If It Ain't Broke, Don't Fix It:
Challenges and New Directions for Inferring the Impact of Software Patches

Jon Oberheide, Evan Cooke, Farnam Jahanian

University of Michigan

May 19th, 2009

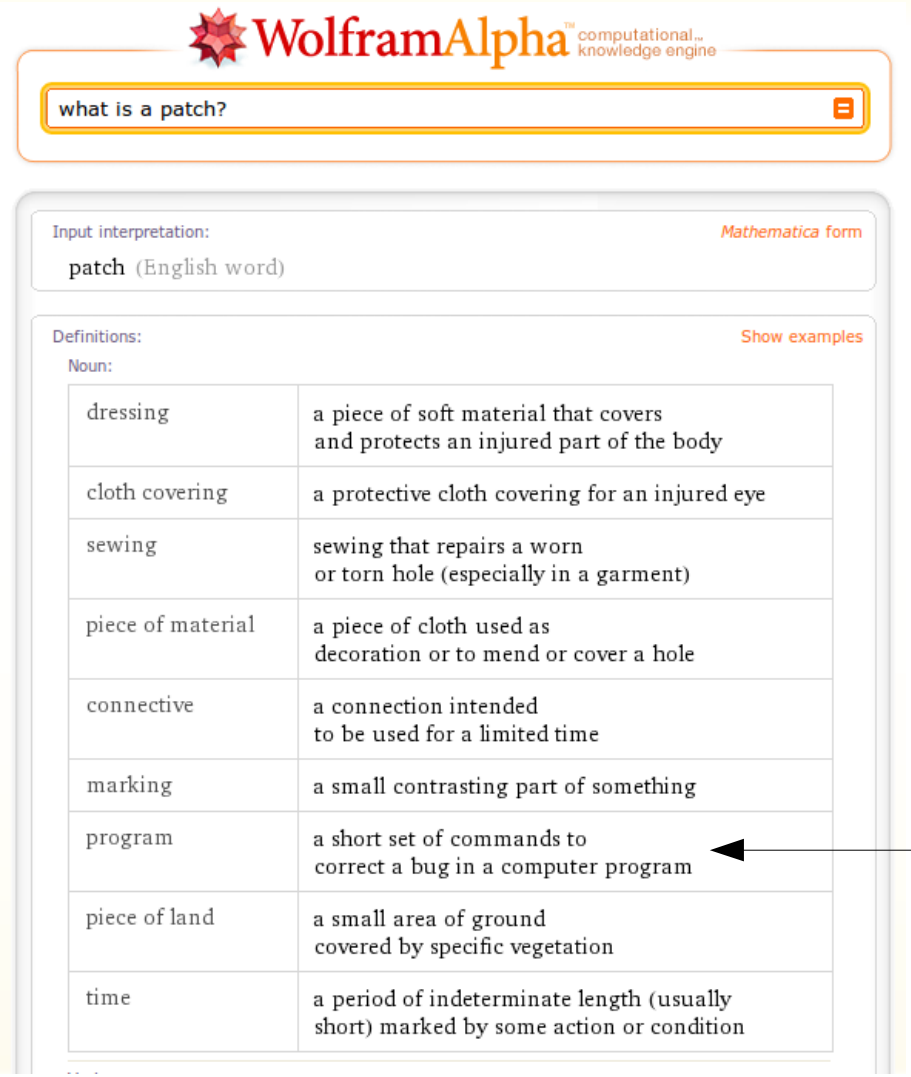
HotOS XII

Outrageous Claim

We can automatically infer the impact a patch will have on a software system.*

* DISCLAIMER: Prices and participation may vary. Additional charge for extra meat or cheese. Offer not valid in Switzerland.

What is a Patch?

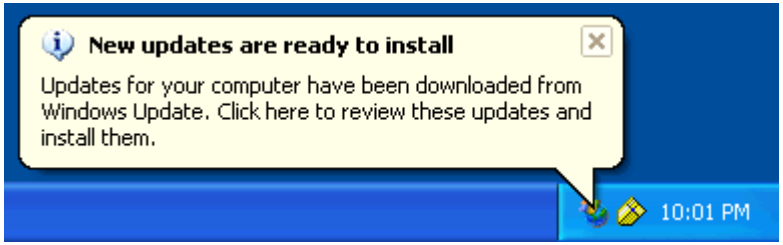


patch:

A short set of commands to correct a bug in a computer program

Different Interpretations

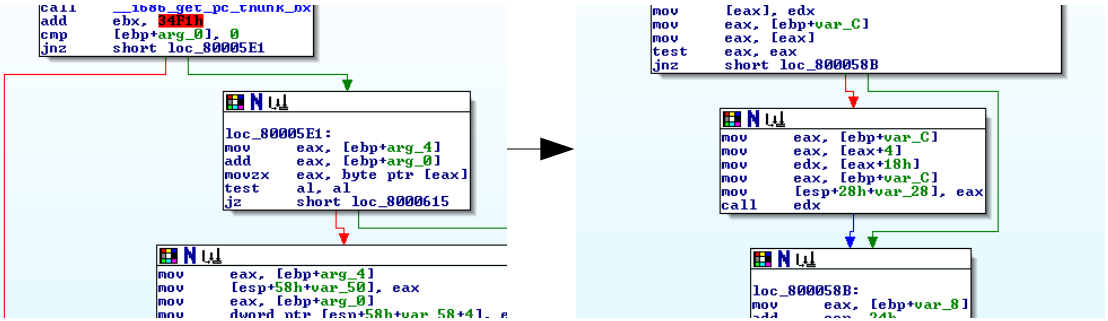
To a user:



To a developer:

```
diff --git a/fs/cifs/connect.c b/fs/cifs/connect.c
index 01e280c..1a93604 100644 (file)
--- a/fs/cifs/connect.c
+++ b/fs/cifs/connect.c
@@ -3756,16 +3756,13 @@ CIFSTCon(unsigned int xid, struct cifsSesInfo *ses,
                                BCC(smb_buffer_response)) {
                        kfree(tcon->nativeFileSystem);
                        tcon->nativeFileSystem =
-                               kzalloc(2*(length + 1), GFP_KERNEL);
+                               kzalloc((4 * length) + 2, GFP_KERNEL);
                        if (tcon->nativeFileSystem)
```

To a machine:



Our Target Audience

- System administrators
  - Intelligence: more than user, less than developer
- Tasked with patching complex systems
  - Software has bugs, software needs patching
  - Admin determines whether to apply patch
- Problems
  - Limited resources
  - Limited understanding of software system interactions
  - Lots of patches to assess!

Patch Frequency

- 3 Gentoo Linux machines
  - Workstation, data munger, web server
  - 923, 655, and 122 installed software packages
  - 1453 unique packages total
- During 2008 calendar year
  - 2402 new upstream versions
  - 260 working days, 8 hour workday
  - Over 1 update/hour for admin...non-stop!

Audience Reaction

Most do.

But breakage can be very expensive.

Not all are.

We aim to complement existing tools.

Does not compute.

Food for Thought

On one extreme...

Would you expend resources to patch an area of code that is never exercised by your application?

On the other extreme...

Would you avoid patching an area of code that is a hot path due to the potential risk?

What about the non-extremes?

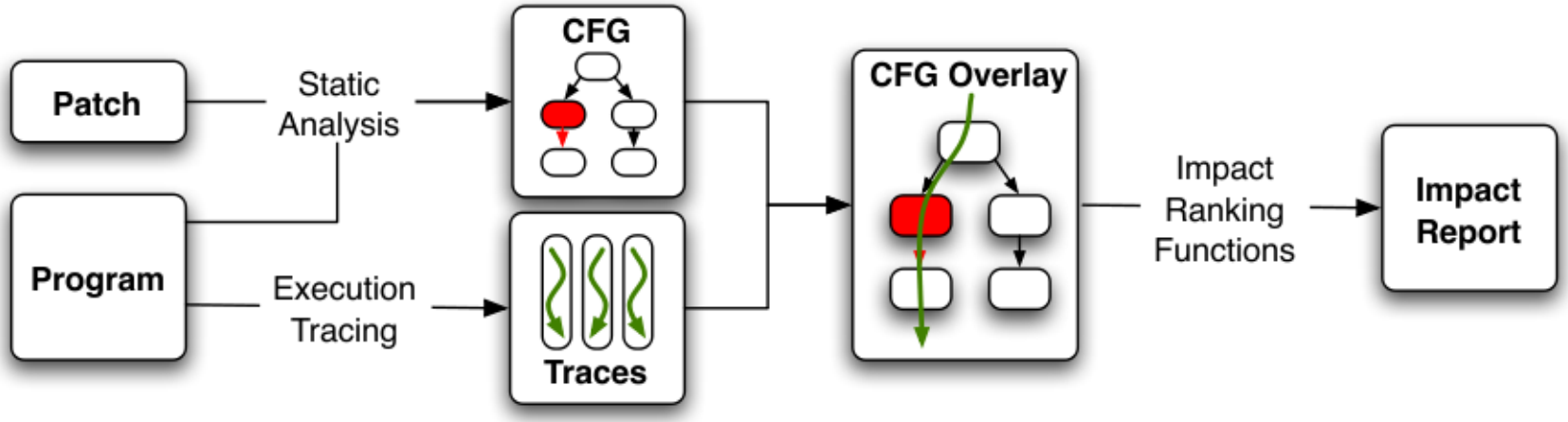
How do we balance the risk/benefit trade-off of a patch?

Our Approach

Simple intuition: Modifying more commonly used code will likely have a greater impact (whether positive or negative) on a software system.

- PatchAdvisor
  - Infer patch impact in an automated fashion
  - Combination of static analysis and dynamic tracing
- Our Goal
  - Provide administrator useful information about patch impact (or at least > 0 information!)
  - Enable informed decisions about patching

PatchAdvisor Overview

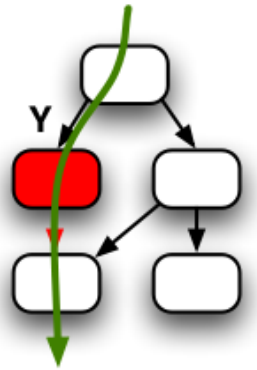


- Three stages of PatchAdvisor
  - Pre/post-patch CFG generation and diffing
  - Execution tracing and CFG overlay
  - Impact analysis and reporting

Impact Analysis

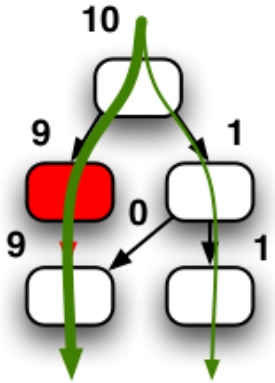
Three proposed functions to infer patch impact:

Naive Binary:



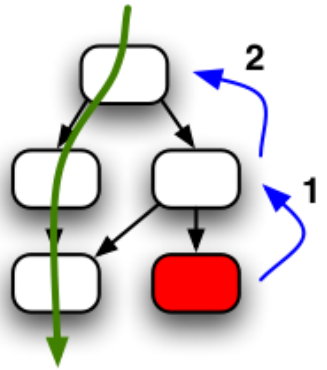
Do we intersect at all with the modified code areas?

Trace Weighted:



How often do we intersect with modified code areas?

Proximity Ranking:



If we don't intersect, how close are we?

Impact Reporting

Open question: How can we most effectively convey actionable information to the administrator?

- Output of impact metrics?
- List of affected functions?
- List of impacted inputs?
- Risk index based on previous patches and/or failures of a package?
- SVN commits / mailing list messages that correspond to patched machine code?
- ...

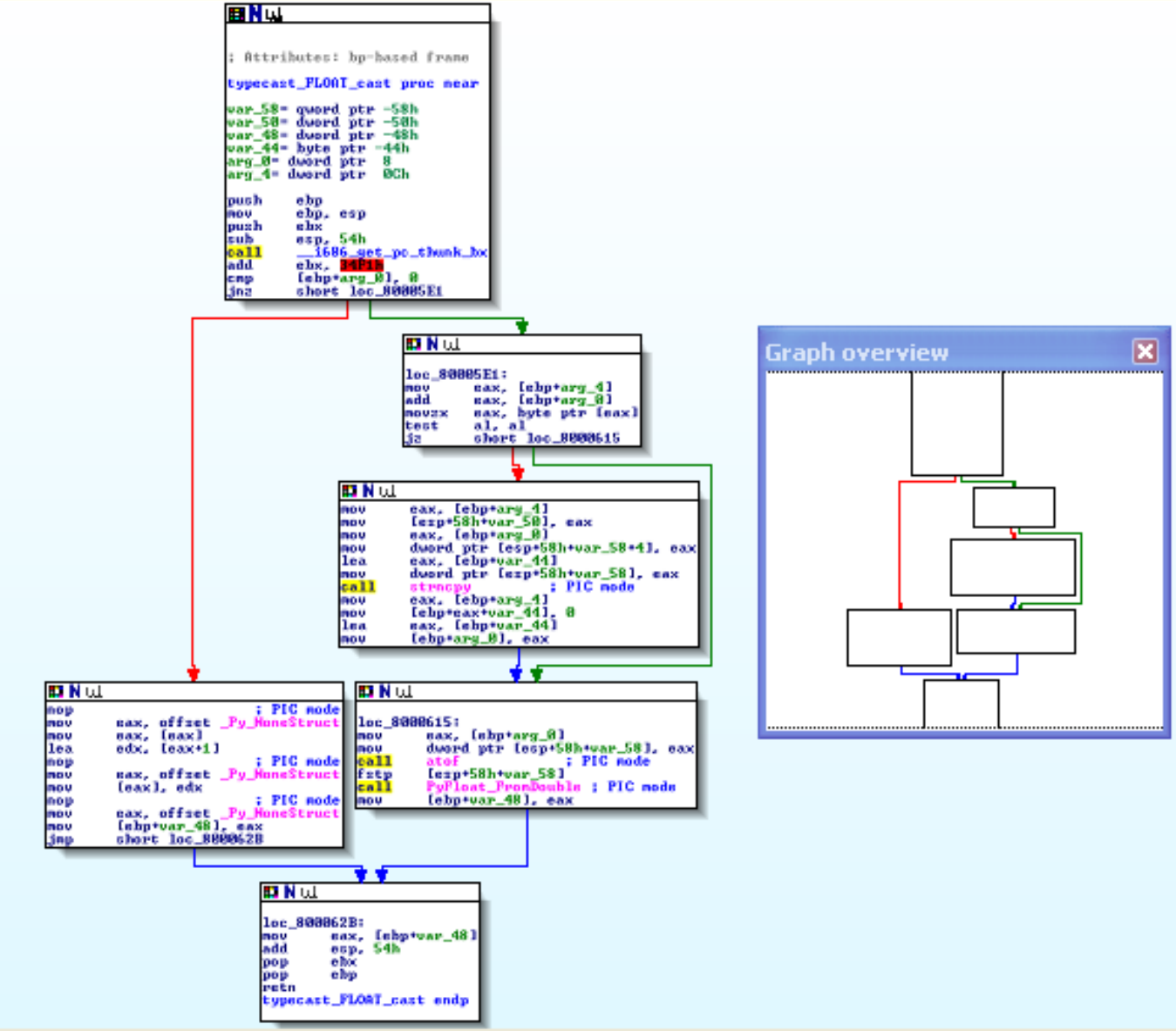
Implementation

- IDA Pro
  - Disassembler, debugger
  - Intended for hostile code analysis
  - http://www.hex-rays.com/idapro/
- Pai Mei
  - Extensible python framework for RCE
  - http://code.google.com/p/paimei/
- CFG generation, binary diffing, func/bb tracing, overlay
- Not yet fully automated

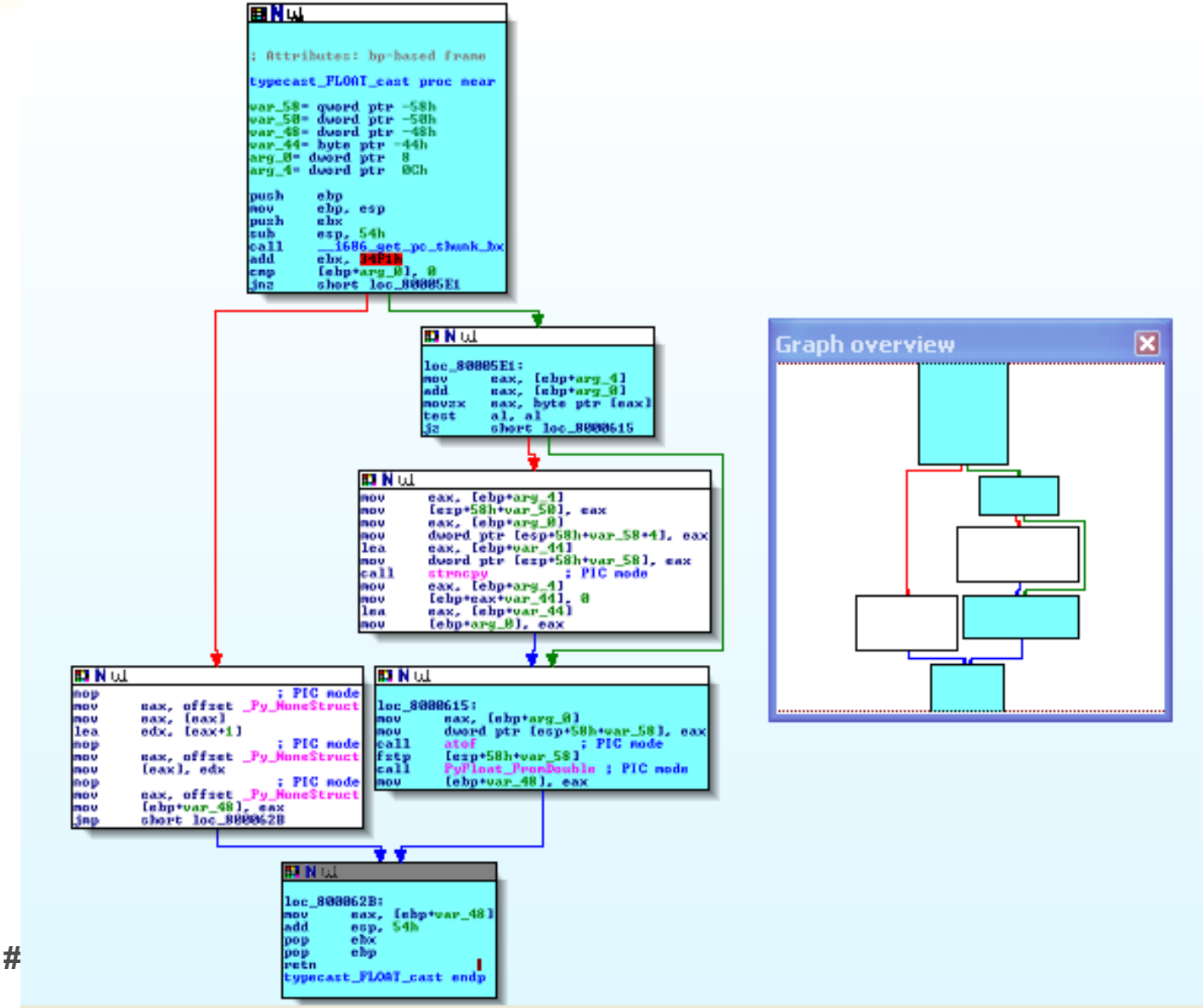
Preliminary Evaluation

- Web stacks have many layers
  - HTTP server, scripting languages, dispatcher, ORM, backend DB, template engine, etc
- Psycopg2
  - Popular PostgreSQL Python bindings
  - Minor revision upgrade: 2.0.2 → 2.0.3
  - Innocent looking ChangeLog
  - Has a test suite!
- NULL deref when involving any FLOAT column types

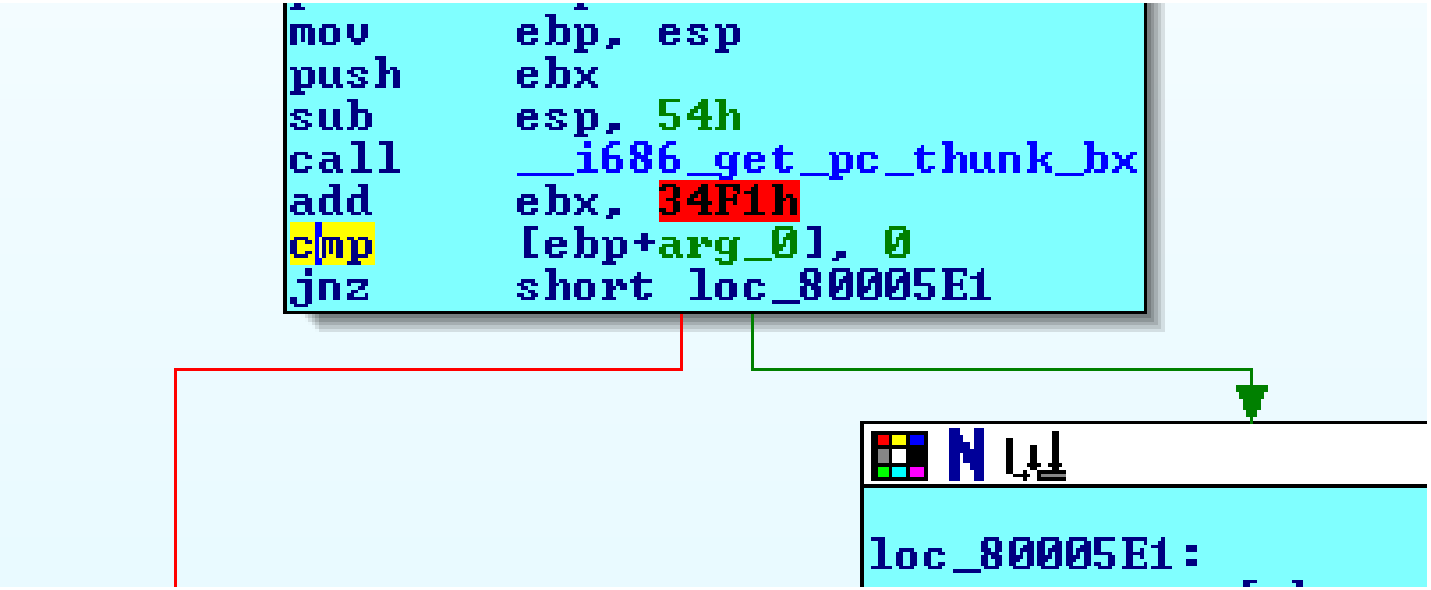
Before Patch



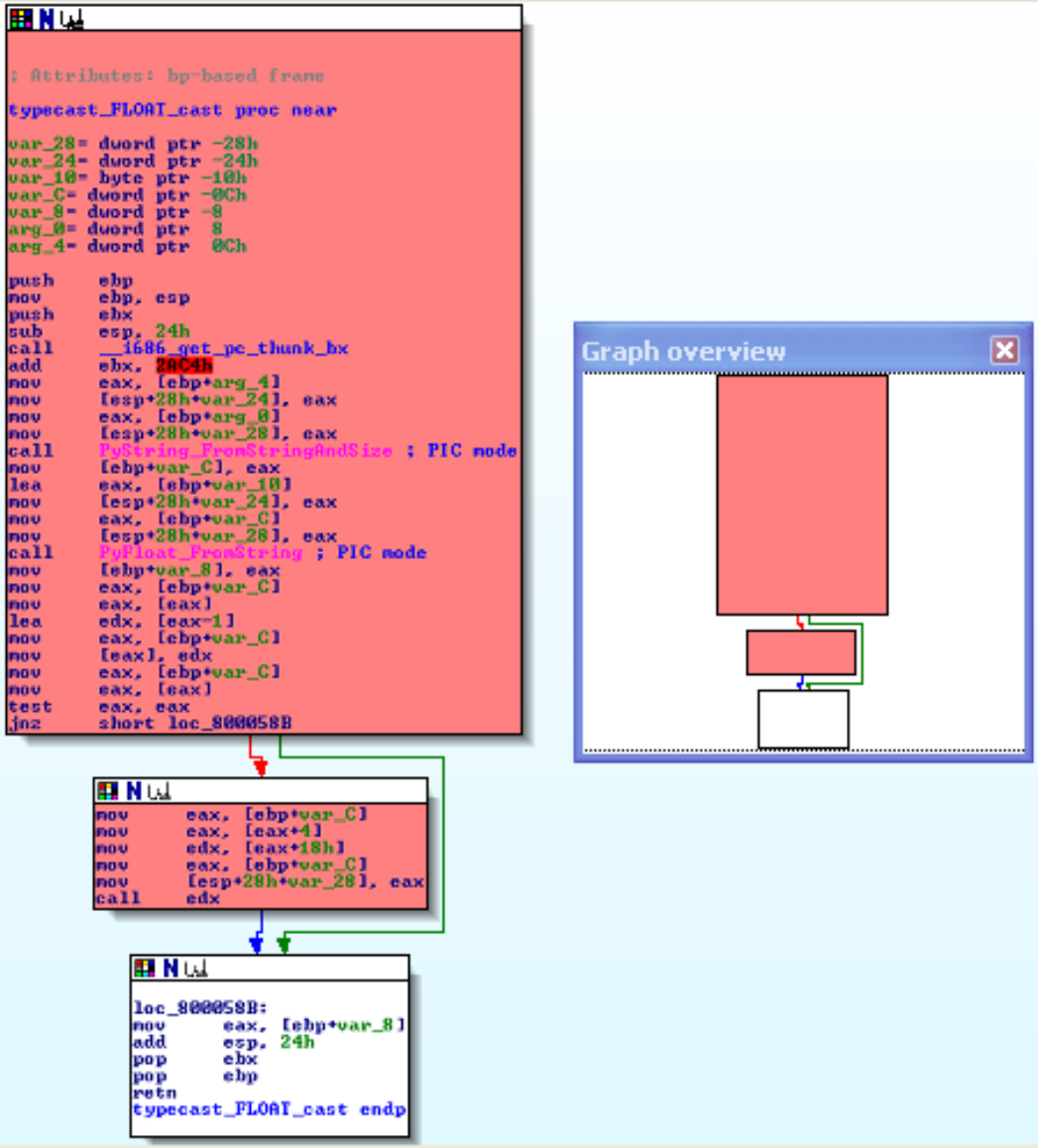
Execution Trace



NULL check


mov ebp, esp
push ebx
sub esp, 54h
call __i686_get_pc_thunk_bx
add ebx, 34F1h
cmp [ebp+arg_0], 0
jnz short loc_80005E1
loc_80005E1:

After Patch


; Attributes: bp-based frame
typecast_FLOAT_cast proc near
var_28= dword ptr -28h
var_24= dword ptr -24h
var_10= byte ptr -10h
var_C= dword ptr -0Ch
var_8= dword ptr -8
arg_0= dword ptr 8
arg_4= dword ptr 0Ch
push ebp
mov ebp, esp
push ebx
sub esp, 24h
call __i686_get_pc_thunk_bx
add ebx, 2AC4h
mov eax, [ebp+arg_4]
mov [esp+28h+var_24], eax
mov eax, [ebp+arg_0]
mov [esp+28h+var_28], eax
call PyString_FromStringAndSize ; PIC mode
mov [ebp+var_C], eax
lea eax, [ebp+var_10]
mov [esp+28h+var_24], eax
mov eax, [ebp+var_C]
mov [esp+28h+var_28], eax
call PyFloat_FromString ; PIC mode
mov [ebp+var_8], eax
mov eax, [ebp+var_C]
mov eax, [eax]
lea edx, [eax-1]
mov eax, [ebp+var_C]
mov [eax], edx
mov eax, [ebp+var_C]
mov eax, [eax]
test eax, eax
jnz short loc_800588B
mov eax, [ebp+var_C]
mov eax, [eax+4]
mov edx, [eax+18h]
mov eax, [ebp+var_C]
mov [esp+28h+var_28], eax
call edx
loc_800588B:
mov eax, [ebp+var_8]
add esp, 24h
pop ebx
pop ebp
retn
typecast_FLOAT_cast endp
Graph overview

Future Directions

- Improved ranking heuristics
  - How do programs often fail in the real-world?
- Application-specific knowledge
  - Can we use application/domain-specific information to aid our inference (eg. web apps)?
- Patch splicing
  - Can we determine intra-patch dependencies and splice out high risk changes?
- Patch classification
  - Can we infer whether a patch fixes a semantic bug, performance issue, or security vulnerability?

Question and Answer

Questions?

- Contact information
  - Jon Oberheide
  - University of Michigan
  - jonojono@umich.edu
  - http://www.eecs.umich.edu/fjgroup/