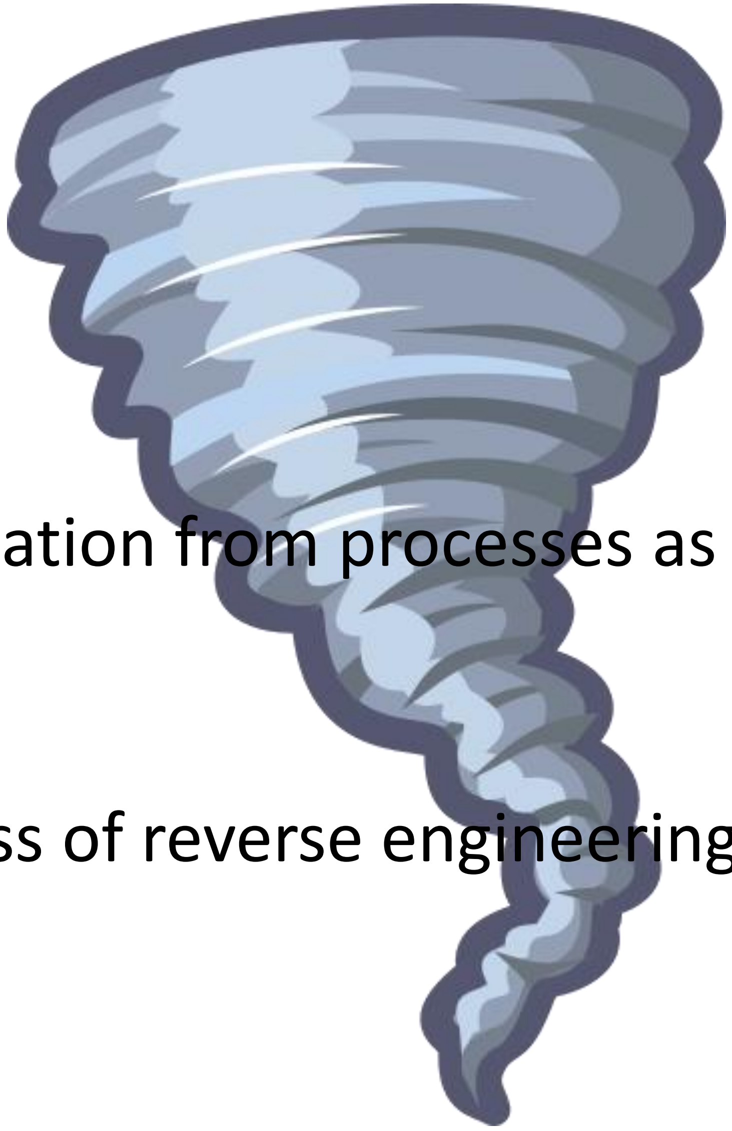
black hat USA 2016

Augmenting Static Analysis Using Pintool: Ablation

Paul Mehta, Senior Research Scientist

Inspiration for this talk

What is Ablation?



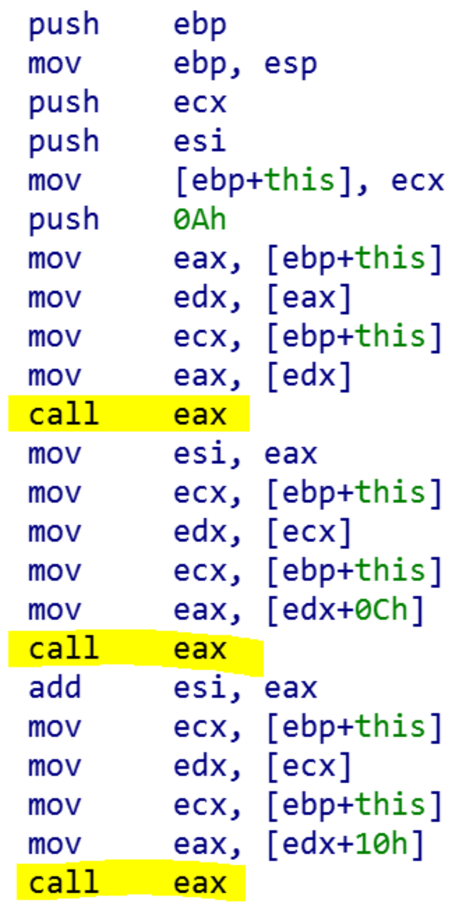
- Ablation is a tool that extracts information from processes as they execute.
- It was designed to simplify the process of reverse engineering

What is Ablation?

- The original intention was to jump-start code audits by automating the more tedious aspects of reverse engineering.
- It can also help find interesting code to audit.
- Portable & easy to use

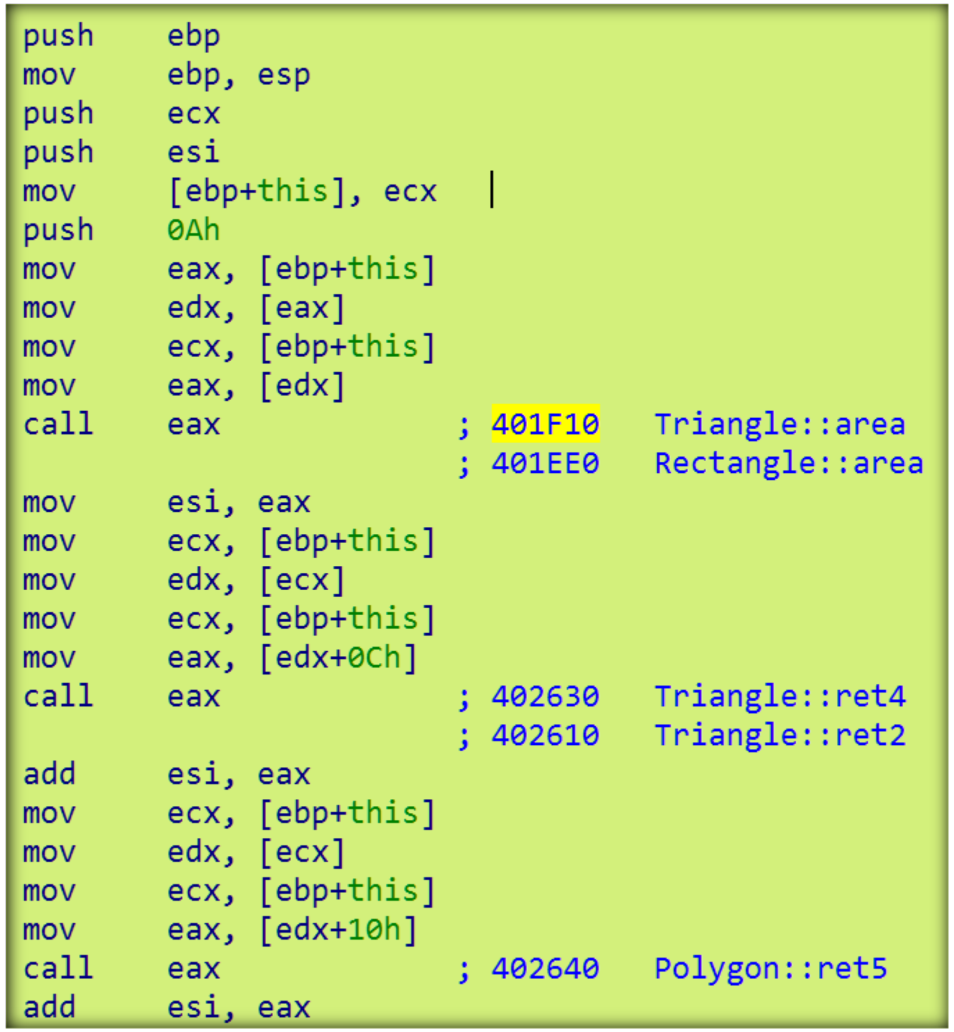
Where is it useful?

- Disassembling C++ often leaves newer reverse engineers feeling confused and overwhelmed.
- C++ binaries can be a pain to audit sometimes due to virtual calls.



```
push    ebp
mov     ebp, esp
push    ecx
push    esi
mov     [ebp+this], ecx
push    0Ah
mov     eax, [ebp+this]
mov     edx, [eax]
mov     ecx, [ebp+this]
mov     eax, [edx]
call    eax
mov     esi, eax
mov     ecx, [ebp+this]
mov     edx, [ecx]
mov     ecx, [ebp+this]
mov     eax, [edx+0Ch]
call    eax
add     esi, eax
mov     ecx, [ebp+this]
mov     edx, [ecx]
mov     ecx, [ebp+this]
mov     eax, [edx+10h]
call    eax
```

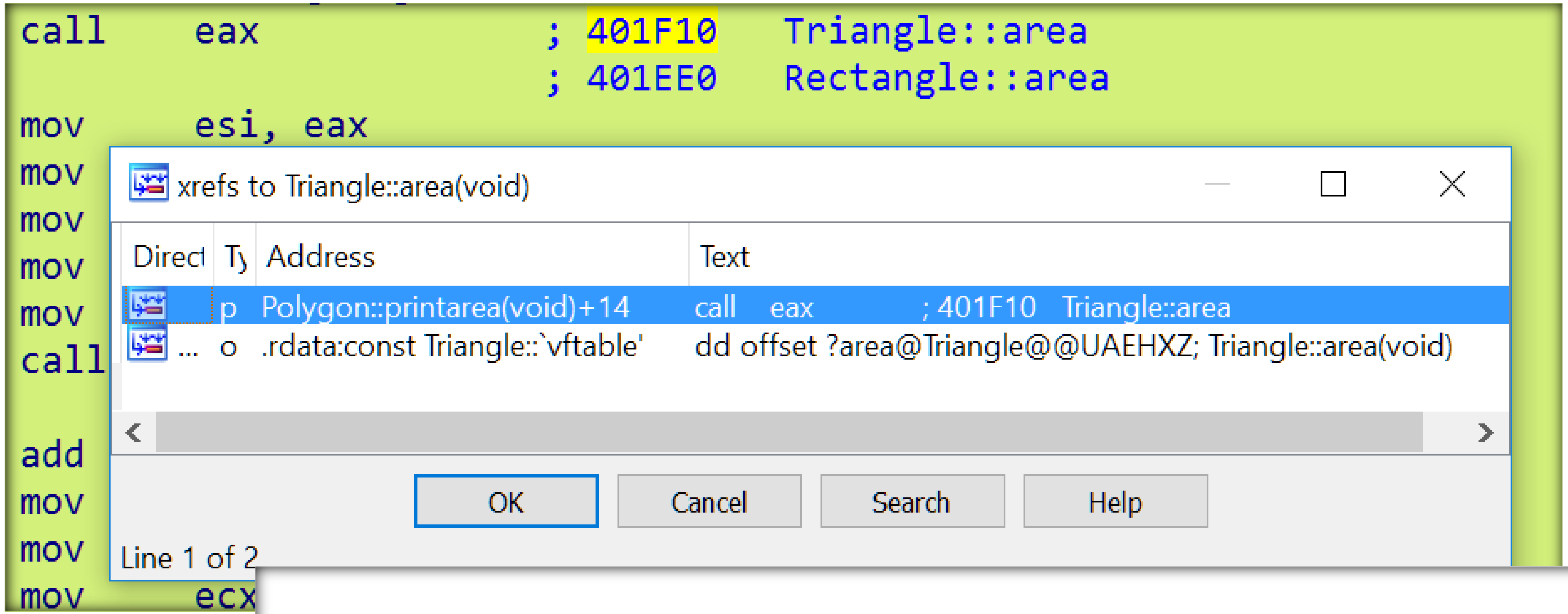
- Instead of having to reverse engineer C++ classes, and figure out inheritance relationships
- Ablation will resolve any observed virtual calls for you.



```
push    ebp
mov     ebp, esp
push    ecx
push    esi
mov     [ebp+this], ecx
push    0Ah
mov     eax, [ebp+this]
mov     edx, [eax]
mov     ecx, [ebp+this]
mov     eax, [edx]
call    eax                ; 401F10   Triangle::area
                           ; 401EE0   Rectangle::area
mov     esi, eax
mov     ecx, [ebp+this]
mov     edx, [ecx]
mov     ecx, [ebp+this]
mov     eax, [edx+0Ch]
call    eax                ; 402630   Triangle::ret4
                           ; 402610   Triangle::ret2
add     esi, eax
mov     ecx, [ebp+this]
mov     edx, [ecx]
mov     ecx, [ebp+this]
mov     eax, [edx+10h]
call    eax                ; 402640   Polygon::ret5
add     esi, eax
```

- Disassembled C++ reads like C!

- It also creates **fully interactive** x-refs in IDA



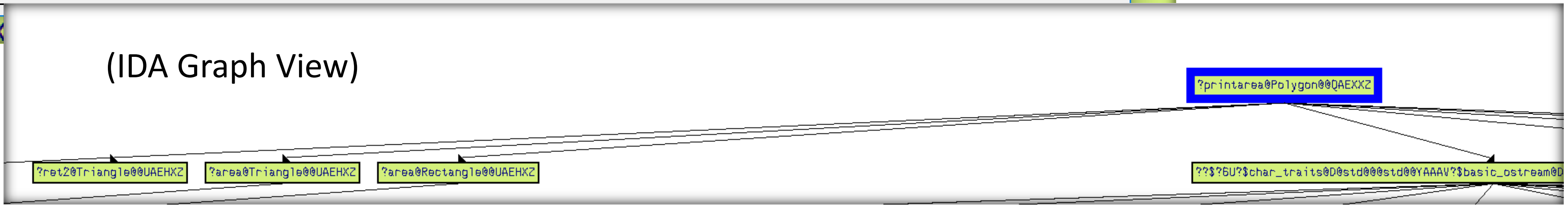
```
call    eax                 ; 401F10    Triangle::area
                            ; 401EE0    Rectangle::area
mov     esi, eax
mov
mov
mov
mov
call

add
mov
mov
mov     ecx
```

xrefs to Triangle::area(void)

Direct	Ty	Address	Text
	p	Polygon::printarea(void)+14	call eax ; 401F10 Triangle::area
...	o	.rdata:const Triangle::`vftable'	dd offset ?area@Triangle@@UAEHXZ; Triangle::area(void)

OK Cancel Search Help

Line 1 of 2


(IDA Graph View)
?printarea@Polygon@@QAEXXZ
?ret2@Triangle@@UAEHXZ
?area@Triangle@@UAEHXZ
?area@Rectangle@@UAEHXZ
??$?6U?$char_traits@D@std@@@std@@YAAAV?$basic_ostream@D

Using Ablation

- Launch
  - pin.exe -t Ablation.dll -module [modulename] -- application.exe
- Attach
  - pin.exe -pid [pid] -t Ablation.dll -module [modulename]
- Display help
  - Pin.exe -t Ablation.dll -h -- application.exe

Examples:

```
pin.exe -t Ablation.dll -module LibGLESv2 -verbose -- "c:\Program Files (x86)\Mozilla Firefox\firefox.exe" | AblationClientLite.exe LibGLESv2.ablation.py
pin.exe -pid 1234 -t Ablation.dll -module vgx
```

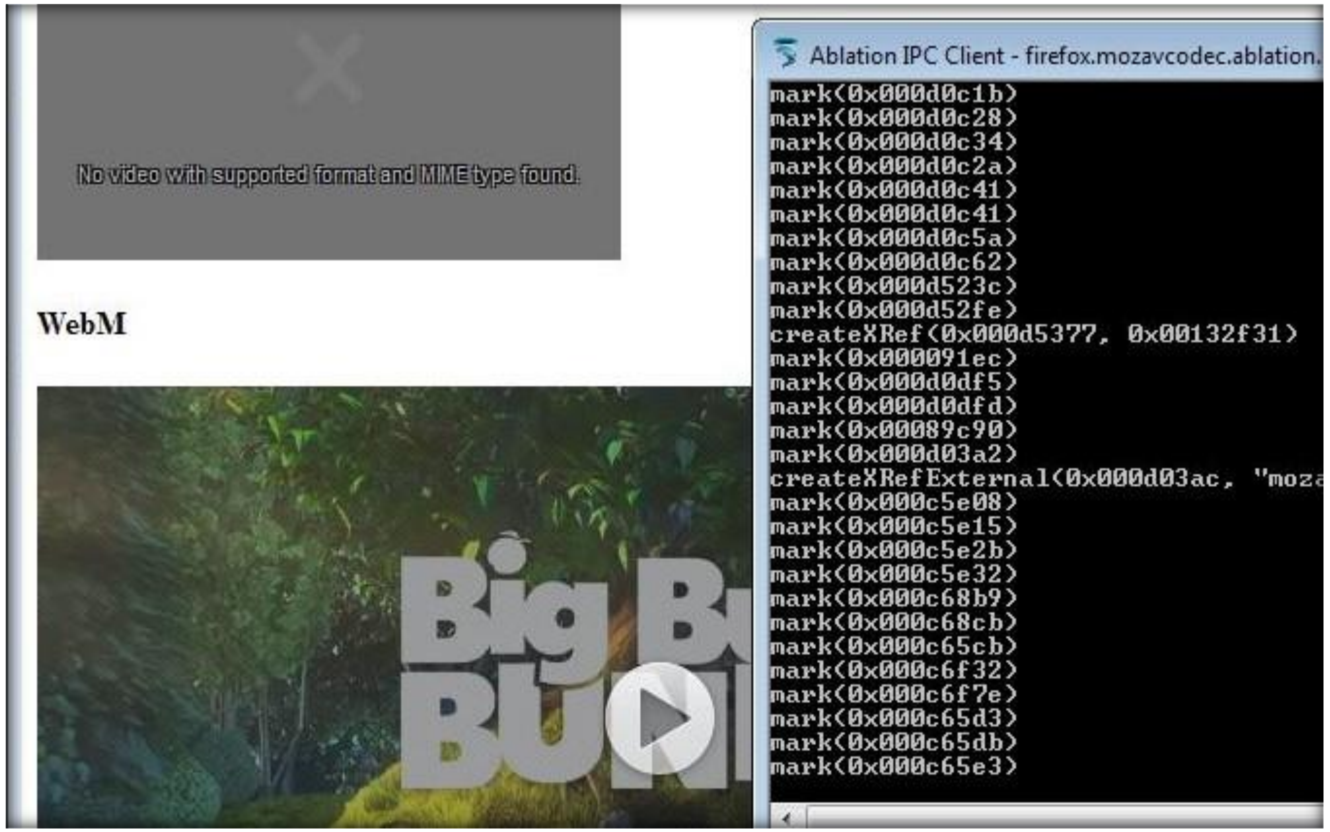
What information can we gather at runtime?

- Control flow data
- Execution frequency
- Resolve Virtual Calls
- Identify Interfaces

This information is then imported into the disassembly environment

Control flow data

- At runtime, record the first instance a basic block is executed
  - Only the first

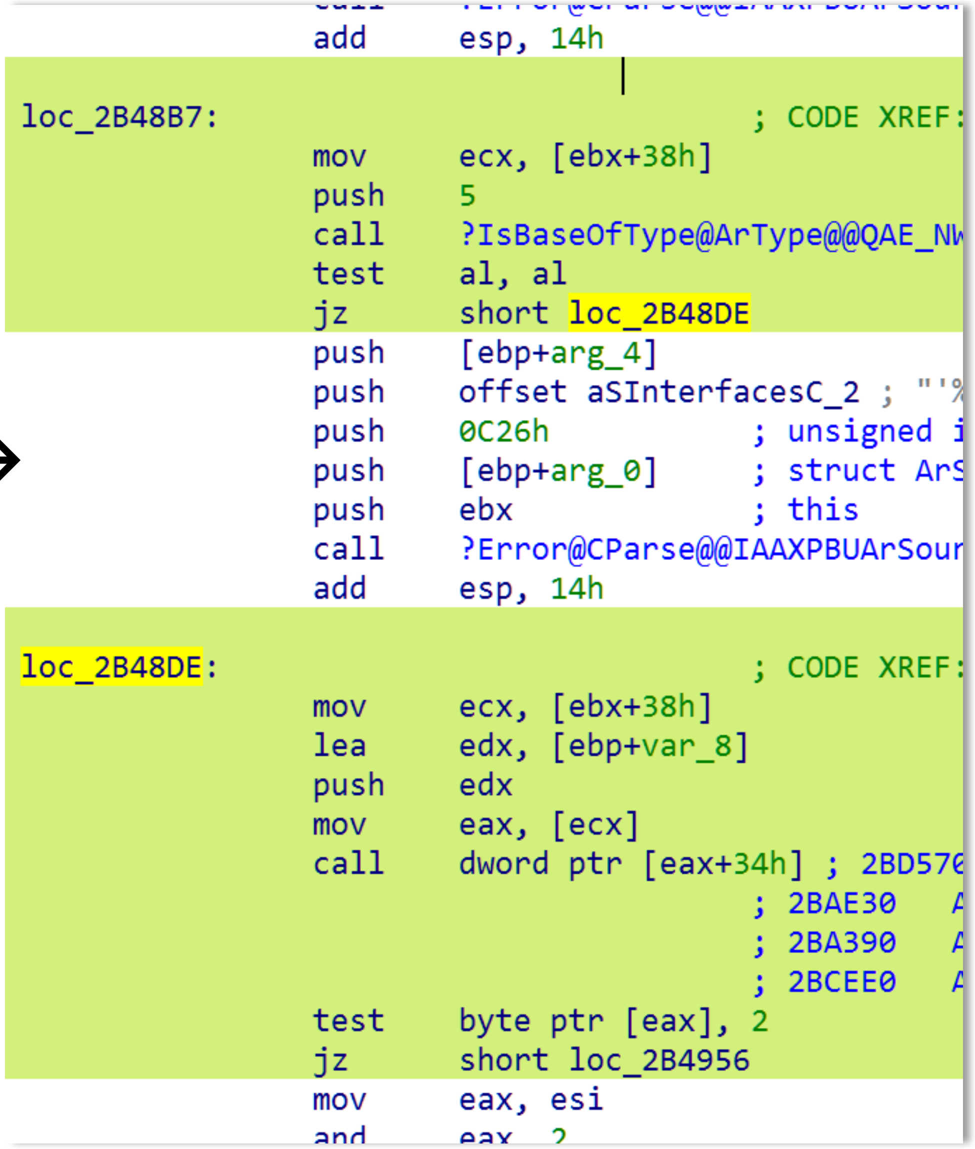


Ablation IPC Client - firefox.mozavcodec.ablation.

```
mark(0x000d0c1b)
mark(0x000d0c28)
mark(0x000d0c34)
mark(0x000d0c2a)
mark(0x000d0c41)
mark(0x000d0c41)
mark(0x000d0c5a)
mark(0x000d0c62)
mark(0x000d523c)
mark(0x000d52fe)
createXRef(0x000d5377, 0x00132f31)
mark(0x000091ec)
mark(0x000d0df5)
mark(0x000d0dfd)
mark(0x00089c90)
mark(0x000d03a2)
createXRefExternal(0x000d03ac, "moza
mark(0x000c5e08)
mark(0x000c5e15)
mark(0x000c5e2b)
mark(0x000c5e32)
mark(0x000c68b9)
mark(0x000c68cb)
mark(0x000c65cb)
mark(0x000c6f32)
mark(0x000c6f7e)
mark(0x000c65d3)
mark(0x000c65db)
mark(0x000c65e3)
```

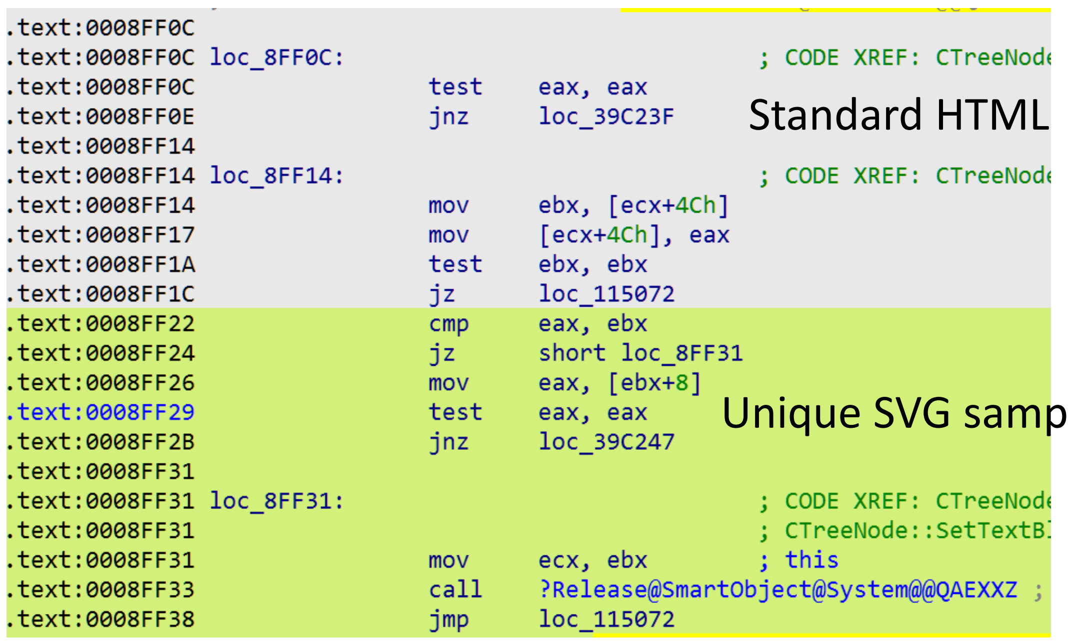
Once Imported

- Control flow data ends up looking like this →



```
                add     esp, 14h

loc_2B48B7:                                     ; CODE XREF:
                mov     ecx, [ebx+38h]
                push    5
                call    ?IsBaseOfType@ArType@@QAE_NW
                test    al, al
                jz      short loc_2B48DE
                push    [ebp+arg_4]
                push    offset aSInterfacesC_2 ; "'%
                push    0C26h           ; unsigned i
                push    [ebp+arg_0]     ; struct ArS
                push    ebx             ; this
                call    ?Error@CParse@@IAAXPBUArSour
                add     esp, 14h

loc_2B48DE:                                     ; CODE XREF:
                mov     ecx, [ebx+38h]
                lea     edx, [ebp+var_8]
                push    edx
                mov     eax, [ecx]
                call    dword ptr [eax+34h] ; 2BD570
                                        ; 2BAE30    A
                                        ; 2BA390    A
                                        ; 2BCEE0    A
                test    byte ptr [eax], 2
                jz      short loc_2B4956
                mov     eax, esi
```

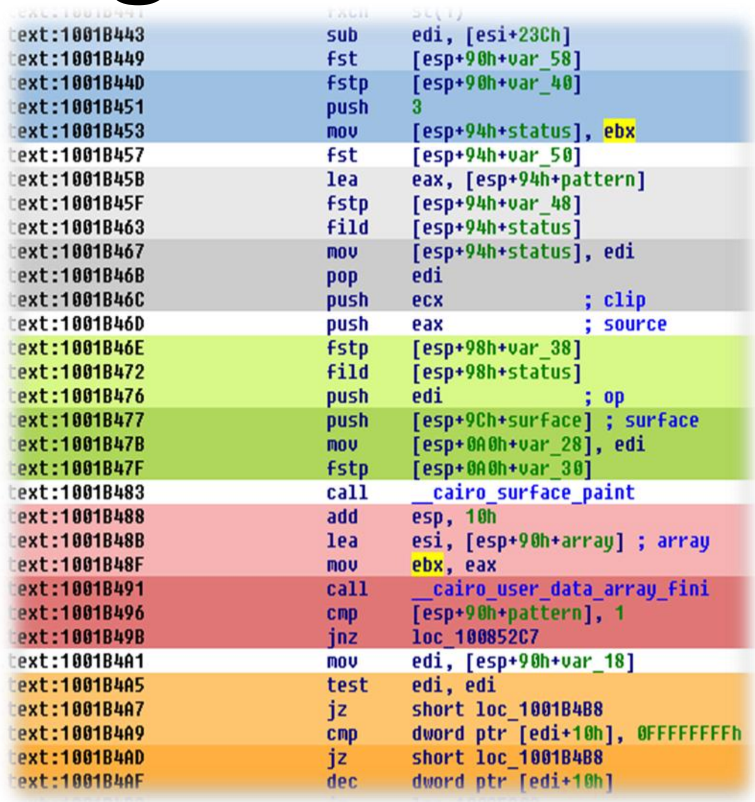
The Importance of choosing a good color!



```
.text:0008FF0C
.text:0008FF0C loc_8FF0C:                              ; CODE XREF: CTreeNode
.text:0008FF0C                 test    eax, eax
.text:0008FF0E                 jnz     loc_39C23F
.text:0008FF14
.text:0008FF14 loc_8FF14:                              ; CODE XREF: CTreeNode
.text:0008FF14                 mov     ebx, [ecx+4Ch]
.text:0008FF17                 mov     [ecx+4Ch], eax
.text:0008FF1A                 test    ebx, ebx
.text:0008FF1C                 jz      loc_115072
.text:0008FF22                 cmp     eax, ebx
.text:0008FF24                 jz      short loc_8FF31
.text:0008FF26                 mov     eax, [ebx+8]
.text:0008FF29                 test    eax, eax
.text:0008FF2B                 jnz     loc_39C247
.text:0008FF31
.text:0008FF31 loc_8FF31:                              ; CODE XREF: CTreeNode
.text:0008FF31                                         ; CTreeNode::SetTextB
.text:0008FF31                 mov     ecx, ebx        ; this
.text:0008FF33                 call    ?Release@SmartObject@System@@QAEXXZ ;
.text:0008FF38                 jmp     loc_115072
```

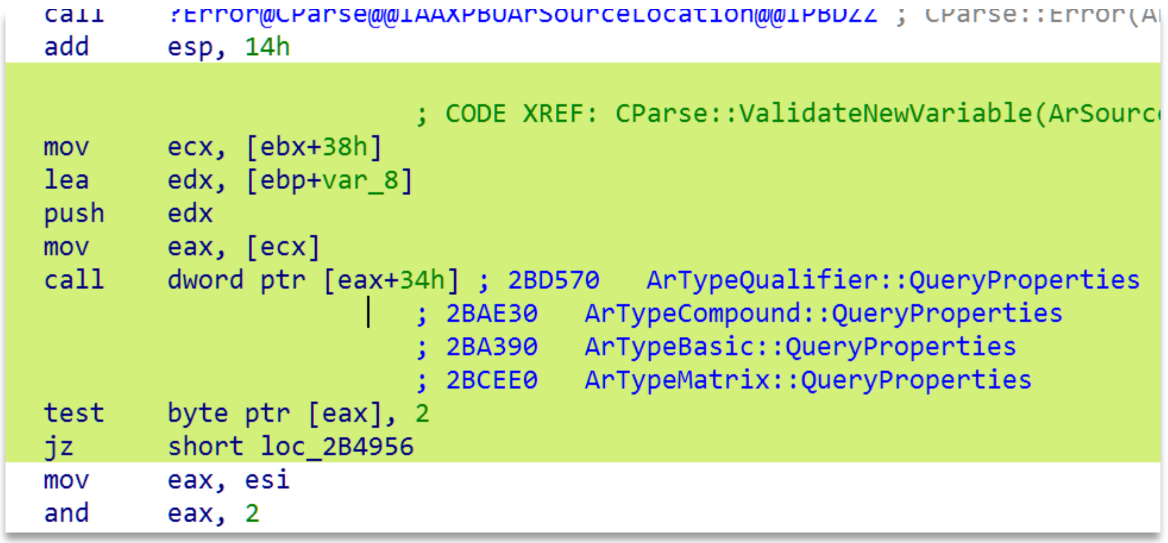
Standard HTML

Unique SVG sample



```
text:1001B443                 sub     edi, [esi+23Ch]
text:1001B449                 fst     [esp+90h+var_58]
text:1001B44D                 fstp    [esp+90h+var_40]
text:1001B451                 push    3
text:1001B453                 mov     [esp+94h+status], ebx
text:1001B457                 fst     [esp+94h+var_50]
text:1001B45B                 lea     eax, [esp+94h+pattern]
text:1001B45F                 fstp    [esp+94h+var_48]
text:1001B463                 fild    [esp+94h+status]
text:1001B467                 mov     [esp+94h+status], edi
text:1001B46B                 pop     edi
text:1001B46C                 push    ecx             ; clip
text:1001B46D                 push    eax             ; source
text:1001B46E                 fstp    [esp+98h+var_38]
text:1001B472                 fild    [esp+98h+status]
text:1001B476                 push    edi             ; op
text:1001B477                 push    [esp+9Ch+surface] ; surface
text:1001B47B                 mov     [esp+0A0h+var_28], edi
text:1001B47F                 fstp    [esp+0A0h+var_30]
text:1001B483                 call    __cairo_surface_paint
text:1001B488                 add     esp, 10h
text:1001B48B                 lea     esi, [esp+90h+array] ; array
text:1001B48F                 mov     ebx, eax
text:1001B491                 call    __cairo_user_data_array_fini
text:1001B496                 cmp     [esp+90h+pattern], 1
text:1001B49B                 jnz     loc_100852C7
text:1001B4A1                 mov     edi, [esp+90h+var_18]
text:1001B4A5                 test    edi, edi
text:1001B4A7                 jz      short loc_1001B4B8
text:1001B4A9                 cmp     dword ptr [edi+10h], 0FFFFFFFFh
text:1001B4AD                 jz      short loc_1001B4B8
text:1001B4AF                 dec     dword ptr [edi+10h]
```

Resolving Virtual Calls



```
call    ?Error@CParse@@IAAXPBUArSourceLocation@@IPBDZZ ; CParse::Error(A
add     esp, 14h

                                ; CODE XREF: CParse::ValidateNewVariable(ArSourc
mov     ecx, [ebx+38h]
lea     edx, [ebp+var_8]
push    edx
mov     eax, [ecx]
call    dword ptr [eax+34h] ; 2BD570   ArTypeQualifier::QueryProperties
                      |         ; 2BAE30   ArTypeCompound::QueryProperties
                                ; 2BA390   ArTypeBasic::QueryProperties
                                ; 2BCEE0   ArTypeMatrix::QueryProperties
test    byte ptr [eax], 2
jz      short loc_2B4956
mov     eax, esi
and     eax, 2
```

- For indirect call instructions
  - Maintain a list of observed call targets.
- During trace instrumentation, for each indirect call instruction,
  - If the target address has not been observed, add it to the list.

DEMO!

How Ablation augments static analysis

- Adds context by
  - Resolving virtual calls
    - Shows the various objects that are operating on data
  - If I never comment in another xref, it'll be too soon.

How Ablation augments static analysis

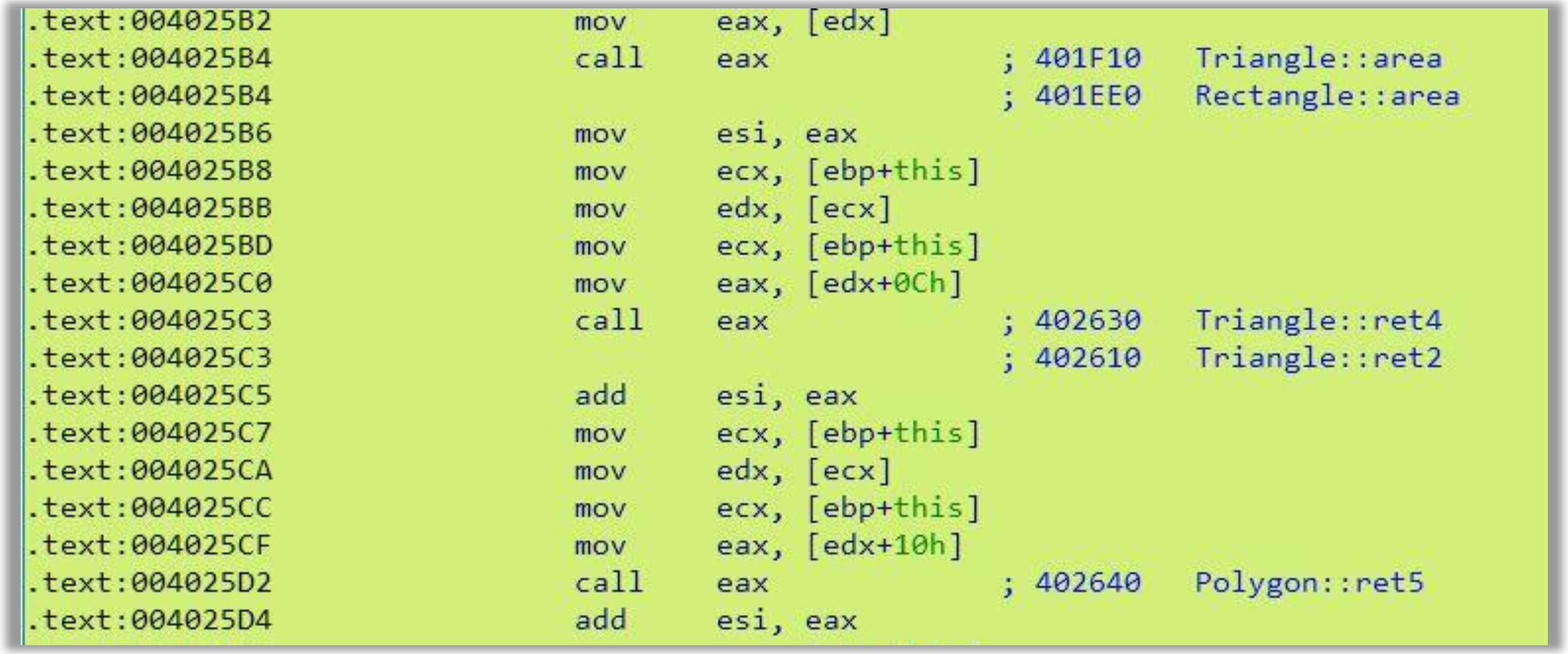
- Improves interactivity of the IDB
  - Imported data is displayed in x-ref lists and call graphs.
  - Being able to look at cross references for virtual calls is awesome!

How Ablation augments static analysis

- Adds *trace* info
  - Info is displayed to help the user understand the flow of data, code, and identify areas that may have been overlooked.

How Ablation augments static analysis

- Displays object inheritance info that can be inferred



```
.text:004025B2                 mov     eax, [edx]
.text:004025B4                 call    eax             ; 401F10   Triangle::area
.text:004025B4                                         ; 401EE0   Rectangle::area
.text:004025B6                 mov     esi, eax
.text:004025B8                 mov     ecx, [ebp+this]
.text:004025BB                 mov     edx, [ecx]
.text:004025BD                 mov     ecx, [ebp+this]
.text:004025C0                 mov     eax, [edx+0Ch]
.text:004025C3                 call    eax             ; 402630   Triangle::ret4
.text:004025C3                                         ; 402610   Triangle::ret2
.text:004025C5                 add     esi, eax
.text:004025C7                 mov     ecx, [ebp+this]
.text:004025CA                 mov     edx, [ecx]
.text:004025CC                 mov     ecx, [ebp+this]
.text:004025CF                 mov     eax, [edx+10h]
.text:004025D2                 call    eax             ; 402640   Polygon::ret5
.text:004025D4                 add     esi, eax
```

Observing Live Execution

- The rate at which new basic blocks execute decreases greatly after initialization
- This actually allows you to watch the process executing in real-time
  - This is because you're only seeing the first time a BBL is executed. After that, it's silent.

Observing Live Execution

- I like to use this feature to find samples that are worth looking at further.
- Run over a sample set, and then fiddle around till you see some new activity!
  - You may have found yourself a sample that does something rare.
    - If you're really lucky, it may be an undocumented feature!

DEMO!

ExSample (Execution Sample)

- An ExSample is the set of unique basic blocks executed by processing a sample
    - An ExSample can be thought of as a subset of all basic blocks in a module

- Highlights where related samples diverge
- Comparing ExSamples can be used to determine if samples traverse different execution paths (has a different effect)

Examples of how Ablation can be used

- Calculate code coverage for a sample set
- Highlight untested features
- Auto-Generate new sample input (yah right… but I actually did this a while back)
- Simple Crash Analysis

Simple crash analysis

- Use ablation to displaying the live process. Then load 3 samples.
  1. A null sample
     - To traverse the code executed when loading additional samples
       - Ie. File -> Open… etc.
  2. The parent sample that does not crash
  3. The sample that crashes.
- The basic blocks displayed after sample 2 are likely related to the crash.
  - This gets you to a solid place to begin understanding. You still have to figure it out.

Examples of how it can be used

- Determining code that processes specific data
    - Ex. SVG related code in Internet Explorer
        - Launch IE, and render some sites that do not contain SVG
        - Once Ablation stops/slows reporting the execution of new basic blocks
            - Render an SVG sample
                - The new burst of basic blocks will be SVG related

Examples of how it can be used

- **CTF!**
    - I've used Ablation to breeze through CTF problems
        - It really helps cut through the noise.

Examples of how it can be used

- Exploit analysis
  - Last weekend it was super helpful for analyzing a flash exploit
  - When I saw an indirect call resolve to Flash functionality, as well as KERNELBASE!VirtualProtect, mystery solved
    - Just debugging would definitely have taken longer

Ablative Fuzzing

Fuzzing what not to audit

To audit or to fuzz?

- Why do people audit?
  - It's how you find the good stuff.

- Why do people fuzz?
  - A computer can work while you sleep.

Researchers tend to have a preference for one or another.

Bug Hunting

- Try to hide a bug in 1 slide; good luck! In 100 slides?

- Let's say a fuzzer went at it, and didn't find the bug.
  - Using Ablation you can see what slides the fuzzer thoroughly tested.

  - Doesn't it make sense to start auditing the slides it missed first?

Auditing interesting code is more fun!

- Highly traversed code is not particularly interesting
  - Ex. The set of basic blocks used to render a SVG Circle element is **shared** across many samples.
    - Highly traversed
    - Not particularly interesting
- Infrequently executed code is more interesting to audit
  - Ex. Rendering a SVG Animated Font

Auditing interesting code is more fun!

- Highly traversed code is not particularly interesting
  - Ex. The set of basic blocks used to render a SVG Circle element is **shared** across many samples.
    - Highly traversed
    - Not particularly interesting
- Infrequently executed code is more interesting to audit
  - Ex. Rendering a SVG Animated Font
- By diffing 2 the ExSamples, you can identify the code related to SVG Animated Fonts
  - And exclude the code shared by common samples

Baseline Sample Set

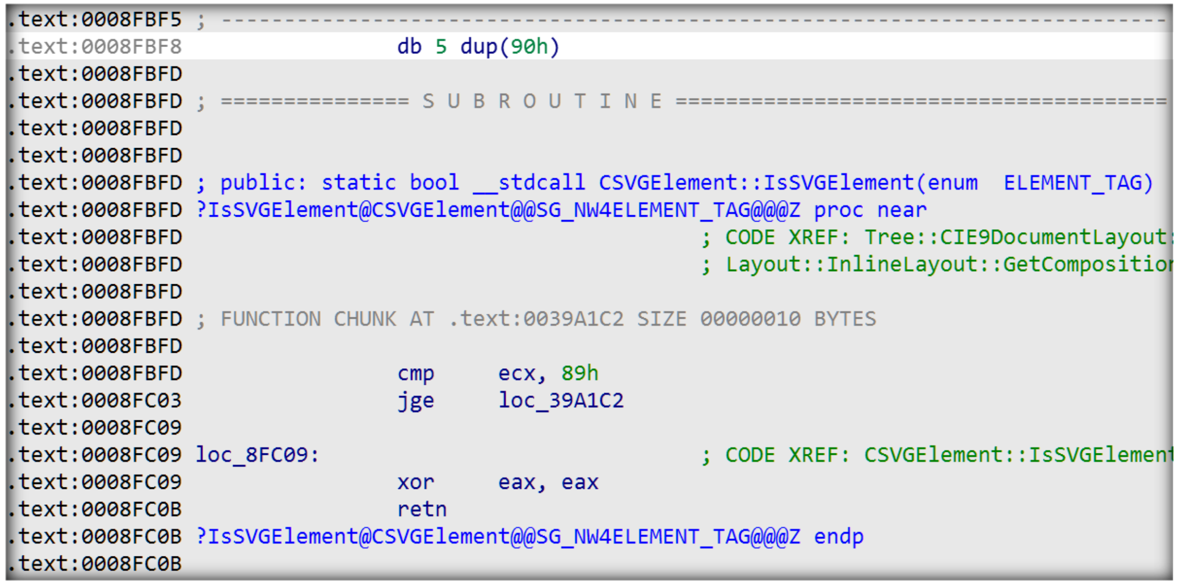
- If you use Ablation to traverse a sample set, that will give you a baseline
  - This is the union of basic blocks executed across the entire set of common samples.
    - “Common samples” refers to *W3C_SVG_12_TinyTestSuite* or similar conformance test suites used during development.

Baseline Fuzzing

- If you proceed to fuzz that sample set, you're moving off the baseline
  - Fuzzing is going to heavily traverse error handling code
  - Lots of conditionals prior to unexplored code
- Depending on your fuzzer, infrastructure, input format, this may work great
  - But, it's more likely that it **looked** great initially
- You’re going to hit new basic blocks that weren’t executed by the baseline samples

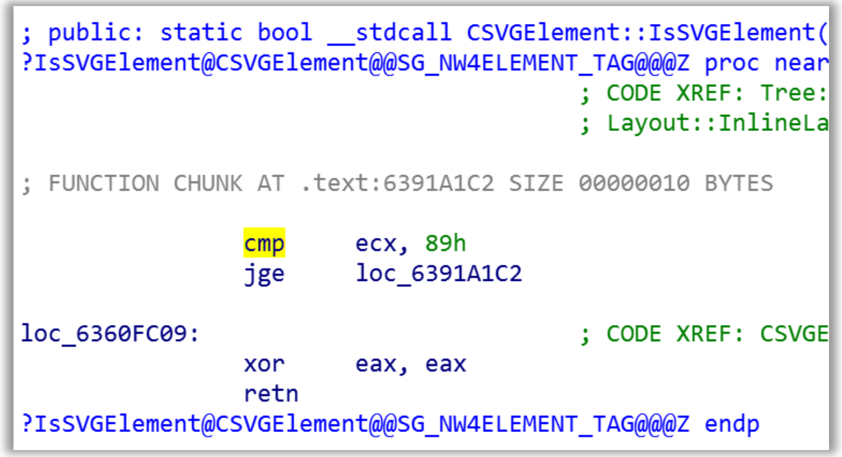
Using Ablation to Show Initialization

- Code executed on startup, or loading "*Hello World*" samples
  - We're going to show this highly traversed code as grey



```
.text:0008FBF5 ; ---------------------------------------------------------------------------
.text:0008FBF8                 db 5 dup(90h)
.text:0008FBFD
.text:0008FBFD ; =============== S U B R O U T I N E =======================================
.text:0008FBFD
.text:0008FBFD
.text:0008FBFD ; public: static bool __stdcall CSVGElement::IsSVGElement(enum  ELEMENT_TAG)
.text:0008FBFD ?IsSVGElement@CSVGElement@@SG_NW4ELEMENT_TAG@@@Z proc near
.text:0008FBFD                                         ; CODE XREF: Tree::CIE9DocumentLayout
.text:0008FBFD                                         ; Layout::InlineLayout::GetCompositio
.text:0008FBFD
.text:0008FBFD ; FUNCTION CHUNK AT .text:0039A1C2 SIZE 00000010 BYTES
.text:0008FBFD
.text:0008FBFD                 cmp     ecx, 89h
.text:0008FC03                 jge     loc_39A1C2
.text:0008FC09
.text:0008FC09 loc_8FC09:                              ; CODE XREF: CSVGElement::IsSVGElemen
.text:0008FC09                 xor     eax, eax
.text:0008FC0B                 retn
.text:0008FC0B ?IsSVGElement@CSVGElement@@SG_NW4ELEMENT_TAG@@@Z endp
.text:0008FC0B
```

Ablation



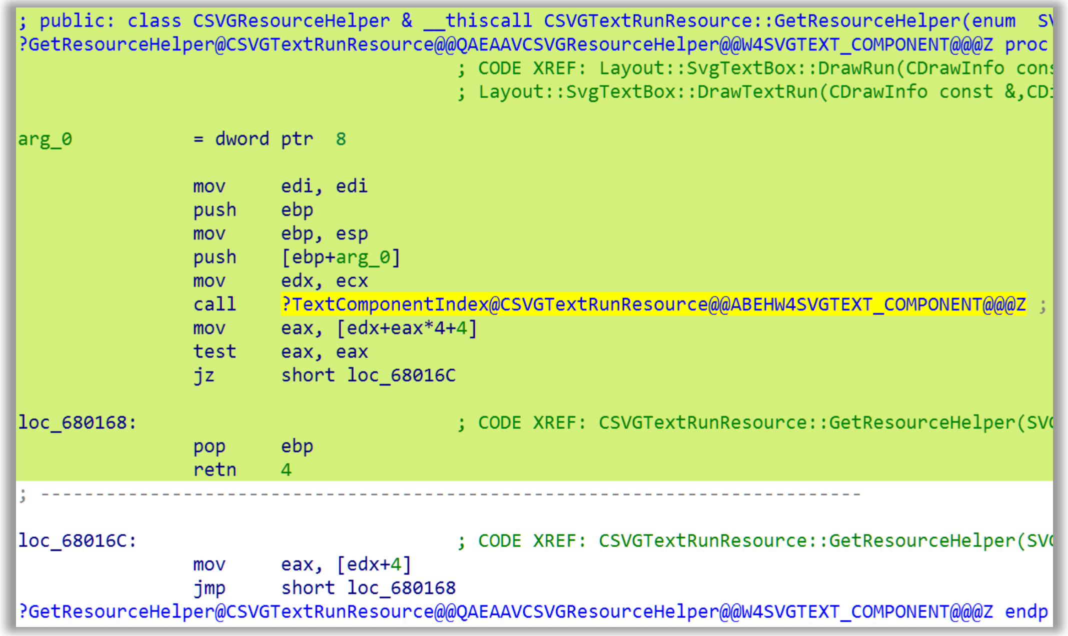
```
; public: static bool __stdcall CSVGElement::IsSVGElement(
?IsSVGElement@CSVGElement@@SG_NW4ELEMENT_TAG@@@Z proc near
                                        ; CODE XREF: Tree:
                                        ; Layout::InlineLa

; FUNCTION CHUNK AT .text:6391A1C2 SIZE 00000010 BYTES

                cmp     ecx, 89h
                jge     loc_6391A1C2

loc_6360FC09:                           ; CODE XREF: CSVGE
                xor     eax, eax
                retn
?IsSVGElement@CSVGElement@@SG_NW4ELEMENT_TAG@@@Z endp
```

Default

CSVGElement::IsSVGElement(enum ELEMENT_TAG)

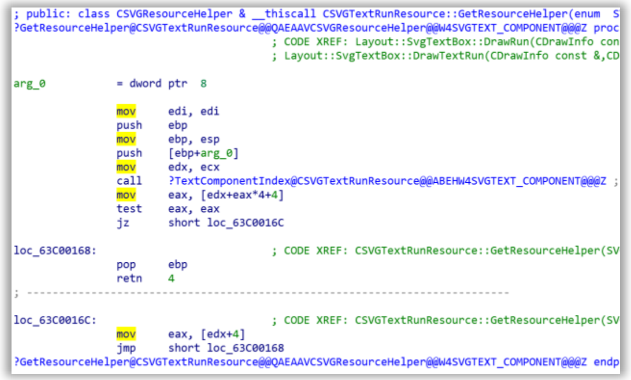
Using Ablation to Show the Baseline

- Code executed by iterating over the samples in a Test Suite (Ex. W3C_SVG_12_TinyTestSuite)
  - We're going to show this light **green**



```
; public: class CSVGResourceHelper & __thiscall CSVGTextRunResource::GetResourceHelper(enum  SV
?GetResourceHelper@CSVGTextRunResource@@QAEAAVCSVGResourceHelper@@W4SVGTEXT_COMPONENT@@@Z proc
                                        ; CODE XREF: Layout::SvgTextBox::DrawRun(CDrawInfo cons
                                        ; Layout::SvgTextBox::DrawTextRun(CDrawInfo const &,CD

arg_0           = dword ptr  8

                mov     edi, edi
                push    ebp
                mov     ebp, esp
                push    [ebp+arg_0]
                mov     edx, ecx
                call    ?TextComponentIndex@CSVGTextRunResource@@ABEHW4SVGTEXT_COMPONENT@@@Z ;
                mov     eax, [edx+eax*4+4]
                test    eax, eax
                jz      short loc_68016C

loc_680168:                             ; CODE XREF: CSVGTextRunResource::GetResourceHelper(SV
                pop     ebp
                retn    4
; ---------------------------------------------------------------------------

loc_68016C:                             ; CODE XREF: CSVGTextRunResource::GetResourceHelper(SV
                mov     eax, [edx+4]
                jmp     short loc_680168
?GetResourceHelper@CSVGTextRunResource@@QAEAAVCSVGResourceHelper@@W4SVGTEXT_COMPONENT@@@Z endp
```

Ablation



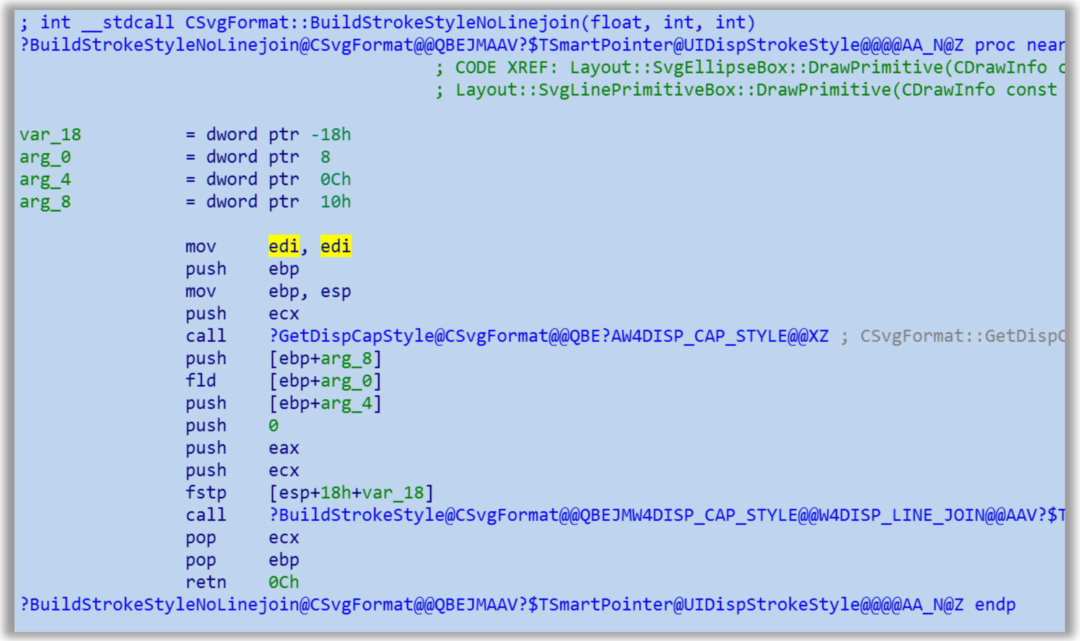
```
; public: class CSVGResourceHelper & __thiscall CSVGTextRunResource::GetResourceHelper(enum  S
?GetResourceHelper@CSVGTextRunResource@@QAEAAVCSVGResourceHelper@@W4SVGTEXT_COMPONENT@@@Z proc
                                        ; CODE XREF: Layout::SvgTextBox::DrawRun(CDrawInfo con
                                        ; Layout::SvgTextBox::DrawTextRun(CDrawInfo const &,CD

arg_0           = dword ptr  8

                mov     edi, edi
                push    ebp
                mov     ebp, esp
                push    [ebp+arg_0]
                mov     edx, ecx
                call    ?TextComponentIndex@CSVGTextRunResource@@ABEHW4SVGTEXT_COMPONENT@@@Z ;
                mov     eax, [edx+eax*4+4]
                test    eax, eax
                jz      short loc_63C0016C

loc_63C00168:                           ; CODE XREF: CSVGTextRunResource::GetResourceHelper(SV
                pop     ebp
                retn    4
; ---------------------------------------------------------------------------

loc_63C0016C:                           ; CODE XREF: CSVGTextRunResource::GetResourceHelper(SV
                mov     eax, [edx+4]
                jmp     short loc_63C00168
?GetResourceHelper@CSVGTextRunResource@@QAEAAVCSVGResourceHelper@@W4SVGTEXT_COMPONENT@@@Z endp
```

Default

CSVGTextRunResource::GetResourceHelper(enum SVGTEXT_COMPONENT)

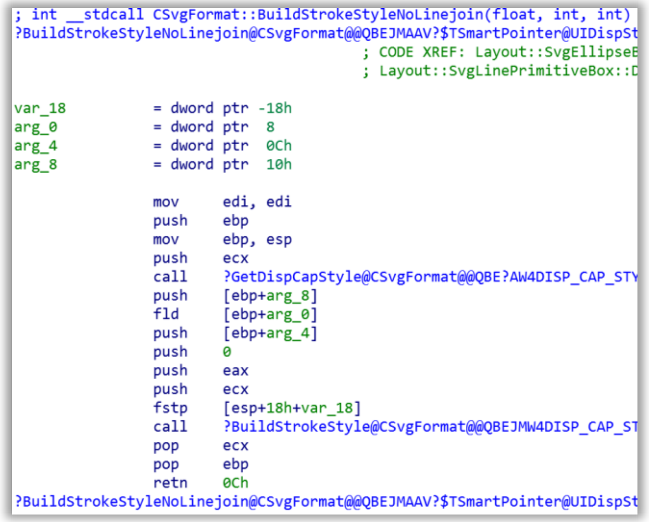
Using Ablation to Show the Mutated Baseline

- Code executed by mutating samples
  - We’re going to show this light Blue



```
; int __stdcall CSvgFormat::BuildStrokeStyleNoLinejoin(float, int, int)
?BuildStrokeStyleNoLinejoin@CSvgFormat@@QBEJMAAV?$TSmartPointer@UIDispStrokeStyle@@@@AA_N@Z proc near
                                        ; CODE XREF: Layout::SvgEllipseBox::DrawPrimitive(CDrawInfo c
                                        ; Layout::SvgLinePrimitiveBox::DrawPrimitive(CDrawInfo const

var_18          = dword ptr -18h
arg_0           = dword ptr  8
arg_4           = dword ptr  0Ch
arg_8           = dword ptr  10h

                mov     edi, edi
                push    ebp
                mov     ebp, esp
                push    ecx
                call    ?GetDispCapStyle@CSvgFormat@@QBE?AW4DISP_CAP_STYLE@@XZ ; CSvgFormat::GetDispC
                push    [ebp+arg_8]
                fld     [ebp+arg_0]
                push    [ebp+arg_4]
                push    0
                push    eax
                push    ecx
                fstp    [esp+18h+var_18]
                call    ?BuildStrokeStyle@CSvgFormat@@QBEJMW4DISP_CAP_STYLE@@W4DISP_LINE_JOIN@@AAV?$T
                pop     ecx
                pop     ebp
                retn    0Ch
?BuildStrokeStyleNoLinejoin@CSvgFormat@@QBEJMAAV?$TSmartPointer@UIDispStrokeStyle@@@@AA_N@Z endp
```

Ablation



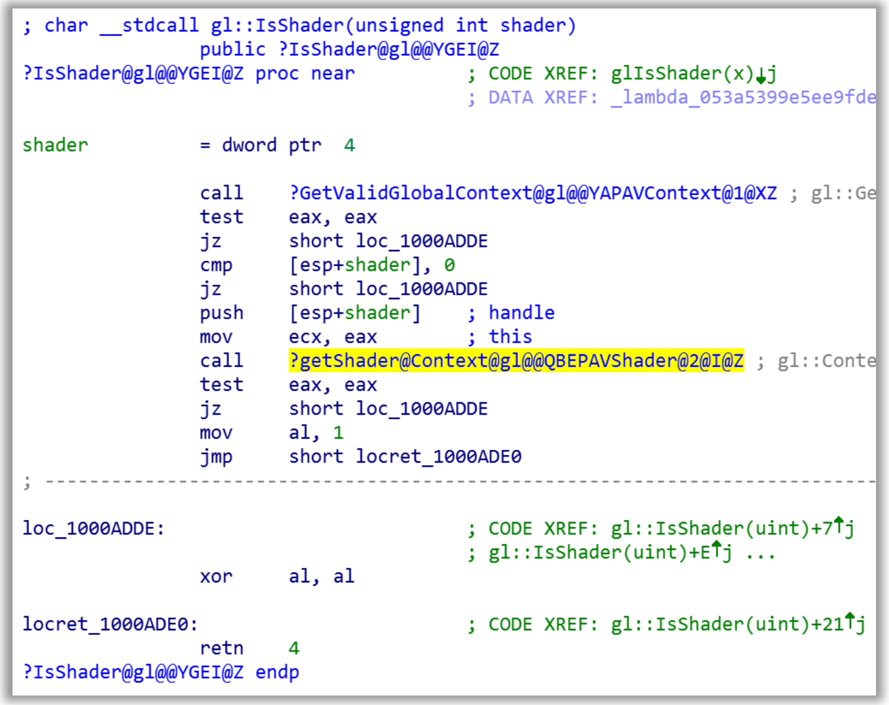
```
; int __stdcall CSvgFormat::BuildStrokeStyleNoLinejoin(float, int, int)
?BuildStrokeStyleNoLinejoin@CSvgFormat@@QBEJMAAV?$TSmartPointer@UIDispSt
                                        ; CODE XREF: Layout::SvgEllipseB
                                        ; Layout::SvgLinePrimitiveBox::D

var_18          = dword ptr -18h
arg_0           = dword ptr  8
arg_4           = dword ptr  0Ch
arg_8           = dword ptr  10h

                mov     edi, edi
                push    ebp
                mov     ebp, esp
                push    ecx
                call    ?GetDispCapStyle@CSvgFormat@@QBE?AW4DISP_CAP_STY
                push    [ebp+arg_8]
                fld     [ebp+arg_0]
                push    [ebp+arg_4]
                push    0
                push    eax
                push    ecx
                fstp    [esp+18h+var_18]
                call    ?BuildStrokeStyle@CSvgFormat@@QBEJMW4DISP_CAP_ST
                pop     ecx
                pop     ebp
                retn    0Ch
?BuildStrokeStyleNoLinejoin@CSvgFormat@@QBEJMAAV?$TSmartPointer@UIDispSt
```

Default

CSvgFormat::BuildStrokeStyleNoLinejoin(float, int, int)

Using Ablation to Show Not Executed

- Everything else
  - We’re going to show this as default (no color)

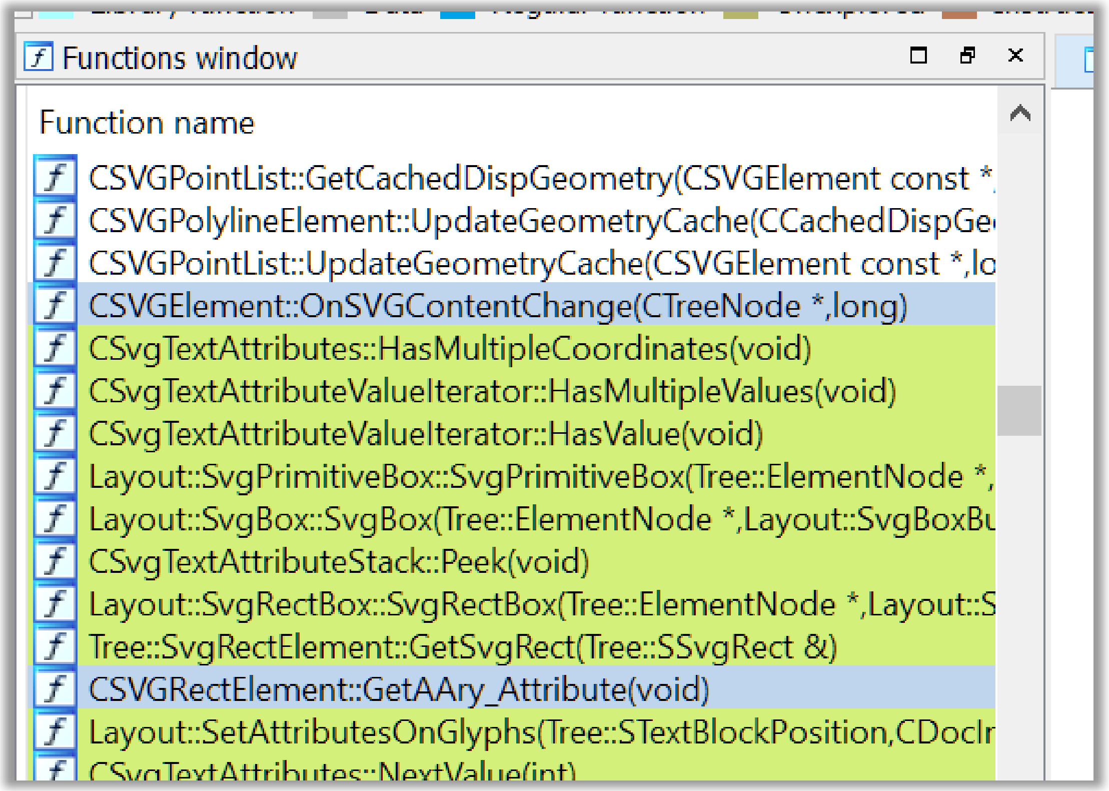


```
; char __stdcall gl::IsShader(unsigned int shader)
                public ?IsShader@gl@@YGEI@Z
?IsShader@gl@@YGEI@Z proc near          ; CODE XREF: glIsShader(x)↓j
                                        ; DATA XREF: _lambda_053a5399e5ee9fde

shader          = dword ptr  4

                call    ?GetValidGlobalContext@gl@@YAPAVContext@1@XZ ; gl::Ge
                test    eax, eax
                jz      short loc_1000ADDE
                cmp     [esp+shader], 0
                jz      short loc_1000ADDE
                push    [esp+shader]    ; handle
                mov     ecx, eax        ; this
                call    ?getShader@Context@gl@@QBEPAVShader@2@I@Z ; gl::Conte
                test    eax, eax
                jz      short loc_1000ADDE
                mov     al, 1
                jmp     short locret_1000ADE0
; ---------------------------------------------------------------------------

loc_1000ADDE:                           ; CODE XREF: gl::IsShader(uint)+7↑j
                                        ; gl::IsShader(uint)+E↑j ...
                xor     al, al

locret_1000ADE0:                        ; CODE XREF: gl::IsShader(uint)+21↑j
                retn    4
?IsShader@gl@@YGEI@Z endp
```

Not Executed

Functions Window

- In the functions list, items are colored as the first basic block is

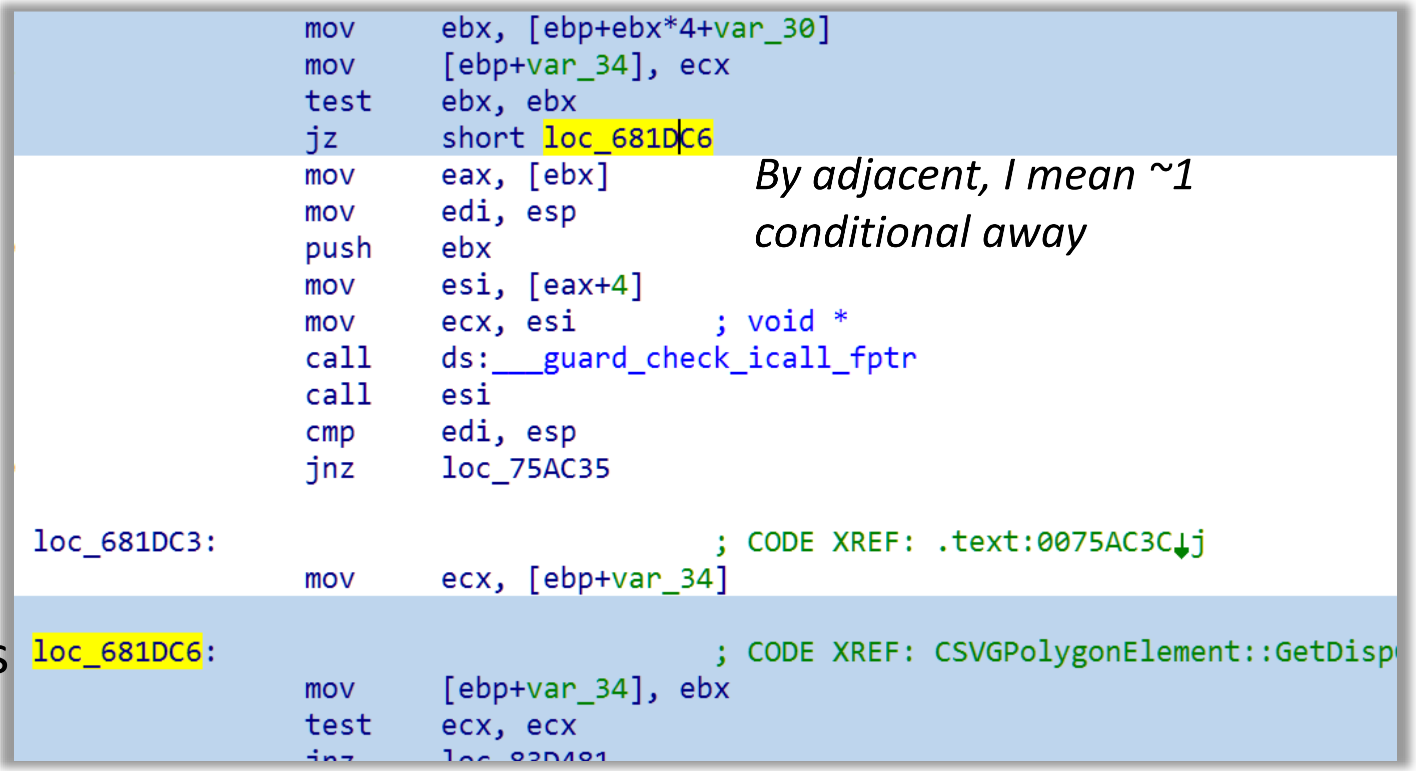

Functions window

Function name

CSVGPointList::GetCachedDispGeometry(CSVGElement const *,
CSVGPolylineElement::UpdateGeometryCache(CCachedDispGe
CSVGPointList::UpdateGeometryCache(CSVGElement const *,lo
CSVGElement::OnSVGContentChange(CTreeNode *,long)
CSvgTextAttributes::HasMultipleCoordinates(void)
CSvgTextAttributeValueIterator::HasMultipleValues(void)
CSvgTextAttributeValueIterator::HasValue(void)
Layout::SvgPrimitiveBox::SvgPrimitiveBox(Tree::ElementNode *,
Layout::SvgBox::SvgBox(Tree::ElementNode *,Layout::SvgBoxBu
CSvgTextAttributeStack::Peek(void)
Layout::SvgRectBox::SvgRectBox(Tree::ElementNode *,Layout::S
Tree::SvgRectElement::GetSvgRect(Tree::SSvgRect &)
CSVGRectElement::GetAAry_Attribute(void)
Layout::SetAttributesOnGlyphs(Tree::STextBlockPosition,CDocIr
CSvgTextAttributes::NextValue(int)

Adjacent Code is more interesting

- Looking at code that is adjacent to the baseline will highlight areas more likely to be
  - Less scrutinized, untested, undocumented, or experimental
- The SVG test suite didn’t hit this code.
- Looks like it could interesting function to audit.
- If you're looking at C++ classes, this includes uncolored methods that belong to colored classes.



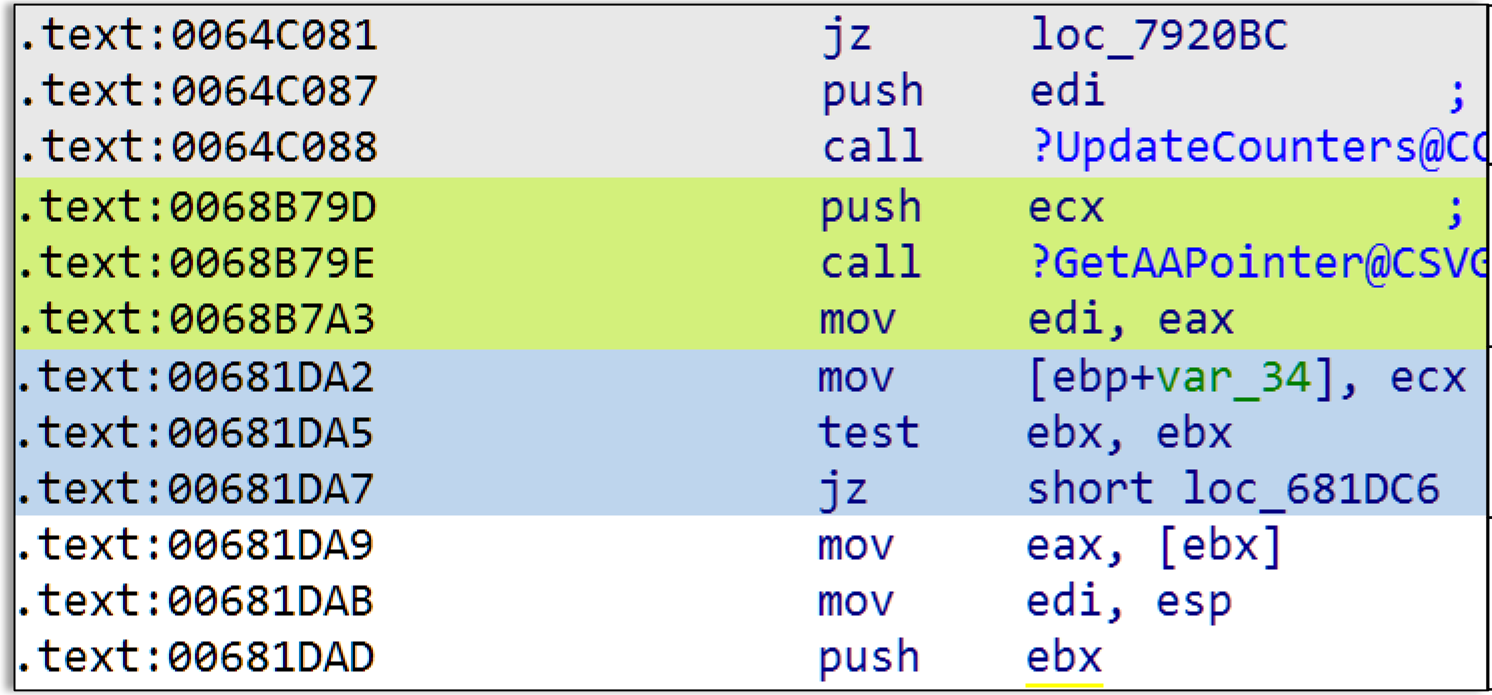
```
                mov     ebx, [ebp+ebx*4+var_30]
                mov     [ebp+var_34], ecx
                test    ebx, ebx
                jz      short loc_681DC6
                mov     eax, [ebx]
                mov     edi, esp
                push    ebx
                mov     esi, [eax+4]
                mov     ecx, esi        ; void *
                call    ds:___guard_check_icall_fptr
                call    esi
                cmp     edi, esp
                jnz     loc_75AC35

loc_681DC3:                             ; CODE XREF: .text:0075AC3C↓j
                mov     ecx, [ebp+var_34]

loc_681DC6:                             ; CODE XREF: CSVGPolygonElement::GetDisp
                mov     [ebp+var_34], ebx
                test    ecx, ecx
                jnz     loc_83D481
```

By adjacent, I mean ~1 conditional away

Ablative Fuzzing Overview

- Fuzzing to
  - Excluding code from an audit
  - Highlight adjacent code to include in an audit

Preferably a Genetic fuzzing algorithm, using the # of new basic blocks executed as a heuristic



```
.text:0064C081                jz      loc_7920BC
.text:0064C087                push    edi                   ;
.text:0064C088                call    ?UpdateCounters@CC
.text:0068B79D                push    ecx                   ;
.text:0068B79E                call    ?GetAAPointer@CSVG
.text:0068B7A3                mov     edi, eax
.text:00681DA2                mov     [ebp+var_34], ecx
.text:00681DA5                test    ebx, ebx
.text:00681DA7                jz      short loc_681DC6
.text:00681DA9                mov     eax, [ebx]
.text:00681DAB                mov     edi, esp
.text:00681DAD                push    ebx
```

Highly traversed code *Not of interest*
Sample Set *Moderate interest*
Mutated Samples *More interesting*
Not Executed *Interesting in relation*

What to take away?

- It’s easy to use.
- It’s portable.
- It can save a lot of time

- It doesn’t do the interesting work for you, but it may help you get there ;)

Questions?

Thank you