

Behind DARPA's CGC

A DARPA Cyber Grand Challenge
White Team Retrospective

Presented by: Paul Makowski



Disclaimer

The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

Personnel

- Steve Bassi, Nick Davis, Paul Makowski, Ben Schmidt, Max Koo
- 6 DEF CON CTF wins among us
 - members of: sk3wl 0f r00t, Samurai, DDTek (CTF organizers)
- DARPA CFT: Kensa, BBemu (baseband emulation in QEMU)
- 1 patent granted, 2 pending on (anti-)memory corruption exploitation
- former US government





What We Worked On

Challenge Binary (CB) Development, Bugfinding & CI

CB dev

AFL + Clang in CI

Sparring Partner

“Hadouken”: a mock CRS for
competitors to “spar”
against



CAWS: CGC Architecture Wrecking System

infrastructure audit

dependency fuzzing & VR

Competition Integrity

RE of competitor submissions

hackathon

human CTF team during SE-II

Terminology

CB	Challenge Binary
CS	Challenge Set: a set of CBs that together represent a single challenge. Sometimes (incorrectly) used interchangeably with CB.
RB	Replacement Binary: a modified / patched CB fielded by a competitor
PoV	Proof of Vulnerability (an “exploit”)
Poller	White Team-authored logic to produce benign traffic to measure correctness of RBs
CRS	Cyber Reasoning System: the autonomous systems that competitors built
PA	Program Analysis: the process of automatically analyzing program behavior for, .e.g correctness & security. CRSs strive for autonomous PA.
SE-[I, II]	Scored Event One & Two
CQE	CGC Qualifying Event
CFE	CGC Final Event

Air Gaps

- DARPA - issued laptops,
“air gapped” by OS X firewall
- framework team had root SSH
- prove we installed updates via a signed blob produced
at end of update script
- DECREE was run inside Vagrant (Virtualbox as VM layer)



Credit: [Sean Gallagher \(@thepacketrat\)](#)

Scored Events & Competitions

- **SE-[I, II]**: [December 2nd 2014, April 16th 2014]
 - scored events are optional
 - they allow competitors to benchmark themselves
- **CQE**: June 3, 2015
 - how to PoV: cause SIGSEGV
 - not head-to-head (not throwing against one another)
 - unintentional vulns = bad (they affect consensus score component)

Scored Events & Competitions

- CFE: August 4, 2016
 - how to PoV: control EIP & another register (Type 1) or leak memory (Type 2)
 - non-zero sum, imperfect information, multi-opponent, infinite search space
 - competitors get each other's RBs -> adversarial patching
 - competitors choose who to throw against
 - unintentional vulns = good
 - the game is head-to-head, competitors can make use

Program Analysis 101

- competitors built on:
 - fuzzing - dumb, grammar-based, evolutionary
 - symbolic execution
 - concolic (concrete + symbolic) execution
- (basic) building blocks:
 - lifters / IR (QEMU/TCG, LLVM, BAP, ...)
 - single static assignment (SSA)
 - SAT & SMT solvers (Z3, STP, ...)
 - tools for working with CFGs, DDGs, CDGs -> PDGs

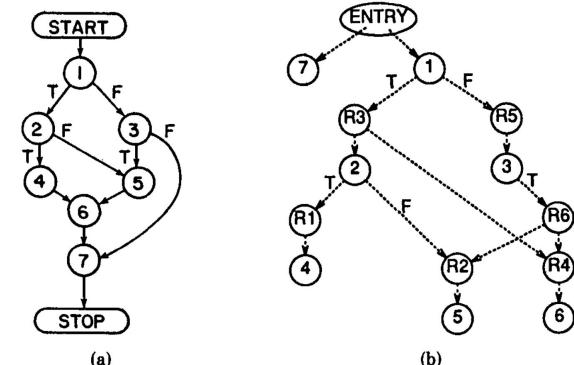


Fig. 1. A control flow graph and its control dependence subgraph.

Jeanne Ferrante; Ottenstein, Karl J.; Warren, Joe D. (July 1987).
["The Program Dependence Graph and its Use in Optimization"](#) (PDF).
 ACM Transactions on Programming Languages and Systems.

Conventional	Static Single Assignment
...	...
$i = 0$	$i_0 = 0$
$i = i + 1$	$i_1 = i_0 + 1$
$j = \text{func}(i)$	$j_0 = \text{func}(i_1)$
$i = 2$	$i_2 = 2$
...	...

CB Development Objectives

- meaningfully differentiate competitors; competitors are graded on:
 - *evaluation score*: ability to PoV
 - *security score*: ability to patch against known PoV(s)
 - *availability score*
 - we test this via *pollers*
 - each CS graded on **unique pairwise differentiation** among competitors
- challenges should **push the state of the art**
- (some) **emulate known vulnerabilities** difficult to find autonomously in popular software

What Goes into a CB?

- CB source & libs
- Makefile
- pollers: verify correct functionality
- PoVs: prove (intended) vulnerabilities
- README.md

depicted:

- single-CB CS
- 2 PoVs (written as programs)
- CFE poll generation script & state graph DAG

```
NRFIN_00074
├── lib
│   ├── libc.c
│   ├── libc.h
│   └── list.h
└── malloc.c
├── Makefile
├── poller
│   └── for-release
│       ├── machine.py
│       └── state-graph.yaml
├── pov_1
│   └── pov1.c
├── pov_2
│   └── pov2.c
└── README.md
└── SIC
    └── service.c
        └── service.h
```

CB Inspiration: Privesc



[Towelroot logo @geohot](#)

- CVE-2014-3153: Linux kernel `futex()` bug (Comex found; Geohot Towelroot) -> NRFIN_00006
- CVE-2012-0809: `sudo` format string bug -> NRFIN_00053
- CVE-2013-3881: NULL pointer dereference in `win32k.sys` -> NRFIN_00040
- `INTERPRETTTHIS` / NRFIN_00037 is privesc-related; has memory range privilege separation
- **vuln type focus:** NULL pointer derefs, (CWE-476), (un)signed conversion errors (CWE-195, CWE-196), return of pointer value outside of range (CWE-466), access to uninitialized pointers (CWE-824), race conditions (CWE-362), TOCTOU (CWE-367), many more ...



CB Inspiration: Server-Side

- CVE-2008-4250 (aka **MS08-067**): Conficker bug; buffer overflow in path canonicalization in Windows SMB -> **GREY MATTER / NRFIN_00011**
[Heartbleed logo](#)
- CVE-2014-0160 (aka “**Heartbleed**”): OpenSSL memory disclosure; OpenSSL custom heap means this wasn’t found with static checkers previously
 - many of our CBs includes custom heap implementations to deter *a priori* modelling
- strategy:
 - focus on Type 2, some allow for Type 1
 - PoV type 1 via reference count errors
 - test CRS ability to reason about state across CBs in a CS



CB Inspiration: Client-Side

V8 logo

- CVE-2014-1705: **V8 intrinsics bug** (Geohot) -> NRFIN_00051
- CVE-2011-0226: **JailbreakMe** (Comex)'s FreeType vuln
 - int overflow -> RCE: **NRFIN_00039**
- **vuln type focus**: use-after-free (CWE-416), type confusion (CWE-843), out-of-bounds accesses (CWE-129), dangling pointers (CWE-401)
 - type confusion: NRFIN_00023
- PA challenges: aliased memory accesses, concolic handoffs

```
var ab = new ArrayBuffer(8);
ab.__defineGetter__("byteLength", function() { return 0xFFFFFFFFC; });
var aaa = new Uint32Array(ab);
// aaa[0x1234567] = 1;
```

Simple PoC for V8
intrinsics vuln
(CVE-2014-1705)

Narf's CB Design Goals

- **manual**:
 - emulate real-world vulnerabilities, CWEs
 - focus on edges of state of the art:
 - **checksums** / hashes, casting, concolic handoff
 - find unintentionals via CI: **AFL + Clang** checkers (custom)
 - break assumptions: heap, **calling conventions**, NULL termination of strings ...
- **automatic**: sliding scale of difficulty
 - **cyclomatic** complexity (deter lifting, symex)
 - CFG / DDG / CDG complexity (deter lifting, symex)
 - excessive **indirection** (like Obj-C)
 - control flow **flattening**
 - **superfluous** / **non-contiguous memory access** patterns

TAINTEDLOVE NRFIN_00019

- mice trying to take over the world via a doomsday device
- CRS' job: deny/degrade/delay the mice's TI-83 calculator they're using to create a doomsday device
- this CB:
 - progressively difficult taint tracking challenges*
 - each challenge is preceded by "gating" logic intended to force symbolic reasoning



[@iannerboy62 on Unsplash](#)

TaintedLove

NRFIN_00019

```
size_t GATE_MAGIC = 0xCAFEBABE;
... gate read directly off the wire, compared to
constant ...
```

```
// Do complex & irrelevant operations with the majority of the input.
// Throw result out.

size_t garbage = 1;
for (size_t i = 1; i < SZ_IRRELEVANT_BYTES; i++) {
    garbage = (rx_buf[i] ^ (rx_buf[i+1] + rx_buf[i+2])) /
        (garbage | 1);
}
```

... garbage is never used after this point ...

#1: entanglement_razzmatazz()

the vuln: not-page-aligned accesses can go out of bounds

```
for (i = 0; i < len-2; i++) {
    // OOB access here when allocation is not page aligned.
    // (When sz_alloc != sz_alloc_align)
    map[i+2] = map[i+1] + map[i] + i;
}
```

PA challenge (Overtaint):
 Ignore / Throw Out (Most) Input ->
 Conditionally branch to vulnerable code based on retained input.

TAIANTEDLOVE NRFIN_00019

```
#define SZ_VULN_BUF 20
unsigned char vuln_buf[SZ_VULN_BUF] = {0};
... sanitize all bytes but multiples of 37 ...
unsigned char GATE_MAGIC = 0xAA;
... compare against the sum of the (unsanitized)
bytes on the wire ...

the vuln: bytes are used as index into vuln_buf;
unsanitized can access outside of buffer -> crash
for (i = 0; i < len-2; i++) {
    // OOB access here when requested allocation is not page aligned.
    // (When sz_alloc != sz_alloc_align)
    map[i+2] = map[i+1] + map[i] + i;
}
```

#2: causality_poppycock()

(Overtaint) Sanitize Input

1. Take a lot of binary input
2. Sanitize most input
3. Do complex, seemingly dangerous operations with the sanitized input
 - things that would be dangerous if it weren't sanitized.
4. Vuln triggered via unsanitized input

PA challenge: terminate taint on subset of input that was sanitized

TAINTEDLOVE

NRFIN_00019

#3 & 4: heisenberg_hooey()
& relativistic_jabberwock()

`heisenberg_hooey()`: (Overtaint & Undertaint)

Terminate or Propagate Taint Unusually

- Constify via taint XORed with itself.
- Constify / propagate via float / integer conversion semantics.
- Constify / propagate via syscalls (reflect the same or a constant value from a syscall).

`relativistic_jabberwock()`: (Overtaint & Undertaint) Symbolic Memory Accesses

- Store / load with symbolic addresses.
- Test CRS' ability to propagate (or not) taint through aliased (or not) memory accesses.
- Challenge: precision in the face of (non-)aliased memory accesses

TAINTEDLOVE

NRFIN_00019

how did competitors* fare?

Competitor	Score	PoV	Defense	Consensus Eval
ForAllSecure	2.25	PoV'ed #2	1 / 4 PoVs	no
TrailofBits	2.25	PoV'ed #2	1 / 4 PoVs	no
DeepRed	2.25	PoV'ed #1	1 / 4 PoVs	no
CodeJitsu	0.65	PoV'ed #1	4 / 4 PoVs	yes
Disekt	0.56	PoV'ed #1	1 / 4 PoVs	no

* Only competitors with non-zero scores are displayed.

CableGrind

NRFIN_00026

- procedurally generated CB with templates of functions that called one another
- think **Wireshark** and its hundreds of dissectors written in C
- we fixed a bug late in this CB late in development, forgot to deliver this fix to DARPA
- this bug was in the procedural code ... and it produced a vulnerability



[@robgun on unsplash](#)

CableGrind

NRFIN_00026

- procedurally generated CB with templates of functions that called one another
- think Wireshark and its hundreds of dissectors written in C
- we fixed a bug late in this CB late in development, forgot to deliver this fix to DARPA
- this bug was in the procedural code ... and it produced a vulnerability
- ... **180 times**



[@robgun on unsplash](#)

CableGrind

NRFIN_00026

- procedurally generated CB with templates of functions that called one another
- think Wireshark and its hundreds of dissectors written in C
- we fixed a bug late in this CB late in development, forgot to deliver this fix to DARPA
- this bug was in the procedural code ... and it produced a vulnerability
- ... **180 times** which means we needed **181 reference PoVs**



[@robgun on unsplash](#)

CableGrid NRFIN_0

- procedurally generated
functions that called
- think **Wireshark** and its
dissectors written in C
- we fixed a bug late in
development, forgot to
DARPA
- this bug was in the pre-
produced a vulnerability
- ... **180 times** which means

- Author-supplied POVs: POV1, POV2, POV3, POV4, POV5, POV6, POV7, POV8, POV9, POV10, POV11, POV12, POV13, POV14, POV15, POV16, POV17, POV18, POV19, POV20, POV21, POV22, POV23, POV24, POV25, POV26, POV27, POV28, POV29, POV30, POV31, POV32, POV33, POV34, POV35, POV36, POV37, POV38, POV39, POV40, POV41, POV42, POV43, POV44, POV45, POV46, POV47, POV48, POV49, POV50, POV51, POV52, POV53, POV54, POV55, POV56, POV57, POV58, POV59, POV60, POV61, POV62, POV63, POV64, POV65, POV66, POV67, POV68, POV69, POV70, POV71, POV72, POV73, POV74, POV75, POV76, POV77, POV78, POV79, POV80, POV81, POV82, POV83, POV84, POV85, POV86, POV87, POV88, POV89, POV90, POV91, POV92, POV93, POV94, POV95, POV96, POV97, POV98, POV99, POV100, POV101, POV102, POV103, POV104, POV105, POV106, POV107, POV108, POV109, POV110, POV111, POV112, POV113, POV114, POV115, POV116, POV117, POV118, POV119, POV120, POV121, POV122, POV123, POV124, POV125, POV126, POV127, POV128, POV129, POV130, POV131, POV132, POV133, POV134, POV135, POV136, POV137, POV138, POV139, POV140, POV141, POV142, POV143, POV144, POV145, POV146, POV147, POV148, POV149, POV150, POV151, POV152, POV153, POV154, POV155, POV156, POV157, POV158, POV159, POV160, POV161, POV162, POV163, POV164, POV165, POV166, POV167, POV168, POV169, POV170, POV171, POV172, POV173, POV174, POV175, POV176, POV177, POV178, POV179, POV180, POV181



@robgun on unsplash

Vs

CableGri NRFIN_0

- procedurally generated functions that called

- Author-supplied POVs: POV1, POV2, POV3, POV4, POV5, POV6, POV7, POV8, POV9, POV10, POV11, POV12, POV13, POV14, POV15, POV16, POV17, POV18, POV19, POV20, POV21, POV22, POV23, POV24, POV25, POV26, POV27, POV28, POV29, POV30, POV31, POV32, POV33, POV34, POV35, POV36, POV37, POV38, POV39, POV40, POV41, POV42, POV43, POV44, POV45, POV46, POV47, POV48, POV49, POV50, POV51, POV52, POV53, POV54, POV55, POV56, POV57, POV58, POV59, POV60, POV61, POV62, POV63, POV64, POV65, POV66, POV67, POV68, POV69, POV70, POV71, POV72, POV73, POV74, POV75, POV76, POV77, POV78, POV79,



Branch: master ▾

[cgc-challenge-corpus](#) / NRFIN_00026 / src / **cablegrindprotos.c**[Find file](#) [Copy path](#) demoray initial commit

b65ebe8 on Apr 5, 2017

1 contributor

1.41 MB

[Download](#) [History](#) [View raw](#)

(Sorry about that, but we can't show files that are this big right now.)

POV170, POV171, POV172, POV173, POV174, POV175, POV176, POV177, POV178, POV179, POV180, POV181

Ben Schmidt



CableGrind

NRFIN_00026

how did competitors* fare?

Competitor	Score	PoV	Defense	Consensus Eval
CSDS	1.78	no	181 / 181 PoVs	yes
Shellphish	1.00	no	1 / 181 PoVs	no
TrailofBits	0.76	yes	179 / 181 PoVs	yes

* Only competitors with non-zero scores are displayed.

Other Notable CQE Mentions

- **REDPILL / NRFIN_00006**
 - emulates the Linux Kernel **futex()** bug
 - CVE-2014-3153 by Comex, Towelroot by Geohot
 - 6 CBs doing IPC
 - PoV'ed by: ForAllSecure and FuzzBOMB
 - PA challenges: concurrency, stack modelling
- **GREY MATTER / NRFIN_00011**
 - emulates **MS08-067**
 - PoV'ed by: ForAllSecure, DeepRed
 - PA challenge: strings are not NULL-terminated; state reasoning across calls to `strncat()`

Other Notable CQE Mentions

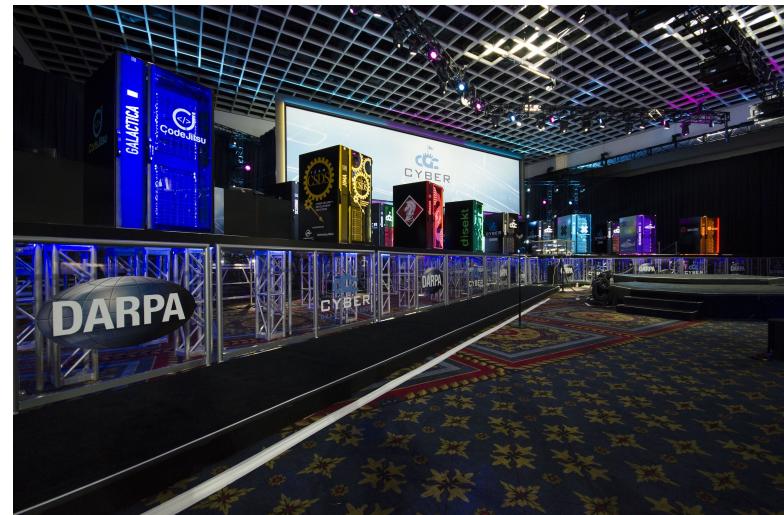
- **SOMECHECKS / NRFIN_00016:** increasingly difficult PA challenges
 - level 1: (un)signed conversion issue
 - level 2: simple checksums
 - level 3: close to real CRC (with new default values)
 - level 4: floating point arithmetic
 - level 5: a simple hash function presented (answer: “admin”)
 - level 6: same as 5, except overflow is in a global buffer
- **GREATVIEW / NRFIN_00042:** “memory as a service”
 - constrained RW by design, very difficult to patch
- **INTERPRETTHIS / NRFIN_00037:**
 - inspired by NaCl, when everyone was blissfully ignorant of Spectre

Narf CQE Differentiation

availability_score						
cset	cqe_score	func_score	perf_score	sec_score	eval_score	
NRFIN_00001	18	11	19	6	12	
NRFIN_00016	21	10	20	19	10	
NRFIN_00017	21	6	18	17	10	
NRFIN_00024	20	11	19	10	12	
NRFIN_00035	21	6	20	6	12	
NRFIN_00037	0	0	17	10	12	
NRFIN_00039	15	6	19	17	12	
NRFIN_00040	6	11	14	10	12	

CQE & CFE Differences

- PoV type 2: leak memory from a flag page
 - report 4 contiguous bytes in flag page to win
- random() may alter control flow
- poll generator must produce a minimum of **1 million unique polls**
- JIT no longer prohibited
- **unintentional vulns aren't necessarily bad anymore**



CFE at DEF CON, image from DARPA

CQE & CFE Differences

- PoV type 2: leak memory from a flag page
 - report 4 contiguous bytes in flag page to win
- random() may alter control flow
- poll generator must produce a minimum of **1 million unique polls**
- JIT no longer prohibited
- **unintentional vulns aren't necessarily bad anymore ----->**

Only one Narf CB had an unintentional vuln during CFE.

Thompson, Michael F., and Timothy Vidas. "Cyber Grand Challenge (CGC) monitor: A vetting system for the DARPA cyber grand challenge." *Digital Investigation* 26 (2018): S127-S135.

Table 2
Author Intention of Exploited CFE Services: Of the 20 exploited services in CFE, half were exploited in ways unintended by the author of the vulnerable service. Six services included a vulnerability unknown to the author, yet discovered by competitors, and an additional four vulnerabilities found were intended, but exploited via unintended means.

CBID	As intended	Description
CROMU_00046	Y	
CROMU_00051	N	unintended path
CROMU_00055	N	unintended path
CROMU_00058	Y	
CROMU_00064	Y	
CROMU_00065	N	unintended path
CROMU_00073	N	unintended path
CROMU_00088	N	unintended vuln
CROMU_00094	Y	
CROMU_00095	N	unintended vuln
CROMU_00096	N	unintended vuln
CROMU_00097	N	unintended vuln
CROMU_00098	Y	
KPRCA_00065	N	unintended vuln
KPRCA_00094	Y	
NRFIN_00052	N	unintended vuln
NRFIN_00059	Y	
NRFIN_00063	Y	
YAN01_00015	Y	
YAN01_00016	Y	

Narf's CFE CBs

of the 21 Narf CBs fielded during CFE:

- **only 3 were PoV'ed**
 - we'll talk about these 3
 - we made the CBs too hard?
- **all competitors patched at least 1 CB**
- Shellphish patched all but 2 CBs
- CSDS only patched 2 CBs
- **ForAllSecure & CSDS had comparatively conservative patching strategies**



That one CB that everyone patched

Overflow Parking

NRFIN_00052



@omerrana on Unsplash

- intended PoV: Integer overflow leads to heap corruption (Type 1)
 - trigger allocation smaller than size of buffer, overflow buffer
- PA challenges:
 - *many* intentional integer overflows, some important to correct functionality
 - vuln in stdlib (but CRSs maybe don't care)
- our estimation of difficulty:
 - Discovering is **medium**
 - Proving is **hard**
 - Fixing is easy

```
// from malloc.c
#ifndef PATCHED_1
if (total > INT_MAX) {
    return NULL;
}
#endif
```

Overflow Parking

NRFIN_00052

- **everyone but CSDS submitted patches**
- final round PoV / patch analysis:
 - CodeJitsu -> CSDS (no patch), DeepRed (w/ patch), Disekt (w/ patch), others mitigated
 - DeepRed -> CSDS (no patch), Disekt (w/ patch), others mitigated
 - ForAllSecure -> CSDS (no patch), DeepRed (w/ patch), others mitigated
 - Shellphish -> CSDS (no patch), DeepRed (w/ patch), Disekt (w/ patch), others mitigated
- unintentional vuln? wrong about difficulty?
- perhaps we were **correct about it being easy to patch**
- everyone that scored, scored on **DeepRed's RB -> patch was ineffective**



[@omerrana on Unsplash](#)

Snail Mail

NRFIN_00059

- intended PoV: request a “stamp”; its serial number (3 bytes) is taken from flag page (Type 2)
- PA challenge: **combine 2 leaks to make 4 byte leak**

```
#ifndef PATCHED_1
    if (sizeof(stamp_t) != memcpy(s->serial, &seed[seed_idx], sizeof(stamp_t)))
        return NULL;
    seed_idx = (seed_idx + 3) % SEED_MAX;
#else
    for (int i = 0; i < sizeof(stamp_t); i++) {
        rand((char *)&seed_idx, 2);
        s->serial[i] = seed[seed_idx % SEED_MAX];
    }
#endif
```

patch TL;DR:
if patched,
return bytes
from random()



[@alex19perz on Unsplash](#)

estimated difficulty

- Discovering is **easy**
- Proving is **medium**
- Fixing is **medium**

Snail Mail

NRFIN_00059

final round PoV / patch analysis:

- **only ForAllSecure and Disekt's patches deterred PoVs**
- ForAllSecure -> CodeJitsu, CSDS, DeepRed, Shellphish, TECHx (all no patch)
- Shellphish -> CodeJitsu, CSDS, DeepRed, Disekt (w/ patch), TECHx
- **Disket's patch deterred ForAllSecure, but not Shellphish**
- **Shellphish could PoV but not patch**



[@alex19perz on Unsplash](#)

CFE: On Sale

NRFIN_00063

- intentional PoVs: heap, stack buffer overflows (Type 1)
- same bug, two different locations



[@jijiving on Unsplash](#)

```
#ifdef PATCHED_1
    bytes_recv = recv_until_delim_n(STDIN, DESC_TERM[0], (char *)p->desc, MAX_DESC_LEN);
#else
    bytes_recv = recv_until_delim_n(STDIN, DESC_TERM[0], (char *)p->desc, sizeof(Product));
#endif
```

```
#ifdef PATCHED_2
    bytes_recv = recv_until_delim_n(STDIN, DESC_TERM[0], desc_buf, MAX_DESC_LEN);
#else
    bytes_recv = recv_until_delim_n(STDIN, DESC_TERM[0], desc_buf, sizeof(Product));
#endif
```

estimated difficulty:

- Discovering is **easy**
- Proving is **medium**
- Fixing is **easy**

CFE: On Sale

NRFIN_00063

final round PoV / patch analysis::

- CodeJitsu -> CSDS, ForAllSecure (no patches)
 - deterred by: **Disekt**, Shellphish, TECHx
- Shellphish -> CodeJitsu (w/ patch), CSDS (no patch), ForAllSecure (no patch)
 - deterred by: **Disekt**
- **no one could PoV through Disekt's patches**
- Shellphish PoV'ed CodeJitsu's patch; CodeJitsu could not PoV through Shellphish's patch
- **CodeJitsu turned this Type 1 into Type 2 (leak memory shellcode) and cleanly exited**



[@jiving on Unsplash](#)

CFE: On Sale

NRFIN_00063

final round PoV / patch analysis::

- CodeJitsu -> CSDS, ForAllSecure (no patches)
 - deterred by: **Disekt**, Shellphish, TECHx
- Shellphish -> CodeJitsu (w/ patch), CSDS (no patch), ForAllSecure (no patch)
 - deterred by: **Disekt**
- **no one could PoV through Disekt's patches**
- Shellphish PoV'ed CodeJitsu's patch; CodeJitsu could not PoV through Shellphish's patch
- **CodeJitsu turned this Type 1 into Type 2 (leak memory shellcode) and cleanly exited ... or there was an unintentional Type 2 vuln**



[@jijiving on Unsplash](#)

Other Notable CFE Mentions

- **CAT / NRFIN_00051**
 - emulates remote RAM access to slot machines
 - PA challenge: difficult to reason about what is permitted access
- **CLOUDCOMPUTE / NRFIN_00046**: inspired by **LangSec** principles
 - implements a bytecode interpreter & type confusion
 - CRS needs to break out of interpreter for R/W over memory space
- **OUTLAW / NRFIN_00066**: inspired by **TLS CRIME (CVE-2012-4929)**
 - CRS is man-in-the-middle
 - Type 2: auth token leak via a compression oracle side-channel
 - Type 1: use auth token to reconfigure the service to jmp to an attacker-supplied address

Hackathon

- evaluated scoring design
- evaluated implementation:
 - DECREE kernel
 - userspace tooling
- learned that CFE wasn't going to be run on DECREE/Linux
 - it was going to be run on FreeBSD
- developed a spec for MULTI-PASS protocol, a bespoke protocol that other CB devs can re-use but have unique vulns in

Tuesday 10/7/2014:

- ❖ Franklin's BBQ Lunch
- Hack

Hack time

Wednesday 10/8/2014:

- ❖ There will be a team outing at a time and place TBD
- Hack

Hack time

Thursday 10/9/2014:

- Hack

Hack time

Friday 10/10/2014:

- Hack

Hack time

Schedule excerpt from one of the hackathons

Hackathon Evaluation Recommendations & Results

- **accepted**: replace tar with ar
- **accepted**: re-write ar in pure Python
- **rejected**: replace OpenSSL with NaCl / libsodium
- **DoS in Python Expat / LXML** -> cb-replay and poll-validate
 - **solution**: hard time limits applied to cb-replay, poll-validate
- **DoS in Python PCRE** -> linearly scales with complexity of regex
 - **solution**: hard time limits applied to cb-replay, poll-validate

Hackathon Evaluation Recommendations & Results 2

- **vulns in libcgcef / libcgcdwarf:**
 - many vulns; they were based on outdated libelf and libdwarf
 - vuln types: integer overflows -> heap overflows, type confusions
 - some reachable in cgcef-verify; most reachable in readcgcef
 - per (limited) analysis, game infrastructure RCE seemed possible
- **recommendation:** re-write (subset of) libs in pure Python or apply seccomp() filters
- **solution:** rebase on more recent libelf / libdwarf; addressed all observed vulns

Hackathon Evaluation Recommendations & Results 3

- **hardening recommendations** (all fixed):
 - many critical-path ELF's lacked PIE -> not full ASLR protection
 - many critical-path ELF's lacked stack protection
 - recommended Clang upgrade for --fstack-protector-strong
 - many critical-path ELF's have a writable GOT (RELRO now)
 - seccomp(); drop privs and change to nobody everywhere possible
- **we ported GRsecurity / PaX to DECREE kernel ("Frakenpatch")**
 - this ended up not being used because CFE would be on FreeBSD
- **rebased**: DECREE kernel was vulnerable to futex() bug (NRFIN_00006)

```
tk@tk-desktop: ~/Desktop/checksec$ ./checksec.sh --dir /sbin/
RELRO      STACK CANARY    NX      PIE           FILE
Partial RELRO No canary found NX enabled No PIE          /sbin/acpi_available
No RELRO   No canary found NX enabled No PIE          /sbin/activate
No RELRO   No canary found NX enabled Not an ELF file /sbin/alsa
Partial RELRO Canary found  NX enabled No PIE          /sbin/alsactl
Partial RELRO No canary found NX enabled No PIE          /sbin/apm_available
Partial RELRO Canary found  NX enabled No PIE          /sbin/apparmor_parser
Partial RELRO Canary found  NX enabled No PIE          /sbin/badblocks
Partial RELRO Canary found  NX enabled No PIE          /sbin/blkid
Partial RELRO Canary found  NX enabled No PIE          /sbin/blockdev
Partial RELRO Canary found  NX enabled No PIE          /sbin/brltty
Partial RELRO Canary found  NX enabled No PIE          /sbin/brltty-setup
Partial RELRO Canary found  NX enabled No PIE          /sbin/cfdisk
Partial RELRO No canary found NX enabled No PIE          /sbin/ctrlaltdel
Partial RELRO Canary found  NX enabled No PIE          /sbin/debugfs
No RELRO    Canary found  NX enabled No PIE          /sbin/debugreiserfs
Partial RELRO Canary found  NX enabled No PIE          /sbin/depmod
Partial RELRO Canary found  NX enabled PIE enabled     /sbin/dhclient
Partial RELRO Canary found  NX enabled PIE enabled     /sbin/dhclient3
Partial RELRO No canary found NX enabled Not an ELF file /sbin/dhclient-script
Partial RELRO Canary found  NX enabled No PIE          /sbin/dmsetup
Partial RELRO Canary found  NX enabled No PIE          /sbin/dosfsck
Partial RELRO Canary found  NX enabled No PIE          /sbin/dosfslabel
Partial RELRO Canary found  NX enabled No PIE          /sbin/dump2fs
Partial RELRO Canary found  NX enabled No PIE          /sbin/e2fsck
Partial RELRO Canary found  NX enabled No PIE          /sbin/e2image
Partial RELRO Canary found  NX enabled No PIE          /sbin/e2label
Partial RELRO Canary found  NX enabled No PIE          /sbin/e2undo
Partial RELRO Canary found  NX enabled No PIE          /sbin/fdisk
Partial RELRO Canary found  NX enabled No PIE          /sbin/findfs
Partial RELRO Canary found  NX enabled No PIE          /sbin/fsck
Partial RELRO No canary found NX enabled No PIE          /sbin/fsck.cramfs
Partial RELRO Canary found  NX enabled No PIE          /sbin/fsck.ext2
Partial RELRO Canary found  NX enabled No PIE          /sbin/fsck.ext3
```

checksec.sh screenshot (not taken in DECREE VM)

Hackathon Findings

Narf won the “breakit” track award via this chain:

- if you submit an ELF as an RB, infrastructure would run it
- .ELFs were **not syscall-sandboxed**
 - **unlike DECREE bins**
- passwordless sudo means **no user / root security boundary**
- Linux means **no root / kernel security boundary**
- **TL;DR: submitting an ELF as RB gives you kernel exec in VM**



The “breakit” token

In the background: the most interesting InfoSec conference attendee in the world

CAWS: CGC Architecture Wrecking System 1/4



CAWS, 2015, colorized

[@vitreous_macula on Unsplash](#)

- automatic efforts:
 - continuous integration (CI) testing using:
 - AFL
 - Clang checkers (including some custom checkers we wrote)
- manual efforts:
 - RBs stressors
 - infrastructure edge cases

CAWS: CGC Architecture Wrecking System 2/4



- integrated AFL into the CB testing pipeline because unintended vulns were bad (for CQE)
 - **AFL used to diversify poller corpora**
 - found **7 unintended vulns** in our own corpus and **9 unintended vulns** in the SE-II set
- modified CBs to strip out sleep() and random() -> speed++ & determinism++
- modified DECREE build scripts to **produce polls that work as AFL seeds**
- in CBs with AFL-found vulns:
 - we performed a **minimum of 150k** and **maximum of 27M** executions
 - **AFL exhausted after ~ 8 hours of fuzzing / CB** - no bugs found after

CAWS: CGC Architecture Wrecking System 3/4



- “stress” infrastructure with high-consumption (RAM, CPU, network) RBs
- implemented Clang/LLVM checkers for unintentionals; we found:
 - NULL dereferences
 - bad malloc() usages
 - out of bound accesses
 - memory leaks
 - uses of uninitialized memory
 - cases of improper NULL termination
 - cases of ignoring tx bytes/rx bytes from receive/transmit

CAWS: CGC Architecture Wrecking System 4/4



- run Clang checkers on candidate CQE CBs that were “patched”:
 - 164 CBs + 21 checkers -> **994 bugs before triage**
 - **95 serious issues across 50 CBs** (unintended vuln)
 - 31 minor issues (wouldn’t crash, but affected design of the CBs)
 - **109 bug reports to other teams**
- we shared AFL + Clang checker rig as .debs for other CB developers to use in their CI
 - found: 347 intentional vulns + 243 unintentional = 590 vulns
 - 180 unintentionals were in CableGrind



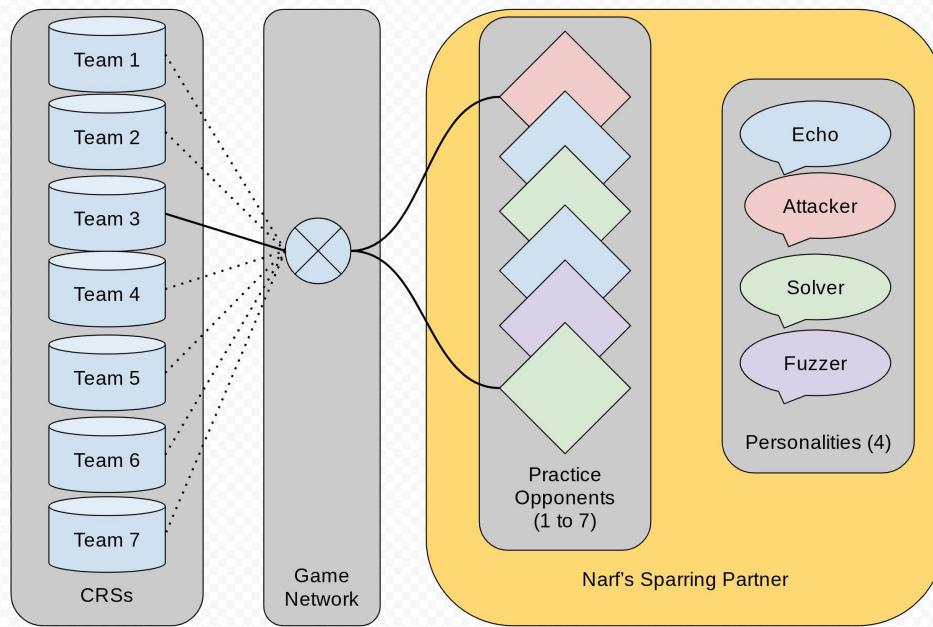
Competition Integrity

- “competed” as human team in SE-II, provided human baseline
- RE’ed PoVs and RBs from CQE:
 - look for any cheating (human intervention)
 - **no smoking gun indications of cheating**
 - several cases where we had insufficient data
 - **only 3 of the teams were able to make progress on some of the more difficult CBs**
 - **only 2 CQE teams performed any substantial code re-writing**

[Lady Justice](#)

Sparring Partner “Hadouken”

Sparring Partner presents 1 CRS with 1 to 7 Practice Opponents each having 1 of 4 Personalities



Hadouken: A mock CRS for competitors to “spar” against during SE-II and run-ups to CQE & CFE

Sparring Partner “Hadouken”



- v1: simply submit **unpatched & patched** CBs
 - Hadouken had CB source code (*we found the cheater!*)
- v2: “cheeseball” + legitimate compile-type & run-time defenses
- “cheeseball” defenses in Hadouken’s RBs:
 - (custom) nop out free()
 - (custom) nop out rand()
 - **(custom) triple allocate**
 - (custom) memory border checks

Sparring Partner “Hadouken”

Hadouken cheated; had access to source code:

1. `alloca()` / stack frame padding
2. stack cookie injection
3. variable reordering
4. `-fsanitize=bounds,null`
5. UBSan
6. various CFLAGS to cause PoV breakage ----->



```
-O3
-freg-struct-return
-fpack-struct=32
-fpcc-struct-return
-fshort-enums
-ftrap-function=_terminate
-funroll-loops
-mno-global-merge
-fomit-frame-pointer
-mrelax-all
-mstack-alignment=64
-mstackrealign -Wall
-Wno-overlength-strings
-Wno-packed
```

CFE Was Really Interesting*

- “stealth” PoVs (get RCE, leak bytes, cleanly exit) did not help
 - DeepRed & Disekt: RCE to `_terminate()`
 - CodeJitsu: ROP to `_terminate()`
 - ForAllSecure: RCE to `jmp eip`
 - tight loop, potentially affecting availability of the victims’ other CBs
- DeepRed deployed a honeypot in their RBs (more info: [1](#), [2](#))
 - honeypot RBs had trivial stack buffer overflow that was not exploitable on game infrastructure
 - CodeJitsu fell for it (but should have known due to game feedback)

CFE Was Really Interesting*

- **Shellphish threw 0day DoS in QEMU**, which hung visualization software
- several competitors threw **backdoor'ed RBs** (would leak Type 2 via pseudo-crypto handshake)
- **Shellphish & TECHx did pseudo forward-edge CFI in RBs**
- TECHx also did psuedo forward and backward-edge CFI
- of the 20 PoV'ed CBs in CFE, 10 were unintentional PoVs / **unintentional paths**
- in total, 5/7 teams used ROP during competition

CFE Was Really Interesting*

NRFIN_00063 differentiated CRS' generic patching methodology:

1. CodeJitsu got EIP; used (obfuscated) payload to PoV Type 2 via `transmit()`
 - o obfuscation prevents IDSing on the magic page
2. CodeJitsu payload crashed against DeepRed
3. DeepRed responded by **fielding an RB with NX stack**
4. **CodeJitsu deployed ROP**, scored again against DeepRed
 - o ROP payload doesn't crash the service, but CRSs have already seen it crash

CGC Impact

- **then**: machines competed against one another
- **now**: machines help humans find vulns and play CTFs
- **future**: we don't have a job

CGC Impact

- tech going in:
 - **lifters**: BAP (semi-private; For All Secure), BitBlaze (semi-private; UC Berkeley), Valgrind (public), QEMU/TCG (public)
 - **fuzzing**: AFL (public)
 - **dynamic / tracing / emulation**: PIN, QEMU (both public)
 - **SAT/SMT solving**: Z3, STP (both public)
 - **concolic**: Mayhem (private; For All Secure) *This is not a complete list!*
- tech coming out:
 - **lifters**: McSema (Trail of Bits), CLE (Shellphish)
 - **fuzzing**: grr (Trail of Bits)
 - **dynamic / tracing / emulation**: PySymEmu (Shellphish)
 - **concolic**: angr (Shellphish)
 - **full CRS**: Mechaphish (Shellphish), S2E (CodeJitsu)

What We Would Change for CGC 2.0

1. rebalance scoring weights

- replacing binaries = too expensive,
getting pwned = not expensive enough
 - 1 round to patch + 2 rounds to realize
it was bad + 1 round to revert patch =
4 rounds lost
- *arguably the best strategy was to never
proactively patch*



hacker voice "I'm in." *hacker voice*
- this guy, probably

[@neonbrand on Unsplash](#)

What We Would Change for CGC 2.0

2. **eliminate wall clock**: PoVs & pollers had 15s deadline

- *wall clocks are non-deterministic*; instead do:
 - hardware counters
 - need weights, x86 is a big ISA
 - dynamic tracing (e.g. PIN)
 - trade-offs: security, fidelity issues
 - emulation
 - trade-offs: security, fidelity issues



hacker voice "I'm in." *hacker voice*
- this guy, probably

[@neonbrand on Unsplash](#)

3. **eliminate IDS**; not used much, stretched competition too thin

A Selection of Publications by Other Performers

Competitors (names used in this presentation are bolded and sorted alphabetically):

- (CFE Qualifier) "GALACTICA" (**CodeJitsu** - UC Berkeley, Cyberhaven, and Syracuse): [Cyber Grand Challenge and CodeJitsu](#)
- (CFE Qualifier) "JIMA" (**CSDS** @ University of Idaho): [part 1](#), [part 2](#)
- (Placed 1st in CFE) "Mayhem" (**ForAllSecure**): [Blog](#), [Original Mayhem Paper](#)
- **FuzzBOMB** (SIFT, University of Minnesota): [FUZZBOMB: Autonomous Cyber Vulnerability Detection and Repair](#)
- Cyberdyne" (**Trail of Bits**): [Blog](#), [Toward Smarter Vulnerability Discovery Using Machine Learning, The Past, Present, and Future of Cyberdyne](#)
- (Placed 3rd in CFE) "MechaPhish" (**Shellphish**): [Phrack](#)
- (Placed 2nd in CFE) "TECHx" / "Xandra" (GrammaTech + University of Virginia): [Blog](#)

White Team

- (2015) Walker & Wiens: [Machine vs. Machine: Inside DARPA's Fully Automated CTF](#)
- (2015) Walker: [Machine vs. Machine: Lessons from the First Year of Cyber Grand Challenge](#)
- (2015) Price & Zhivich: [DARPA's Cyber Grand Challenge: Creating a League of Extra-Ordinary Machines](#)
- (2018) Thompson & Vidas: [CGC Monitor: A Vetting System for the DARPA Cyber Grand Challenge](#)
- (2018) The DECREE Team*: [DECREE: A Platform and Benchmark Corpus for Repeatable and Reproducible Security Experiments](#)
- ... many more



Thank You!

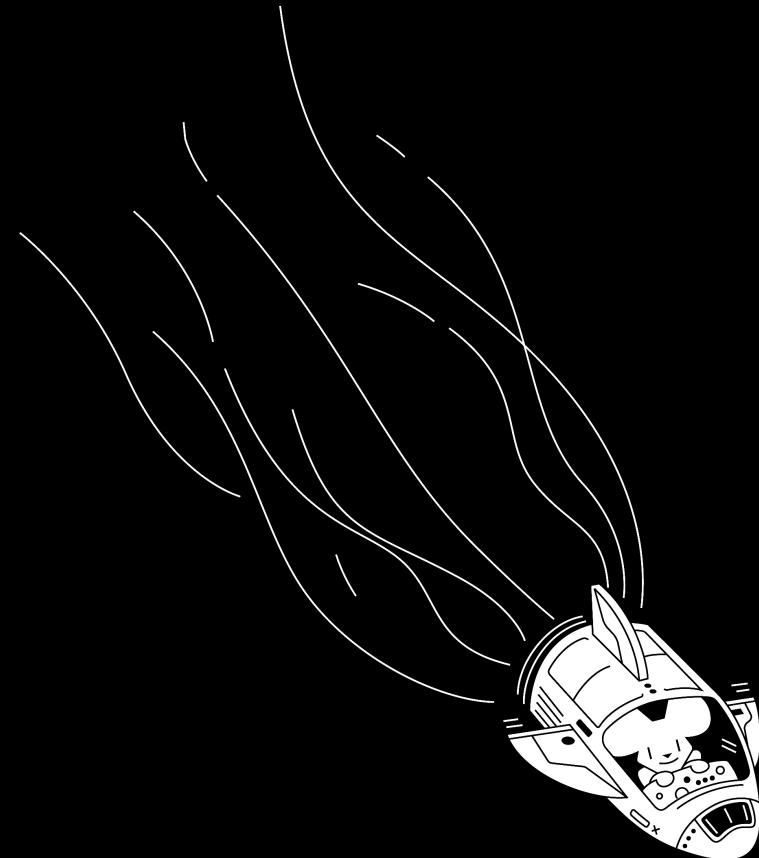
Questions?

A DARPA Cyber Grand Challenge
White Team Retrospective

Presented by: Paul Makowski



Bonus Slides



CQE: TAINTEDLOVE

NRFIN_00019

#3: heisenberg_hooey()
XOR games

```
size_t GATE_MAGIC = 0xDEADBEEF;  
... gate compared against two bytes in input ...  
  
////////// XOR  
  
// XOR, Easy: Constify via taint XORed with itself.  
rx_buf[XOR_CONST_OFF] =  
    rx_buf[XOR_CONST_OFF] ^ rx_buf[XOR_CONST_OFF];  
  
// XOR, Medium: now the constified index is symbolic.  
// The taint engine must properly alias memory in the trivial case.  
rx_buf[1+rx_buf[XOR_CONST_OFF_PTR]] =  
    rx_buf[1+rx_buf[XOR_CONST_OFF_PTR]] ^  
    rx_buf[1+rx_buf[XOR_CONST_OFF_PTR]];
```

```
// XOR, Medium-ish: the constified memory is symbolic, but it's not  
// immediately obvious that the two memory accesses are aliased.  
rx_buf[1+rx_buf[XOR_CONST_OFF_PTR*2]] =  
    rx_buf[1+rx_buf[XOR_CONST_OFF_PTR*2]] ^  
    rx_buf[1+rx_buf[(XOR_CONST_OFF_PTR*2)+0xFFFFFFFF+1]]
```

CQE: TAINTEDLOVE

NRFIN_00019

#3: heisenberg_hooey()
float games

```
/////////// FLOAT

// Constify via float
float infinity = 1.0 / 0.0;
// NOTE: this should always constify to 0, but *technically* the
// behavior is undefined.
rx_buf[FLOAT_CONST_OFF] =
    (unsigned char)((float)rx_buf[FLOAT_CONST_OFF] * infinity);

// Propagate via float
// This doesn't have to be complicated because many analysis programs
// can't handle floats at all.
// REM: I have confirmed this retains value in simple test.
rx_buf[FLOAT_PROP_OFF] =
    2*(unsigned char)((float)rx_buf[FLOAT_PROP_OFF] * 1.0);
```

CQE: TAINTEDLOVE NRFIN_00019

```
/////////// FLOAT

// Constify via float
float infinity = 1.0 / 0.0;
// NOTE: this should always constify to 0, but *technically* the
// behavior is undefined.
rx_buf[FLOAT_CONST_OFF] =
    (unsigned char)((float)rx_buf[FLOAT_CONST_OFF] * infinity);

// Propagate via float
// This doesn't have to be complicated because many analysis programs
// can't handle floats at all.
// REM: I have confirmed this retains value in simple test.
rx_buf[FLOAT_PROP_OFF] =
    2*(unsigned char)((float)rx_buf[FLOAT_PROP_OFF] * 1.0);
```

#3: heisenberg_hooey()
float games

I super over-engineered this ...

CQE: TAINTEDLOVE

NRFIN_00019

#3: heisenberg_hooey()
syscall games

```
////////////////// SYSCALL

// Constify via syscall: deallocate()
// Because rx_buf[CONST_SYSCALL_OFFSET] must be in [0, 255],
// any deallocation would fail, resulting in a write of EINVAL
// into rx_buf[1+CONST_SYSCALL_OFFSET], constifying it.
// Failsafe: deallocate() of length 0 should cause EINVAL.
rx_buf[SYSCALL_CONST_OFF] =
    deallocate((void *)rx_buf[SYSCALL_CONST_OFF], 0);
```

```
// Propagate (reflect) via syscall: random()
// We use a tainted value as the count argument to a random() syscall.
// If the random() syscall is successful, count number of random bytes
// will be written into a garbage buffer and ignored, but rnd_bytes will
// take on the same value as count.
size_t rnd_bytes = 0;
if (SUCCESS != (ret = random(&garbage, rx_buf[SYSCALL_PROP_OFF],
&rnd_bytes))) {
    ret = ERRNO_RANDOM;
    goto _bail_hooey;
}
```

```
// The actual number of returned bytes is written back into the tainted
// memory location, amounting to a reflection of the taint via a syscall.
rx_buf[SYSCALL_PROP_OFF] = 1+(2*(unsigned char)rnd_bytes);
```

CQE: TAINTEDLOVE

NRFIN_00019

```
// The following rx_buf indices should be PROPAGATED:  
//     FLOAT_PROP_OFF  
//     SYSCALL_PROP_OFF  
  
// There is potential these have been undertainted.  
// We use these and other bytes as a gate to guard  
// vulnerability, forcing symexec reasoning on these  
// potential undertaints.  
  
size_t gate =    rx_buf[EF_OFF]          <<  0 | // EF  
                 rx_buf[BE_OFF]          <<  8 | // BE (even)  
                 rx_buf[SYSCALL_PROP_OFF] << 16 | // AD (odd)  
                 rx_buf[FLOAT_PROP_OFF]  << 24; // DE (even)
```

```
// Vuln: if the GATE_MAGIC is satisfied, mult is 0, otherwise 1.  
// We attempt a write to rx_buf[VULN_OFF] * mult, which will cause an  
// attempt to write to the NULL page if the gate is satisfied.  
  
size_t mult = !(GATE_MAGIC ^ gate);  
unsigned char *ptr =  
    (unsigned char *)((size_t)&(rx_buf[VULN_OFF]) * mult);  
  
#ifdef PATCHED  
    if (0 != mult) { *ptr = 0x42; }  
#else  
    *ptr = 0x42;  
#endif  
}
```

#3: heisenberg_hooey()
the vuln