UINT16</td>
<td>ui16Padding</td>
</tr>
<tr>
<td>USTRING.UINT16</td>
<td>UStrInChars.Len</td>
</tr>
<tr>
<td>USTRING.WCHAR</td>
<td>UStrInChars.Buffer[Len+1]</td>
</tr>
</tbody>
</table>
Fuzzer Implementation

- Fuzzing Decisions
- Fuzzing Setup
Fuzzer Implementation

++

Protected-View Scenario

- App-broker: Higher code coverage of overall IPC messages
  - PVCore and PVApp (vs Outlook-broker: PVCore subset only)

- Outlook-broker: (Possibly) Higher code coverage of PVCore (Subset 1) messages

- Decision: Fuzz both scenarios, but dedicate longer fuzz time for App-broker
Valid IPC Message Fields

- Valid ViewID value of 1
  - First Protected-View file
- Valid MsgSize value
  - Fixed value for static-sized msg, or max 2000h for dynamic-sized msg
- Valid MsgTag values
  - Depends on chosen Protected-View scenario
- Valid USTRING structure, where applicable
- Non-NULL arbitrary pointer value
  - Sanity-check functions will update accordingly
Fuzzer Implementation

++

Improve Fuzzing Efficiency

- RE message bodies to ↑ code-coverage, ↓ time
- Only 72 (23 MSO+15 WWLIB+17 EXCEL+17 PPCORE) msgs

```c
typedef struct EXCEL_071200 : IpcHeader {
    WCHAR* UStrInChars.Buffer; //Non-NULL
    UINT8 ui8Unknown_18; //0-2, or others
    UINT8 ui8Unknown_19; //0-1, or others
    UINT16 ui16Padding; //unused
    USTRING UStrInChars;
};
```
Fuzzer Implementation

Cross-Check IPC Message Format

- Sanity-checks would discard most dumb-fuzzed IPC msgs
- Use sanity-checks for the fuzzer advantage...
  - Also verifies reverse-engineering + fuzzer code
  - Produces CVE-2017-8692!
Fuzzer Implementation

++

Fuzzing Algorithm

• No sophisticated fuzzing framework

• Enable page heap “gflags.exe /i <image> +hpa”

• DLL injection of fuzzer DLL into Protected-View process

• Randomly select valid PVCore + PVApp IPC message

• Each IPC message is generated with as many valid fields as possible

• Sends IPC message through Named-Pip “\pipe\OfficeUser_OICE_16_974FA576_32C1D314_<****>”
Crash Reproduction

- Manual reproduction of crash
- Check if crash occurs in IPC-processing functions
  - If yes, identify this as PVTarget message
- Rerun fuzzing, with biased selection for PVTarget message
- *Pray* that crash can be reproduced after X period of time
- Use DebugView for manual minimization
  - Log Header->MsgTag only
**Fuzzer Implementation**

**Crash Minimization**

1. Run fuzzer-RM
2. Has Crashed?
   - Yes: Get set of MsgTags, Remove N MsgTags from set, Add code to log MsgBody
   - No: Return N MsgTags to set, Min set to crash?
     - No: Last Run
     - Yes: Add code to log MsgBody

*Fuzzer-RM: Generate IPC msg from predefined list instead of random*
Fuzzer Implementation

++

Hardware

• Host: HP Elite Notebook
  – Intel i7–4600U CPU
  – 8 GB Memory

• Guest: 1–4 VMWare Workstation Images
  – 1 CPU, 1 GB Memory
  – All crashes occur while testing fuzzer code with 1 Guest
  – “Scale up” fuzzing to 4 Guests, for ~5 days * ~9 hours
Results

- Statistics
- Case Study: CVE-2017-8502
- Case Study: CVE-2017-8692
- MSRC 40761
Results

++

Statistics

- CVE-2017-8502: Mem Corruption Vulnerability
- CVE-2017-8692: Uniscribe RCE
- *NSI: RtlFailFast() DOS

- MSRC 40761: Invalid Pointer Typecast (Open)
- *NSI: RtlFailFast() DOS

- MSO.DLL 45%
- WWLIB.DLL 22%
- EXCEL.EXE 33%
- PPCORE.DLL 0%

*NSI: Stack Exhaustion DOS
*NSI: Dir Traversal via WER
*NSI: memcpy() Read-AV DOS
*NSI: Null-Ptr Dereference DOS

* NSI: Non-Security Issue
Results

++

CVE-2017-8502: Memory Corruption Vulnerability

- Aka “Out-of-Boundary Array Access”
- Affects only Excel Protected-View (PVApp)
- Requires two 0E1200h IPC messages

```assembly
sub_AA1C81(): Start service 0E1200h msg
sub_91AE(): HeapAllocWrapper1 (dwBytes)
Allocation size (dwBytes): 26Ch
```
**Results**

++

**CVE-2017-8502: Pre-Trigger**

- First 0E1200h message

```c
EXCEL_0E1200 PvMsg = {};
PvMsg.bSetGlobalFlag = 1;
PvMsg.ui32Unknown_18 = 0xFFFFFFFF;
PvMsg.ui32Unknown_1C = 0xFFFFFFFF;
PvMsg.ui32Unknown_20 = 0x29;
PvMsg.ui32Unknown_24 = 0xFFFFFFFF;
PvMsg.ui32Unknown_28 = 0xFFFFFFFF;
```
Results

++

CVE-2017-8502: Trigger

• Second 0E1200h message

```
EXCEL_0E1200  PvMsg = {};  
PvMsg.bSetGlobalFlag = 0;
PvMsg.ui32Unknown_18 = 0xFFFFFFFF;
PvMsg.ui32Unknown_1C = 0xFFFFFFFF;
PvMsg.ui32Unknown_20 = 0x23;
PvMsg.ui32Unknown_24 = 0xFFFFFFFF;
PvMsg.ui32Unknown_28 = 0xFFFFFFFF;
```
Results

++

CVE-2017-8502: Array Allocation

• Allocation size = EAX * 7Ch
  – ie: allocates for an array of EAX number of unknown objects
  – Size of each unknown object = 7Ch
• Second 0E1200h message
  – Allocation size = EAX (5) * 7Ch = 26Ch

* Funfact: Addr EXCEL+10BC14 appeared in "!heap -p -a ecx" 3 slides earlier
Results

CVE-2017-8502: Array Allocation

- Array pointer is assigned to global variable, dword_C0637C
  - poi(poi(poi(poi(MSO+C0637Ch))+8))+DCh
- Initialization of all unk_obj in array is completed in MSO.sub_120304() loop
++

CVE-2017-8502: Array Usage

Calls MSO_97() when dword_1461918 = 2Fh

- Calls sub_12B744() to get a pointer into 7Ch-obj array
- Note arg_0 = const 15h (from caller)

[ ecx+DCh ] is the pointer to start of 7Ch-obj array, from dword_C0637C
- Offset into 7Ch-obj array = (15h-10h) * 7Ch = 26Ch
- Recall allocation size = 26Ch
- Therefore returns a pointer to end of 7Ch-obj array

OOB dereference at offset 34h from end of 7Ch-obj array

Calling into sub_12B744()…

Ret from sub_12B744()…

Calls crash_sub() with ptr to end of 7Ch-obj array (ESI)
**Results**

**CVE-2017-8502: Execution Trace**

---

**List of Breakpoints**

**First 0E1200h Message**

**Second 0E1200h Message**

ECX = 38002D90 + 26C

= 38002FFC

= assumed-present 6th unk_obj
Results

++

**CVE-2017-8502: Summary**

- **Root-cause**
  - Excel assumes there are always (at least) 6 objects in the array when global dword_1461918 is 2Fh
  - Leading to out-of-boundary array dereference (aka memory corruption)

- **Exploitation**
CVE-2017-8692: Uniscribe RCE Vulnerability

- Affects only Excel Protected-View (PVApp)
- Aka “Out-of-Boundary Array Access”
- Requires one 071200h IPC message
  - Part of Excel Protected-View design
  - Update formula bar with cell content, though editing still not allowed
Results

++

CVE-2017-8692: Trigger

```c
EXCEL_071200  PvMsg = {};
PvMsg.UStrInChars.Buffer = 0x1;
PvMsg.ui8Unknown_18 = 0x2;
PvMsg.ui8Unknown_19 = 0x0;
PvMsg.ui16Padding = 0x00;
PVMsg.UStrInChars = {0x25, L"j(@-g?Mcav)MzM_<m+T[zA46ykI#V5\2Kj|42"};

ScriptItemize (
  [in] pwcInChars  = L"j(@-g?Mcav)MzM_<m+T[zA46ykI#V5\2Kj|42"
  [in] cInChars = 0x00000004
  [in] cMaxItems = 0x00000005
  [in, opt] psControl = &(0x00800009)
  [in, opt] psState = &(0x0001)
  [out] pItems
  [out] pcItems
)
```
Results

++

CVE-2017-8692: Trigger

```
ScriptItemize ( 
    [in] pwcInChars = L"j(@-g?McaMzM <mT[^A46yk#V5V2k]^42"
    [in] cinChars = 0x000000004
    [in] cMaxItems = 0x00000005
    [in, opt] psControl = & (0x00800009)
    [in, opt] psState = & (0x00001)
    [out] pItems
    [out] pcItems
)
```

Crash in gdi32full!ScriptItemize() API call HeapAllocWrapper2 (dwBytes, ppBufferOut)
Results

CVE-2017-8692: Buffer Allocation

- **cInChars** = 4
- Size of UnkBuf = $cInChars \times 4 \times 3$ = 30h bytes

- **cMaxItems** = $cInChars > 2$ ? ($cInChars + 1$) : 2 = 5
- Size of pItems = $(cMaxItems) \times \text{sizeof}(\text{SCRIPT\_ITEM})$ = $(cMaxItems) \times 8$ = 28h bytes

![Diagram showing buffer allocations and calculations]
+++ 

**CVE-2017-8692: gdi32full!FindMatchingPair()**

- Function loops pItems buffer to find matching SCRIPT_ITEM
- Last SCRIPT_ITEM in pItems buffer is a End-of-Array marker

```c
int __stdcall FindMatchingPair(SCRIPT_ITEM **ppItems, SCRIPT_ITEM *pItems_LastItem, ...) {
    SCRIPT_ITEM* var_10;
    if (*ppItems < pItems_LastItem) {
        do {
            var_10 = *ppItems;
            int iCharPosOfCurrScriptItem = (var_10 + 0)->iCharPos;
            int iCharPosOfNextScriptItem = (var_10 + 1)->iCharPos;  // Out-of-Bound Dereference
            int EDX = iCharPosOfNextScriptItem - iCharPosOfCurrScriptItem;
            if (var_10->a.eScript > 0x114) { ... }
            else if (var_10->a.eScript == 0x114) { ... }
            else { ... }
            (*ppItems)++;
        } while (*ppItems <= pItems_LastItem);
    }...
}
```

Each loop dereferences current and next SCRIPT_ITEM

Loop until last SCRIPT_ITEM(!)
Results

++

**CVE-2017-8692: Summary**

`pItems [out]`

Pointer to a buffer in which the function retrieves `SCRIPT_ITEM` structures representing the items that have been processed. The buffer should be `(cMaxItems + 1) * sizeof(SCRIPT_ITEM)` bytes in length. It is invalid to call this function with a buffer to hold less than two `SCRIPT_ITEM` structures. The function always adds a terminal item to the item analysis array so that the length of the item with zero-based index “i” is always available as:

```
pItems[i+1].ICharPos - pItems[i].ICharPos;
```

- **MSDN ScriptItemize() Reference**
  - Responsibility fall on developers to allocate the sufficient sized buffer

- **Recall: Size of pItems = cMaxItems * sizeof(SCRIPT_ITEM)**

- **But there should have been checks in ScriptItemize() API?**
  - “... while (*ppItems <= pItems_LastItem); ...”
  - Loop should have terminated at pItems[-2]?
Results

++

**MSRC 40761: Invalid Pointer Typecast**

- Case still open
- Lucky bug discovered while minimizing for MSRC 40765 RtIFailFast()
- Patch is scheduled for release Dec 2018/Jan 2019
- Will release technical details in due course...
MS Office 2016 Protected-View Sandbox

Conclusion
Conclusion

- Motivation and goals of this project
- 3 attack surfaces for Protected-View sandbox escapes/bypass
- IPC mechanism
  - Internal objects, scenarios and message formats
  - 72 (23 MSO+15 WWLIB+17 EXCEL+17 PPCORE) messages
- Fuzzing optimizations to ↑ code-coverage, ↓ time
Conclusion

Results

- Total issues: 4 (MSO.DLL), 2 (WWLIB.DLL), 3 (EXCEL.EXE), 0 (PPCORE.DLL)
- 3 security issues
- Vulnerabilities trend: Non-linear (buf[n] = x) heap corruption issues
## Conclusion

- Most modern mitigations (eg: DisableWin32kSystemCalls) disabled

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DEP:Enable</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
</tr>
<tr>
<td>DEP:DisableATL</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
</tr>
<tr>
<td>ASLR:BottomUp</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
</tr>
<tr>
<td>ASLR:ForceRelocate</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
</tr>
<tr>
<td>ASLR:*</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>System Call:DisallowWin32kSysCalls</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>ExtensionPoint:DisableExtensionPoints</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>DynamicCode:*</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>CFG:*</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>BinarySignature:*</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>FontDisable:*</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>ImageLoad:*</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
</tbody>
</table>

[*] Results of ProcessMitigations 1.0.7 “Get-ProcessMitigation” on various (x86) MS Office
Future Work

- Check for new Protected-View features
  - New IPC messages (rem 071200h:Update Formula Bar with Cell Content)

- Remaining attack surfaces
  - Logic Quirks: “Trendy” Escapes
  - System-Calls: “Generic” Escapes
++

Thank You!

• Questions?