

# A Bug or Malware?

## Catastrophic consequences either way.

Ben Holland, Suresh Kothari  
Iowa State University

Terminal

```
$ whoami  
benholland  
$
```

Terminal

```
$ whoami
```

```
benholland
```

```
$ groups
```

```
benholland break-fix-teach-it iastate
```

```
$
```

Terminal

```
$ whoami
```

```
benholland
```

```
$ groups
```

```
benholland break-fix-teach-it iastate
```

```
$ members iastate
```

```
benholland sureshkothari
```

```
$
```

Terminal

```
$ whoami
```

```
benholland
```

```
$ groups
```

```
benholland break-fix-teach-it iastate
```

```
$ members iastate
```

```
benholland sureshkothari
```

```
$ pwd
```

```
/home/derbycon/talks
```

```
$
```

Terminal

```
$ whoami
benholland
$ groups
benholland break-fix-teach-it iastate
$ members iastate
benholland sureshkothari
$ pwd
/home/derbycon/talks
$ ls -l
total 16
drwxr-xr-x 2 derbycon break-it          4096 Aug 28 13:50 track1
drwxr-xr-x 2 derbycon fix-it           4096 Aug 28 13:50 track2
drwxr-xr-x 2 derbycon teach-it         4096 Aug 28 13:50 track3
drwxr-xr-x 2 derbycon break-fix-teach-it 4096 Aug 28 14:15 track4
$
```

```
Terminal
$ whoami
benholland
$ groups
benholland break-fix-teach-it iastate
$ members iastate
benholland sureshkothari
$ pwd
/home/derbycon/talks
$ ls -l
total 16
drwxr-xr-x 2 derbycon break-it          4096 Aug 28 13:50 track1
drwxr-xr-x 2 derbycon fix-it           4096 Aug 28 13:50 track2
drwxr-xr-x 2 derbycon teach-it         4096 Aug 28 13:50 track3
drwxr-xr-x 2 derbycon break-fix-teach-it 4096 Aug 28 14:15 track4
$
```

...but not necessarily in that order ;)

Terminal

```
$ whoami
benholland
$ groups
benholland break-fix-teach-it iastate
$ members iastate
benholland sureshkothari
$ pwd
/home/derbycon/talks
$ ls -l
total 16
drwxr-xr-x 2 derbycon break-it          4096 Aug 28 13:50 track1
drwxr-xr-x 2 derbycon fix-it           4096 Aug 28 13:50 track2
drwxr-xr-x 2 derbycon teach-it         4096 Aug 28 13:50 track3
drwxr-xr-x 2 derbycon break-fix-teach-it 4096 Aug 28 14:15 track4
$ cd track4; ls -la
total 12
drwxr-xr-x 2 derbycon  break-fix-teach-it 4096 Aug 28 14:15 .
drwxr-xr-x 6 derbycon  derbycon           4096 Aug 28 13:50 ..
-rwxrwxrwx 1 benholland root              520 Aug 28 16:08 a_bug_or_malware.sh
$
```



Terminal

```
$ whoami
benholland
$ groups
benholland break-fix-teach-it iastate
$ members iastate
benholland sureshkothari
$ pwd
/home/derbycon/talks
$ ls -l
total 16
drwxr-xr-x 2 derbycon break-it          4096 Aug 28 13:50 track1
drwxr-xr-x 2 derbycon fix-it           4096 Aug 28 13:50 track2
drwxr-xr-x 2 derbycon teach-it         4096 Aug 28 13:50 track3
drwxr-xr-x 2 derbycon break-fix-teach-it 4096 Aug 28 14:15 track4
$ cd track4; ls -la
total 12
drwxr-xr-x 2 derbycon  break-fix-teach-it 4096 Aug 28 14:15 .
drwxr-xr-x 6 derbycon  derbycon           4096 Aug 28 13:50 ..
-rwxrwxrwx 1 benholland root              520 Aug 28 16:08 a_bug_or_malware.sh
$ ./a_bug_or_malware.sh
```

\*This material is based on research sponsored by DARPA under agreement number FA8750-12-2-0126. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

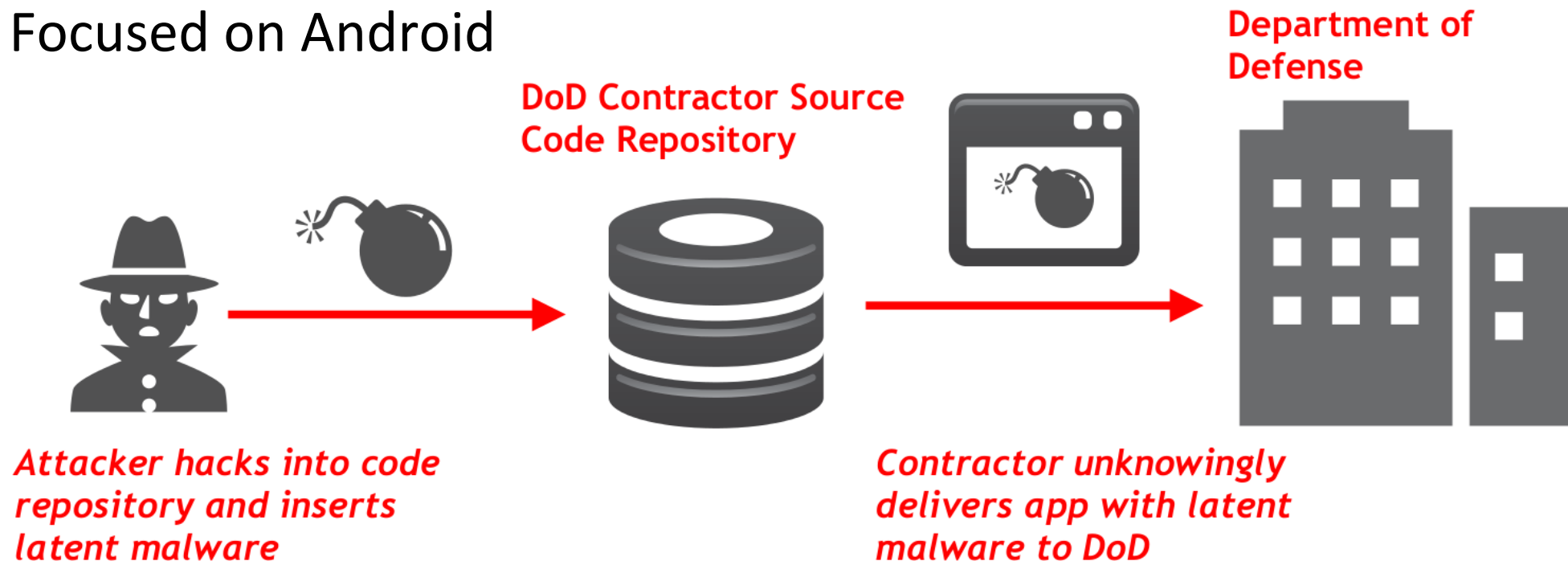
\*\*The views and opinions expressed in this presentation are those of the authors and do not necessarily reflect the official policy or position of any agency of the U.S. government or Iowa State University.

Hello World!

```
$ █
```

# DARPA's APAC Program

- Automated Program Analysis For Cybersecurity (APAC)
- Scenario: Hardened devices, internal app store, untrusted contractors, expert adversaries
- Focused on Android

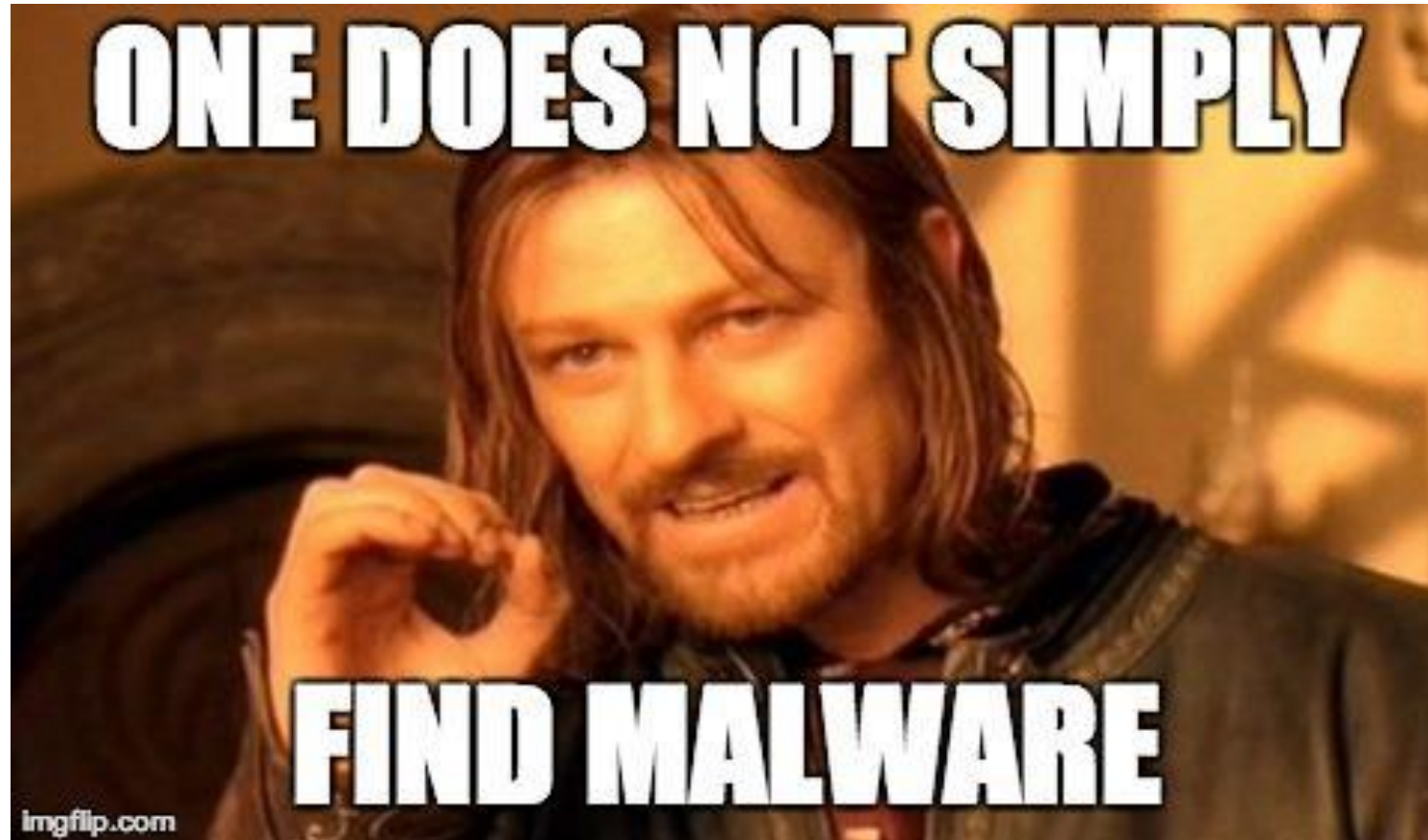


# DARPA's APAC Program

- Automated Program Analysis For Cybersecurity (APAC)
- Scenario: Hardened devices, internal app store, untrusted contractors, expert adversaries
- Focused on Android

Need precision tools to detect **novel** and **sophisticated** malware in advance!

What have we learned?



# What to expect in this talk...

- This talk does not have all the answers...
- Step back and ask some fundamental questions
- Let's start a discussion

# Ice Breaker: Do you agree?

- Antivirus protects us from modern malware.
- Antivirus protects us from yesterday's threats.
- Antivirus protects us from last year's threats.
- Antivirus is totally worthless.

# Exercise: Refactoring CVE-2012-4681

- “Allows remote attackers to execute arbitrary code via a crafted applet that bypasses SecurityManager restrictions...”
- CVE Created August 27th 2012 (~2 years ago...)
- [github.com/benjhollo/CVE-2012-4681-Armoring](https://github.com/benjhollo/CVE-2012-4681-Armoring)

Sample	Notes	Score (positive detections)
Original Sample	<a href="http://pastie.org/4594319">http://pastie.org/4594319</a>	30/55
Technique A	Changed Class/Method names	28/55
Techniques A and B	Obfuscate strings	16/55
Techniques A-C	Change Control Flow	16/55
Techniques A-D	Reflective invocations (on sensitive APIs)	3/55
Techniques A-E	Simple XOR Packer	0/55

# Let's define malware

- Bad (malicious) software
- Examples: Viruses, Worms, Trojan Horses, Rootkits, Backdoors, Adware, Spyware, Keyloggers, Dialers, Ransomware...

Google define malware

Web Images News Shopping Maps More Search tools

About 3,480,000 results (0.41 seconds)

## mal·ware

*/ˈmɑːlˌweɪr/*

*noun* **COMPUTING**  
noun: malware

1. software that is intended to damage or disable computers and computer systems.

Origin

ENGLISH  
malicious

ENGLISH  
software

malware

blend of *malicious* and *software*.

Translate malware to

Use over time for: malware

Mentions

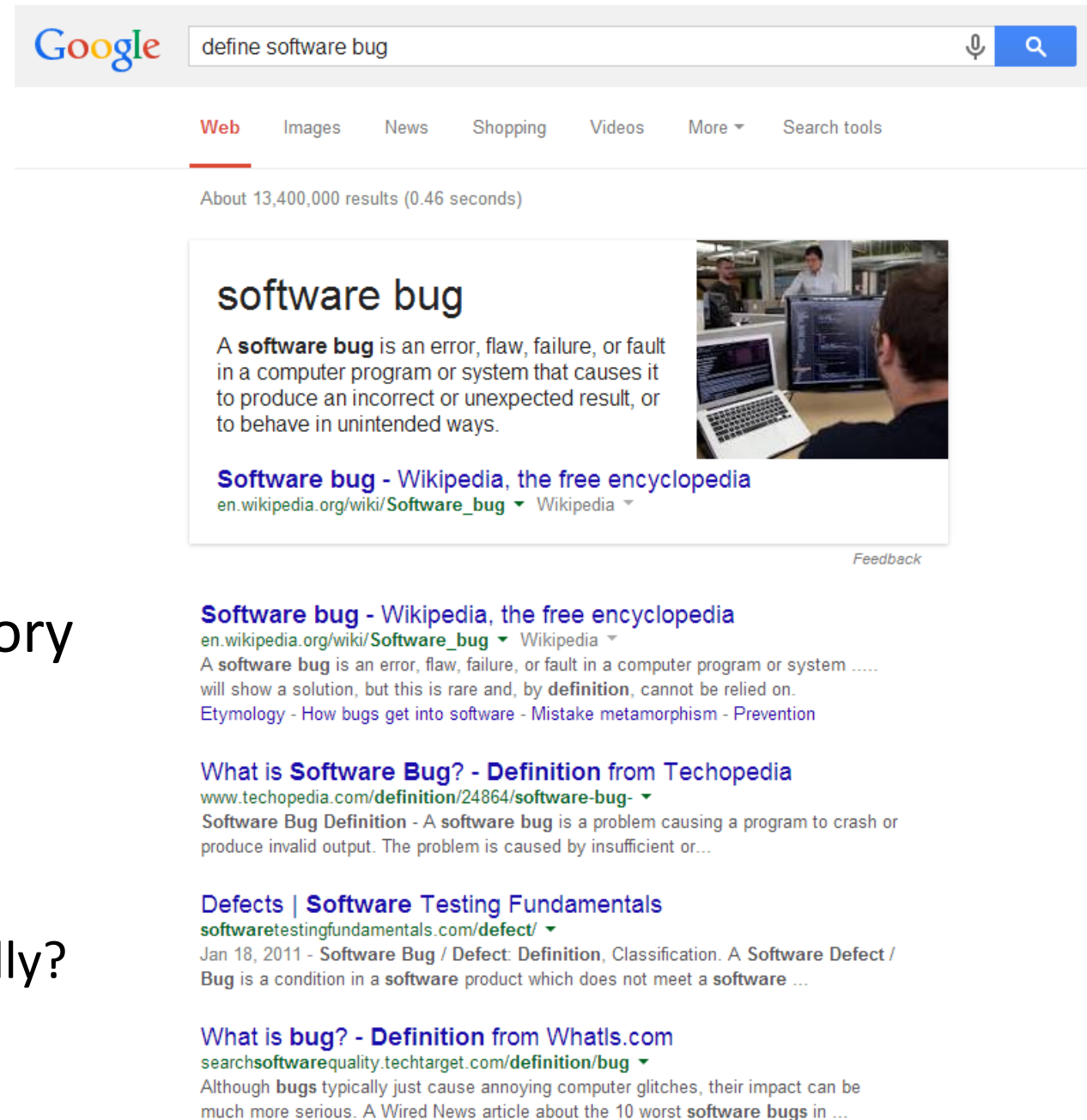
1800 1850 1900 1950 2010

The graph shows a sharp increase in mentions starting around 2000, reaching a peak near 2010. The x-axis represents years from 1800 to 2010, and the y-axis represents the number of mentions.



# Let's define a "bug"

- Unintentional error, flaw, failure, fault
- Examples: Rounding errors, null pointers, infinite loops, stack overflows, race conditions, memory leaks, business logic flaws...
- Is a software bug malware?
  - What if I added the bug intentionally?



Google define software bug

Web Images News Shopping Videos More Search tools

About 13,400,000 results (0.46 seconds)

**software bug**

A **software bug** is an error, flaw, failure, or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways.

[Software bug - Wikipedia, the free encyclopedia](#)  
en.wikipedia.org/wiki/Software\_bug

[Software bug - Wikipedia, the free encyclopedia](#)  
en.wikipedia.org/wiki/Software\_bug

A **software bug** is an error, flaw, failure, or fault in a computer program or system ... will show a solution, but this is rare and, by **definition**, cannot be relied on.  
[Etymology](#) - [How bugs get into software](#) - [Mistake metamorphism](#) - [Prevention](#)

**What is Software Bug? - Definition from Techopedia**  
[www.techopedia.com/definition/24864/software-bug-](#)  
**Software Bug Definition** - A **software bug** is a problem causing a program to crash or produce invalid output. The problem is caused by insufficient or...

**Defects | Software Testing Fundamentals**  
[softwaretestingfundamentals.com/defect/](#)  
Jan 18, 2011 - **Software Bug / Defect: Definition, Classification.** A **Software Defect / Bug** is a condition in a **software product** which does not meet a **software** ...

**What is bug? - Definition from WhatIs.com**  
[searchsoftwarequality.techtarget.com/definition/bug](#)  
Although **bugs** typically just cause annoying computer glitches, their impact can be much more serious. A **Wired News** article about the 10 worst **software bugs** in ...

# A bug or malware?

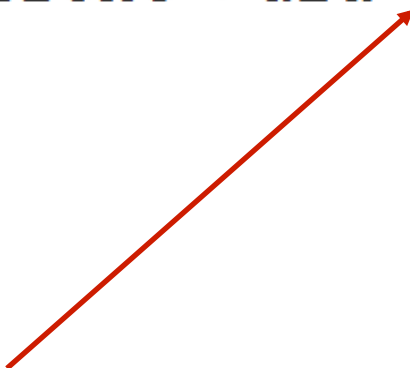
- Context: Found in a CVS commit to the Linux Kernel source

```
if ((options == (__WCLONE|__WALL)) && (current->uid = 0))  
    retval = -EINVAL;
```

Hint: This never executes...



"=" vs. "==" is a subtle yet important difference!  
Would grant root privilege to any user that knew  
how to trigger this condition.



# Malware: Linux Backdoor Attempt (2003)

- <https://freedom-to-tinker.com/blog/felten/the-linux-backdoor-attempt-of-2003/>

```
if ((options == (__WCLONE|__WALL)) && (current->uid = 0))  
    retval = -EINVAL;
```

Hint: This never executes...

"=" vs. "==" is a subtle yet important difference!  
Would grant root privilege to any user that knew  
how to trigger this condition.

# A bug or malware?

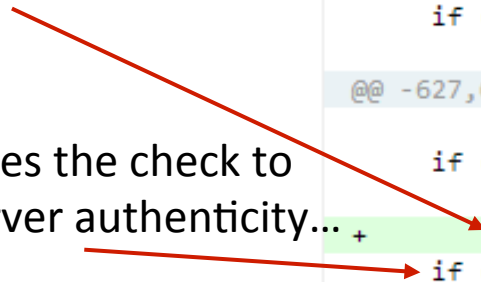
```
-
-         if ((err = ReadyHash(&SSLHashMD5, &hashCtx, ctx)) != 0)
600 +
601 +         if ((err = ReadyHash(&SSLHashMD5, &hashCtx)) != 0)
602             goto fail;
603             if ((err = SSLHashMD5.update(&hashCtx, &clientRandom)) != 0)
604                 goto fail;
... @@ -616,10 +617,10 @@ OSStatus FindSigAlg(SSLContext *ctx,
617
618         hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
619         hashOut.length = SSL_SHA1_DIGEST_LEN;
-         if ((err = SSLFreeBuffer(&hashCtx, ctx)) != 0)
620 +         if ((err = SSLFreeBuffer(&hashCtx)) != 0)
621             goto fail;
622
-         if ((err = ReadyHash(&SSLHashSHA1, &hashCtx, ctx)) != 0)
623 +         if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
624             goto fail;
625             if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
626                 goto fail;
... @@ -627,6 +628,7 @@ OSStatus FindSigAlg(SSLContext *ctx,
628             goto fail;
629             if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
630                 goto fail;
631 +                 goto fail;
632             if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
633                 goto fail;
634
```

# A bug or malware?

```
-
-         if ((err = ReadyHash(&SSLHashMD5, &hashCtx, ctx)) != 0)
+
+         if ((err = ReadyHash(&SSLHashMD5, &hashCtx)) != 0)
+             goto fail;
+         if ((err = SSLHashMD5.update(&hashCtx, &clientRandom)) != 0)
+             goto fail;
@@ -616,10 +617,10 @@ OSStatus FindSigAlg(SSLContext *ctx,
+
+         hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
+         hashOut.length = SSL_SHA1_DIGEST_LEN;
-         if ((err = SSLFreeBuffer(&hashCtx, ctx)) != 0)
+         if ((err = SSLFreeBuffer(&hashCtx)) != 0)
+             goto fail;
-
-         if ((err = ReadyHash(&SSLHashSHA1, &hashCtx, ctx)) != 0)
+         if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
+             goto fail;
+         if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
+             goto fail;
@@ -627,6 +628,7 @@ OSStatus FindSigAlg(SSLContext *ctx,
+         goto fail;
+         if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
+             goto fail;
+         goto fail;
+         if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
+             goto fail;
```

Always goto fail

Never does the check to verify server authenticity...



# Bug?: Apple SSL CVE-2014-1266

```
-         if ((err = ReadyHash(&SSLHashMD5, &hashCtx, ctx)) != 0)
+         if ((err = ReadyHash(&SSLHashMD5, &hashCtx)) != 0)
+             goto fail;
+             if ((err = SSLHashMD5.update(&hashCtx, &clientRandom)) != 0)
+                 goto fail;
@@ -616,10 +617,10 @@ OSStatus FindSigAlg(SSLContext *ctx,
    hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
    hashOut.length = SSL_SHA1_DIGEST_LEN;
-     if ((err = SSLFreeBuffer(&hashCtx, ctx)) != 0)
+     if ((err = SSLFreeBuffer(&hashCtx)) != 0)
+         goto fail;

-     if ((err = ReadyHash(&SSLHashSHA1, &hashCtx, ctx)) != 0)
+     if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
+         goto fail;
+         if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
+             goto fail;
@@ -627,6 +628,7 @@ OSStatus FindSigAlg(SSLContext *ctx,
    goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
+     goto fail;
+     if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
+         goto fail;
```

Always goto fail

Never does the check to  
verify server authenticity...

- Should have been caught by automated tools
- Survived almost a year
- Affected OSX and iOS

# A bug or malware?

```
3969     unsigned int payload;
3970     unsigned int padding = 16; /* Use minimum padding */
3971
3972     /* Read type and payload length first */
3973     hbtype = *p++;
3974     n2s(p, payload);
3975     pl = p;
3976
3977     if (s->msg_callback)
3978         s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
3979                        &s->s3->rrec.data[0], s->s3->rrec.length,
3980                        s, s->msg_callback_arg);
3981
3982     if (hbtype == TLS1_HB_REQUEST)
3983     {
3984         unsigned char *buffer, *bp;
3985         int r;
3986
3987         /* Allocate memory for the response, size is 1 bytes
3988          * message type, plus 2 bytes payload length, plus
3989          * payload, plus padding
3990          */
3991         buffer = OPENSSL_malloc(1 + 2 + payload + padding);
3992         bp = buffer;
3993
3994         /* Enter response type, length and copy payload */
3995         *bp++ = TLS1_HB_RESPONSE;
3996         s2n(payload, bp);
3997         memcpy(bp, pl, payload);
```

Hint: More SSL fun...



# Bug (I hope): Heartbleed

- Much less obvious
- Survived several code audits
- Live for ~2 years

Heartbeat message size controlled by the attacker...

Response size also controlled by the attacker...

Reads too much data!

```
unsigned int payload;
unsigned int padding = 16; /* Use minimum padding */

/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;

if (s->msg_callback)
    s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
                    &s->s3->rrec.data[0], s->s3->rrec.length,
                    s, s->msg_callback_arg);

if (hbtype == TLS1_HB_REQUEST)
{
    unsigned char *buffer, *bp;
    int r;

    /* Allocate memory for the response, size is 1 bytes
     * message type, plus 2 bytes payload length, plus
     * payload, plus padding
     */
    buffer = OPENSSL_malloc(1 + 2 + payload + padding);
    bp = buffer;

    /* Enter response type, length and copy payload */
    *bp++ = TLS1_HB_RESPONSE;
    s2n(payload, bp);
    memcpy(bp, pl, payload);
}
```



"Catastrophic" is the right word. On the scale of 1 to 10, this is an 11.

-Bruce Schneier



# A bug or malware?

Hint...

```
178 /* Parse and execute the commands in STRING. Returns whatever
179 execute_command () returns. This frees STRING. FLAGS is a
180 flags word; look in common.h for the possible values. Actions
181 are:
182     (flags & SEVAL_NONINT) -> interactive = 0;
183     (flags & SEVAL_INTERACT) -> interactive = 1;
184     (flags & SEVAL_NOHIST) -> call bash_history_disable ()
185     (flags & SEVAL_NOFREE) -> don't free STRING when finished
186     (flags & SEVAL_RESETLINE) -> reset line_number to 1
187 */
188
189 int
190 parse_and_execute (string, from_file, flags)
191     char *string;
192     const char *from_file;
193     int flags;
194 {
    ...
```

Fix adds:

```
+ #define SEVAL_FUNCDEF 0x080 /* only allow function definitions */
+ #define SEVAL_ONECMD 0x100 /* only allow a single command */
```

Missing some input validation checks...

```
315 /* Initialize the shell variables from the current environment.
316 If PRIVMODE is nonzero, don't import functions from ENV or
317 parse $SHELLOPTS. */
318 void
319 initialize_shell_variables (env, privmode)
320     char **env;
321     int privmode;
322 {
323     char *name, *string, *temp_string;
324     int c, char_index, string_index, string_length, ro;
325     SHELL_VAR *temp_var;
326
327     create_variable_tables ();
328
329     for (string_index = 0; string = env[string_index++]; )
330     {
331         char_index = 0;
332         name = string;
333         while ((c = *string++) && c != '=')
334             ;
335         if (string[-1] == '=')
336             char_index = string - name - 1;
337
338         /* If there are weird things in the environment, like `=xxx' or a
339            string without an `=', just skip them. */
340         if (char_index == 0)
341             continue;
342
343         /* ASSERT(name[char_index] == '=') */
344         name[char_index] = '\0';
345         /* Now, name = env variable name, string = env variable value, and
346            char_index == strlen (name) */
347
348         temp_var = (SHELL_VAR *)NULL;
349
350         /* If exported function, define it now. Don't import functions from
351            the environment in privileged mode. */
352         if (privmode == 0 && read_but_dont_execute == 0 && STREQN ("() {", string, 4))
353         {
354             string_length = strlen (string);
355             temp_string = (char *)xmalloc (3 + string_length + char_index);
356
357             strcpy (temp_string, name);
358             temp_string[char_index] = ' ';
359             strcpy (temp_string + char_index + 1, string);
360
361             if (posixly_correct == 0 || legal_identifier (name))
362                 parse_and_execute (temp_string, name, SEVAL_NONINT|SEVAL_NOHIST);
```

# Bug (probably): Shellshock CVE-2014-6271/7169

- Bug is the due to the absence of code (validation checks)
- Present for 25 years!?
- Even more complicated to find
- Still learning the extent of this bug

# Bug (probably): Shellshock CVE-2014-6271/7169



**BASH**



**ShellShock**



# A bug or malware?

```
class A {  
    A a1;  
    A a2;  
    B b;  
    A a4;  
    A a5;  
    int i;  
    A a7;  
};
```

```
class B {  
    A a1;  
    A a2;  
    A a3;  
    A a4;  
    A a5;  
    A a6;  
    A a7;  
};
```

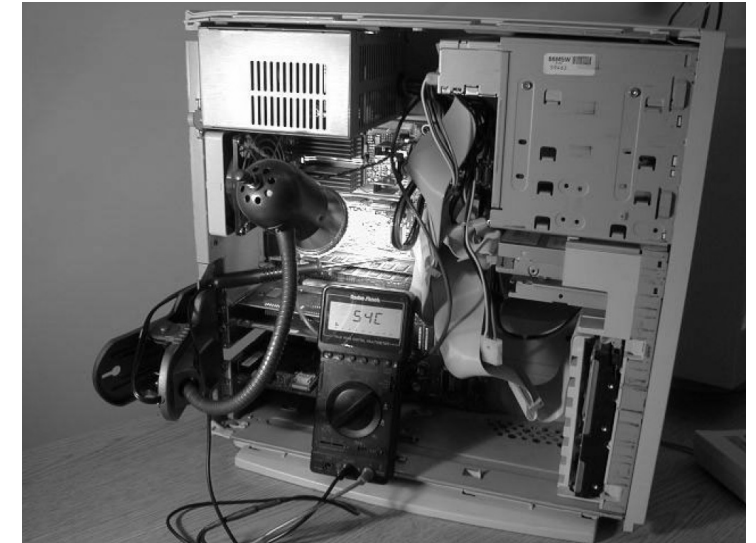
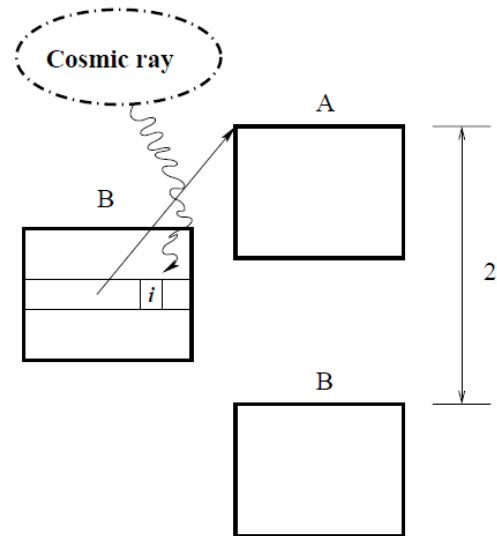
# Malware: VM escape using bit flips

- Govindavajhala, S.; Appel, AW., "Using memory errors to attack a virtual machine," *Proceedings of IEEE Symposium on Security and Privacy*, pp.154-165, May 2003.

```
class A {
  A a1;
  A a2;
  B b;
  A a4;
  A a5;
  int i;
  A a7;
};

class B {
  A a1;
  A a2;
  A a3;
  A a4;
  A a5;
  A a6;
  A a7;
};

A p;
B q;
int offset = 6 * 4;
void write(int address, int value) {
  p.i = address - offset ;
  q.a6.i = value ;
}
```

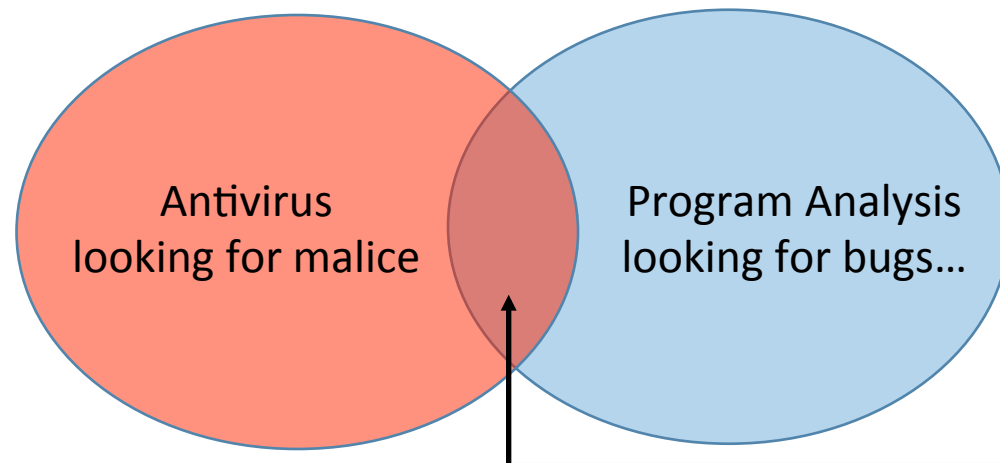


Wait for a bit flip to obtain two pointers of incompatible types that point to the same location to circumvent the type system and execute arbitrary code in the program address space.

# So what's your point?

- Both bugs and malware have catastrophic consequences
- Some bugs are indistinguishable from malware
  - Plausible deniability, malicious intent cannot be determined from code
- Some issues can be found automatically, but not all
- Novel attacks can be extremely hard to detect

Are we doing ourselves a disservice by labeling these as separate problems?



Next time you own a box try dropping a program with an exploitable "bug"

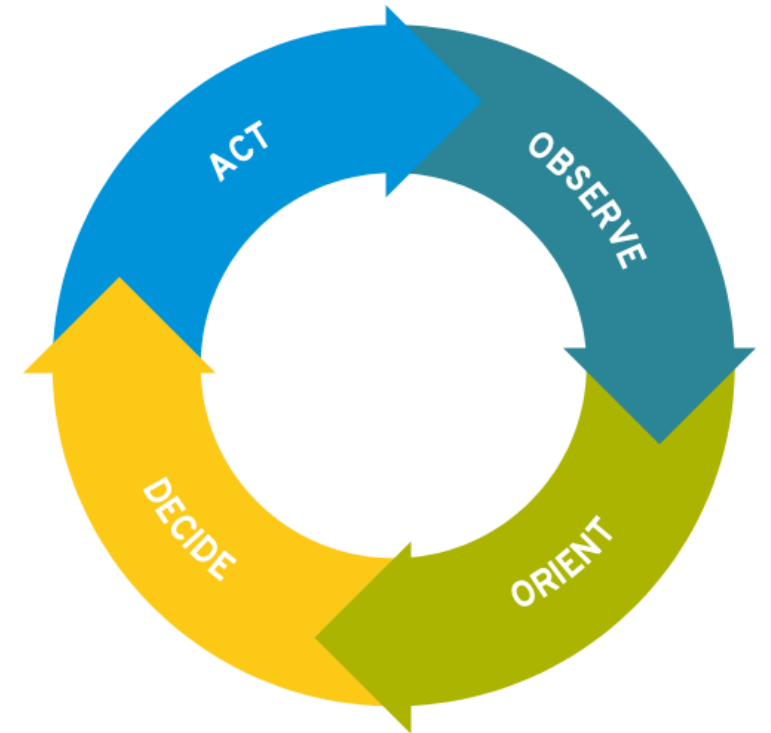
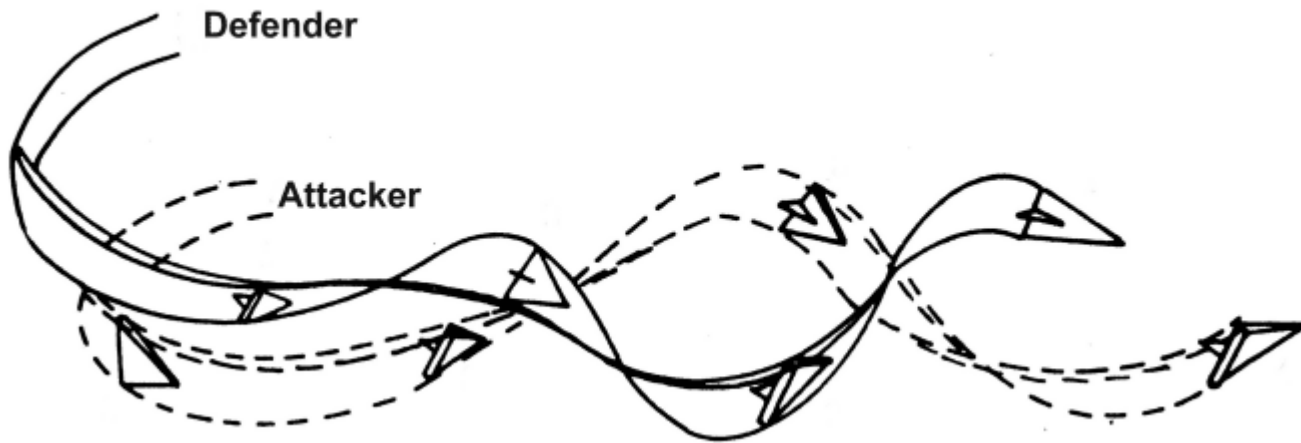
# So what can we do about it?

- Growing infrastructure
  - Complexity of systems keeps increasing
- Manual work is expensive
  - Cost of software is increasing while hardware costs decrease
- We obviously can't automate it all
  - Malware is a cat and mouse game
  - Tricky bugs are tricky...

Need a process to increase human productivity...

# OODA and You

- “Security is a process, not a product” – Bruce Schneier





# OODA and You



## Our opponent

- Time
- Evolution of malware

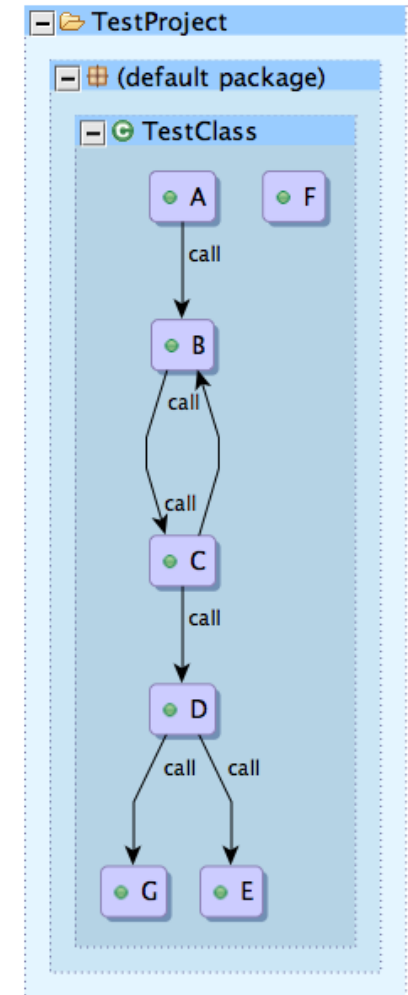
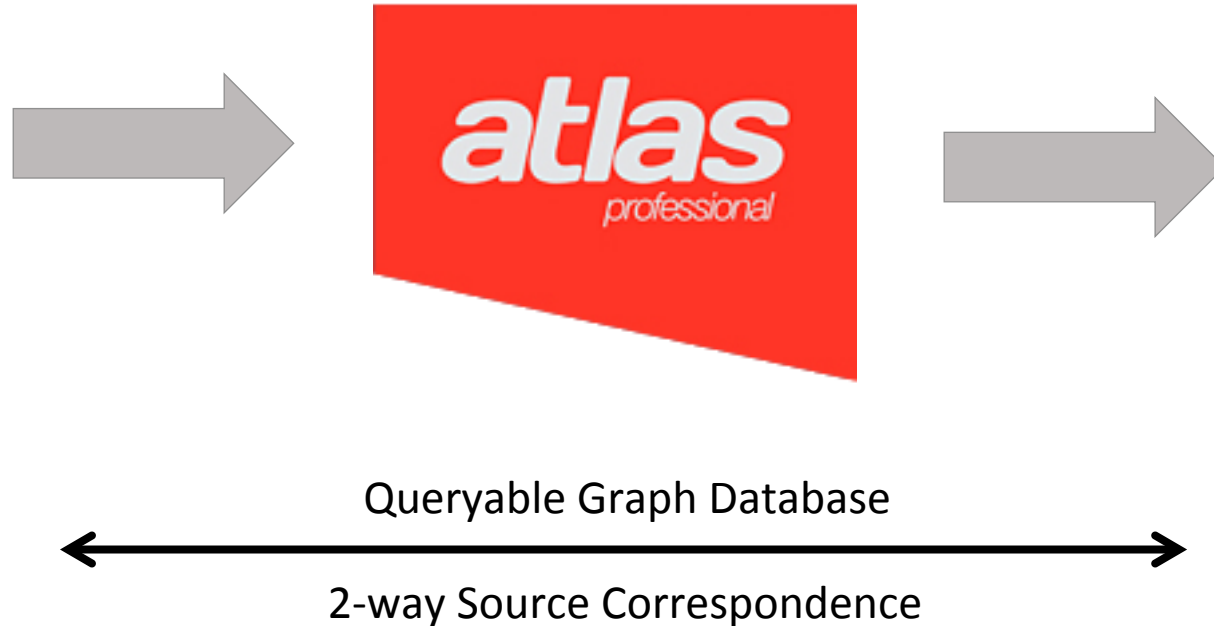
“...IA > AI, that is, that intelligence amplifying systems can, at any given level of available systems technology, beat AI systems. That is, a machine and a mind can beat a mind-imitating machine working by itself.”

– Fred Brooks

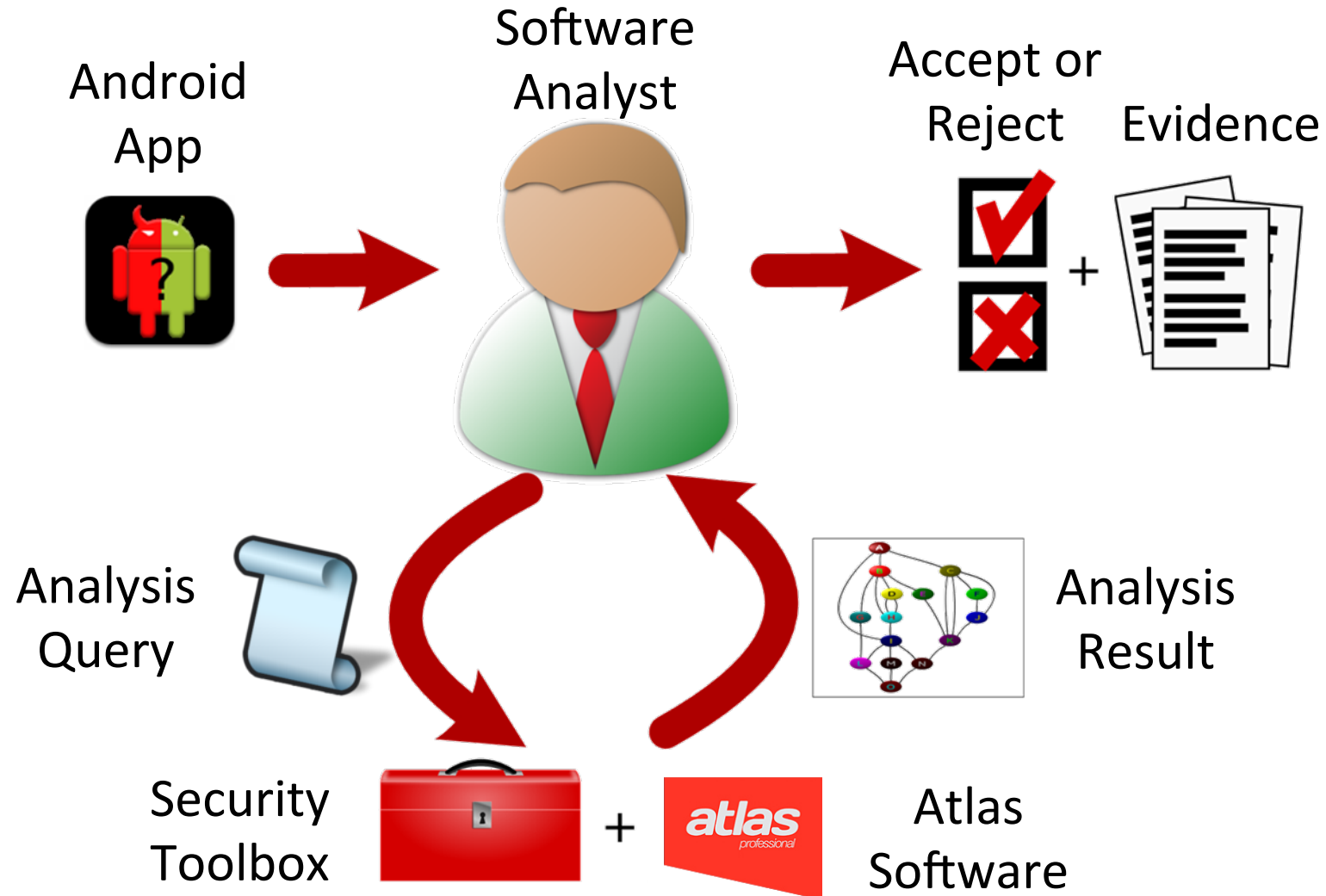
# Speeding through OODA with Atlas

```
1 public class TestClass {  
2  
3     public void AO {  
4         BO;  
5     }  
6  
7     public void BO {  
8         CO;  
9     }  
10  
11    public void CO {  
12        BO;  
13        DO;  
14    }  
15  
16    public void DO {  
17        GO;  
18        EO;  
19    }  
20  
21    public void EO {  
22    }  
23  
24  
25    public void FO {  
26    }  
27  
28  
29    public void GC{  
30    }  
31  
32  
33 }
```

Program Declarations, Control Flow, and Data Flow

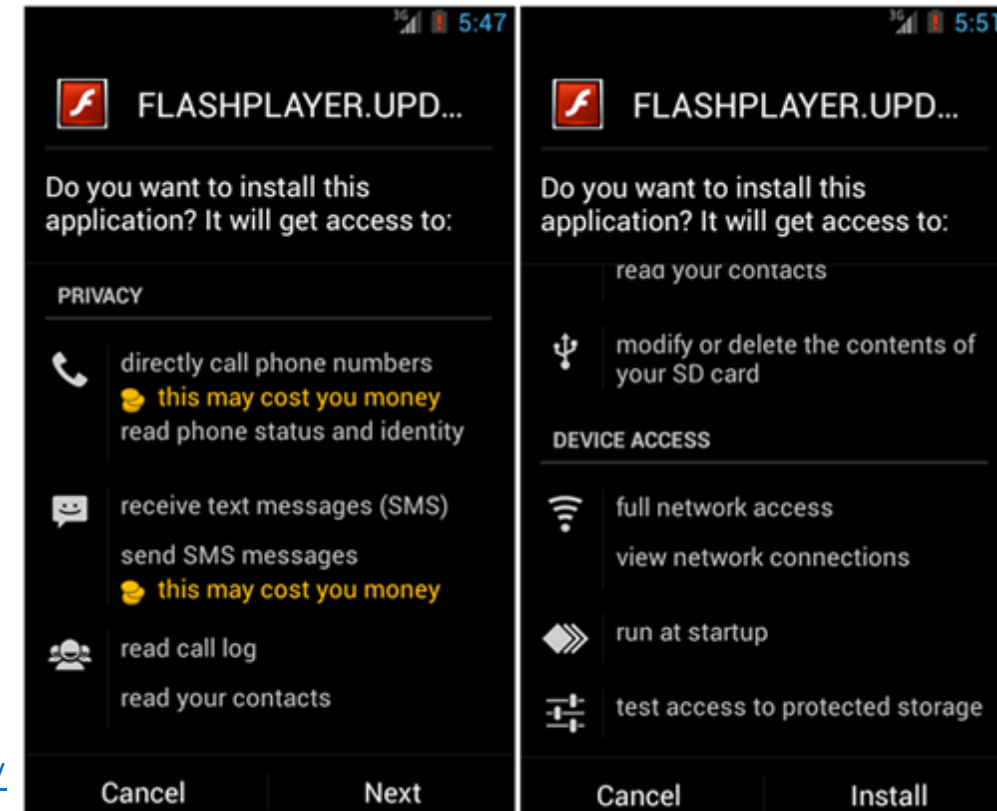


# Speeding through OODA with Atlas



# What about binaries?

- Approach is similar for binary analysis
  - Binary -> Intermediate Language -> Program Graph
- Demo: Analysis of Stels malware
  - Download and execute files
  - Steal contacts lists
  - Report system information
  - Make phone calls
  - Send SMS messages (to premium numbers)
  - Monitor and record and hide SMS messages
  - Show notifications
  - Uninstall apps



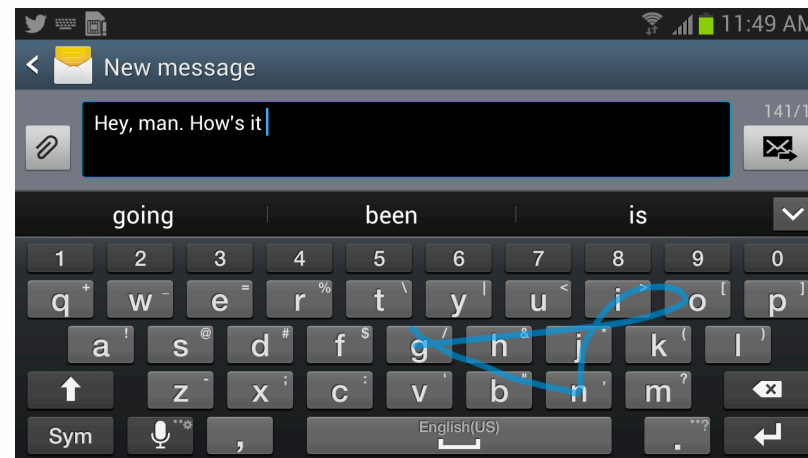
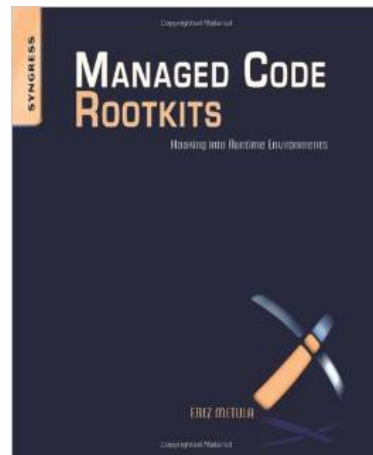
# SpellWrecker

- Consider a spell checker. Invert its logic and what do you get?
- How do we semantically detect the bad one?
- [github.com/benholla/spellwrecker](https://github.com/benholla/spellwrecker)

“Sometimes you have to demo a threat to spark a solution” - Barnaby Jack

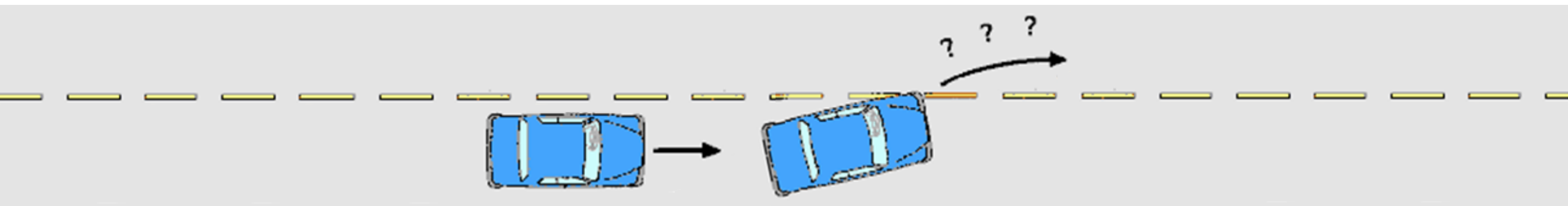
# SpellWrecker

- Consider a spell checker. Invert its logic and what do you get?
- How do we semantically detect the bad one?
- [github.com/benholla/spellwrecker](https://github.com/benholla/spellwrecker)



# Hypothetical Malware

- Cars are becoming drive-by-wire
- Electronic Stability Controls (ESC) are being added to SUVs for rollover prevention



- Invert logic on roll over prevention systems
- Plenty of evil ways to implement it, e.g. greedy algorithms
  - J. Bang-Jensen, G. Gutin, and A. Yeo, "When the greedy algorithm fails," *Discrete Optimizations*, vol. 1, no. 2, pp. 121–127, Nov. 2004.
- Legitimate bugs are hard enough, how can we hope to find illegitimate bugs?

# Questions?

- Thanks!
- Try Atlas: <http://www.ensoftcorp.com/atlas/>
  - Complimentary academic licenses
  - Request a trial

Where did you hear about us?

- EclipseCon Europe
- JavaOne
- YouTube
- Twitter
- Word of Mouth
- Class
- Other

Derbycon wants binary!



# What properties would ideal malware have?

- Operational goals
  - Effective, adaptable
  - Maintaining ownership
  - Cross platform, cross architecture
  - Persistence (survival, removal, updatable)
- Detection avoidance
  - Resistant to static/dynamic analysis (intractable analysis problems)
  - Difficult to characterize
  - Small footprint (low resource consumption, minimal impact)
  - Blends well with legitimate functionality
- Detection mitigation
  - Plausible deniability
  - Kerckhoffs's principle (ex: untraceable transactions)
- General Software Design Issues
  - Maintainable, deployable, scalable, etc.