narf

Behind DARPA's CGC

A DARPA Cyber Grand Challenge
White Team Retrospective

Presented by: Paul Makowski

narf

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

narf

This picture taken after CQE, where we:

- manually RE'ed RBs to ascertain human involvement
- audited infrastructure
- wrote additional reference PoVs (more on this later)

narf

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

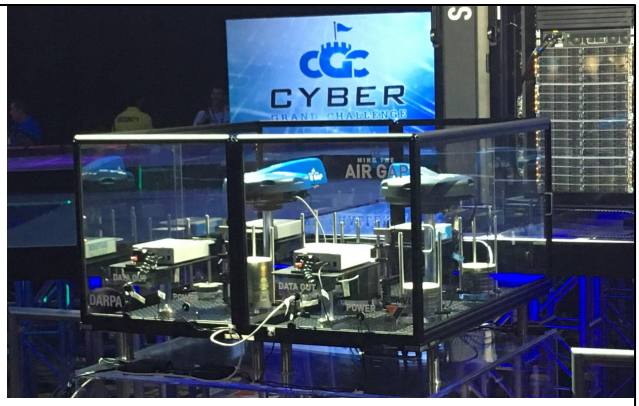
human CTF team during SE-II

narf

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



Credit: Sean Gallagher (@thepacketrat)

- 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)

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)

narf

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

Top 7 teams moved on from CQE to CFE.

narf

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

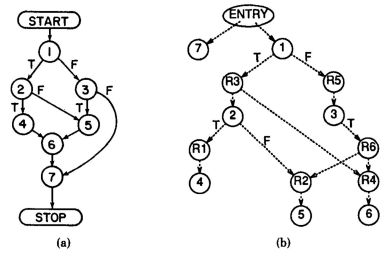


Fig. 1. A control flow graph and its control dependence subgraph.

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

C. Scott Ananian 1998-10-12

Some definitions / info:

- CFG: control flow graph
- DDG: data dependency graph, talk about forward / backward slicing in Ghidra
- CDG: code dependency graph
- PDG: a combination of DDG & CDG
- SAT & SMT solvers
  - SAT: the problem of determining if there exists an interpretation that satisfies a given Boolean formula. In other words, it asks whether the variables of a given Boolean formula can be consistently replaced by the values TRUE or FALSE in such a way that the formula evaluates to TRUE
    - Boolean algebra is the branch of algebra in which the values of the variables are the truth values true and false, usually denoted 1 and 0

        - respectively
    - SMT: Examples of theories typically used in computer science are the theory of real numbers, the theory of integers, and the theories of various data structures such as lists, arrays, bit vectors and so on.
        - basically a generalization of SAT solving; SMT solvers solve a superset of SAT solvers

Other notes:

- Valgrind was another lifter

narf

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

Ideally, each CB would be PoV'ed by ~50% of competitors and which competitors PoV'ed (and successfully defended) would vary across CBs.

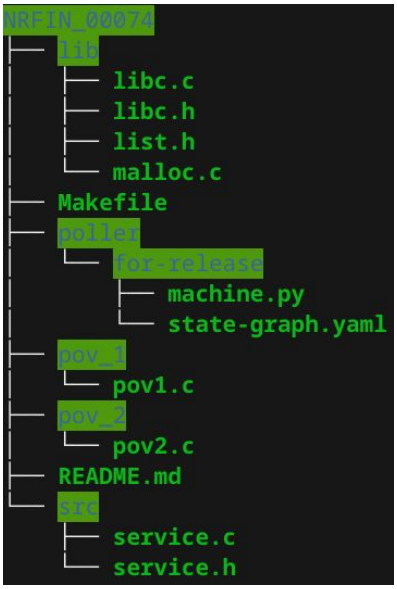
narf

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



This is an example of a CFE CB, so it contains a Directed Acyclic Graph (DAG) of program states (state-graph.yaml) and a Python file describing how to traverse these states and with what frequency (weight).

narf

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
- INTERPRETTHIS / 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 …

- futex() bug was exposed within Chrome seccomp sandbox
- yeah, sudo had a format string vulnerability in 2012, it wasn't just VPN appliances
- NULL pointer dereference bugs aren't exploitable in major consumer OSes anymore, since macOS dropped 32bit support

Notes from our proposal to DARPA:

- CBs seek to **emulate user to kernel exploitation**, where the user is typically able to allocate anywhere in the lower half of virtual address space.

- Reasoning about NULL ptr dereferences demands an inversion of the typical exploit formula: rather than trying to cause an indirect call through an address you control, you need to control the address that the indirect call will reference.
- **The most difficult binaries will leverage a race condition bug (CWE-362), causing a zeroing of an in-use object and a parallel trigger of a NULL page reference.**
- TOCTOU bugs are commonly found in modern kernels, which offer rich attack surfaces and numerous means of interaction. Easily exploitable bugs arise when kernel code implicitly trusts user-supplied input. **Tarjei Mandt showed that Windows' callback interface frequently fails to re-check state following user actions taken during a callback (CWE-374, CWE-822).** This TOCTOU failure opens the door to bugs like: use-after-free (CWE-416), improper resource locking (CWE-413), external control of critical state data (CWE-642), and untrusted pointer dereference (CWE-822) - all of which provide

- exploit primitives.
    - Emulated via: protocol in which a struct that contains a pointer in CB address space is serialized and sent to CRSs. CRSs are expected to send this struct back (plus data) with modifications to the trusted pointer to have the CB consult the wrong address. Method 1 proof-of-vulnerability demonstrates code execution, whereas Method 2 PoV shows ability to read process memory.
- A more difficult CB will act on the modified struct at some later point, after internal state has been advanced. Difficulty scales with the complexity of identifying implicitly-trusted protocol components across protocol state.

narf

CB Inspiration: Server-Side

Heartbleed logo

- CVE-2008-4250 (aka **MS08-067**): Conficker bug; buffer overflow in path canonicalization in Windows SMB -> **GREYMATTER / NRFIN_00011**
- 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 apriori 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

Notes from our proposal to DARPA:

Windows Server Service (SMB)

MS08-067

buffer overflow in NetAPI32.dll

Autonomous exploitation of this vulnerability is made difficult by the relatively complex manipulation of proximate state, the depth of the vulnerable code and the requirement to reason about data outside of
the current stack frame. Among other things, an autonomous system attempting to discover and

exploit this bug will need to (1) mock up a believably-accurate stack for this bug[13], and (2) provide some type inference for memory residing outside of the stack frame. Exploitation relies on the ability to decrement a pointer such that it points before its buffer (CWE-823).

OpenSSL

“Heartbleed”

discovery collision - found by Codenomicon via improvements to grammar-based fuzzer

Not found by AddressSanitizer due to custom heap, which would need modelling

---

CBs will present information leak vulnerabilities (CWE-200), resulting from such problems as uninitialized memory use (CWE-908), out-of-bounds reads (CWE-125), off-by-one errors (CWE-193), and improper null termination (CWE-170) in binary protocol servers. **These vulnerabilities have become increasingly valuable with the adoption of ASLR, but remain difficult to detect autonomously.**

We expect successful competitors to demonstrate

PoV via Method 2, by using the vulnerabilities to read the PoV address. To do so, **CRSs will need to be able to detect information leaks ranging from the obvious (e.g. out of bounds read causes the binary to crash) to the subtle (custom heap implementation, no clear delineation between elements, a la Heartbleed).** Existing state-of-the-art techniques like those used by AddressSanitizer will prove successful for some vulnerabilities, but more complex CBs will require pushing these strategies forward. CRSs capable of improving in this space will have a huge real-world impact.

narf

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 0xFFFFFFFC; });
var aaa = new Uint32Array(ab);
// aaa[0x1234567] = 1;
```

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

- There's (un)fortunately more than enough inspiration on this front.
- **V8 intrinsics bug: R/W to entire process space**
- **JailbreakMe: FreeType Type 1 "weird machine"**
- The difficulty posed by exploitation is an understanding that objects are nothing more or less than the memory backing them. **Many automatic tools make convenient but unrealistic assumptions about address space (e.g. the space is infinite or that a write to one object does not affect another). SSA form is a logical conclusion to these**

- **assumptions. Use-after-free vulnerabilities violate these assumptions.**
- **re concolic handoffs: Trail of Bits' biggest ask of program analysis developers: make your tools' internal state accessible and canonical to allow for hand-offs between tools. (c.f. https://vimeo.com/180882544)**

Notes from our proposal to DARPA:

V8 Intrinsics

CVE-2014-1705 is an out-of-bounds array access bug (CWE-129) in the V8 Javascript engine of Chrome.

This bug (and others like it) provide read / write access to the entirety of the Chrome's renderer memory.

The root cause of the vulnerability is misplaced trust in maliciously modified accessor functions on convenient objects (array buffers).

Locating this bug would likely require autonomous, semantic understanding of the trusted Javascript code, as well as an understanding of how to override a getter function to exercise the

vulnerability.
We do not expect competitors to be capable of reasoning at this level.
Our more challenging CBs in this model will present a simpler versions of this bug.
A CRS need not have a semantic understanding of the code, but simply be able to recognize an out-of-bounds access in native code and be able to construct malicious code that is interpreted for a Method 2 PoV.

Client-side vulnerabilities are an oft-used initial access vector for those attacking hardened networks.
The main factor in this trend is the **relative complexity of client software** compared against network-exposed services.
Complexity here comes from several sources which we emulate in our CBs, and makes automated bug identification difficult.
This trend makes addressing client-side vulnerabilities an important goal for current research.
For this reason, we model common bugs found in

client-side software from classic bug classes such as use-after-free (CWE-416), type confusion (CWE-843), out-of-bounds accesses (CWE-129), all the way to
dangling pointers (CWE-401).
These are all hard problems for CRSs, and **we are targeting our CBs to test (CRS) abilities in dynamic taint analysis[17] and memory usage checking, e.g. AddressSanitizer[18] or Valgrind's Memcheck[19]-like instrumentation.**
Our goal … is to probe likely weaknesses in CRS's static / dynamic handoff implementations.

Language parsers containing a double free / use-after-free (CWE-415, CWE-416), type-confusion (CWE-704, CWE-843, CWE-588, CWE-681), or out-of-bounds access bugs (CWE-125, CWE-787).
CBs emulate client-side applications by determining object sizes, types and usages at runtime.
Discovery of these bugs should be possible with state-of-the-art[12], but autonomous exploitation has not been publicly described.
The difficulty posed by exploitation is an

understanding that objects are nothing more or less than the memory backing them.
Many automatic tools make convenient but unrealistic assumptions about address space (e.g. the space is infinite or that a write to one object does not affect another).
SSA[1] form is a logical conclusion to these assumptions.
Use-after-free vulnerabilities violate these assumptions.
CRSs capable of reasoning about the practical implications of memory usage will excel here.

narf

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

Automatic difficulty scaling was accomplished via **obfuscator-llvm** + custom tooling.

symex = Symbolic Execution

narf

TAINTEDLOVE NRFIN_00019

@jannerboy62 on Unsplash

- 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

* Based in part on All You Ever Wanted to Know About Dynamic Taint Analysis and Forward Symbolic Execution (but might have been afraid to ask)

https://archive.darpa.mil/CyberGrandChallenge_Corpus/challenges/NRFIN_00019/

A CRS that overtaints is not sound.

Such CRSs run the risk of reporting false positives, tracking taint down errant and / or irrelevant paths.

This can present Denial of Service situations that prevent further vulnerability discovery.

Conversely, **a CRS that undertaints is not complete.**

Such CRSs run the risk of reporting false negatives, failing to report on a legitimate vulnerabilities, directly impacting competitor score.

This CB presented 4 levels of difficulty for the state of the art; we'll cover 2 of them here.

Successful competitors will blend static and dynamic approaches to drive execution to vulnerable flows and will precisely track taint in order to understand how to trigger the vulnerable conditions.

narf

TAINTEDLOVE NRFIN_00019

#1: entanglement_razzmatazz()

```
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 ...

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.

PA = Program Analysis

narf

TAINTEDLOVE NRFIN_00019

#2: causality_poppycock()

```
#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;
}
```

(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

narf

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

I'm not going to explicitly cover these two; please review the source.

Bonus slides for heisenberg at the end.

narf

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.

31

Definition: consensus eval is whether or not *any* competitor submitted a PoV that proved vulnerability in RB.

A yes means the competitor successfully defended against other competitors PoVs.

Takeaways

- **Offense - why try harder?** PoV once for each target and stop. It's a boolean: did you pop it or not?
  - **I wonder, could any of the CRSes solve > 1 challenge? CQE data insufficient to tell - or incentivize this behavior in CQE; this was incentivized in CFE.**
- **Defense - perhaps the 1/4 patched what they knew & CodeJitsu patched generically, thereby negatively affecting perf score?**
- **CodeJitsu and CSDS were the only two to pass**

- **consensus evaluation**

narf

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

@robaun on unsplash

narf

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**

@robaun on unsplash

narf

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**

@robaun on unsplash

narf

CableGri
NRFIN_0

- **procedurally generate**
  functions that called
- think **Wireshark** and i
  dissectors written in
- we fixed a bug late i
  development, forgot t
  DARPA
- this bug was in the p
  produced a vulnerabil
- **… 180 times** which means **Vs**

- 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

@robaun on unsplash

narf

CableGri
NRFIN_0

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

Branch: master ▾ cgc-challenge-corpus / NRFIN_00026 / src / cablegrindprotos.c Find file Copy path

demoray initial commit b65ebe8 on Apr 5, 2017

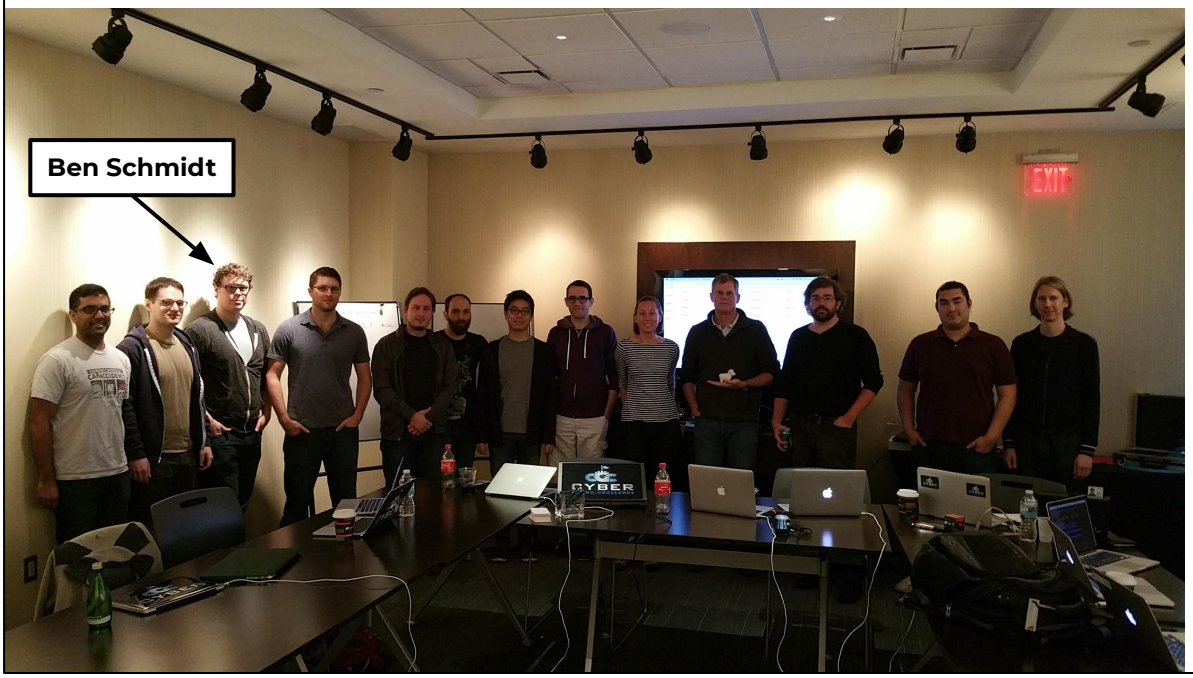
1 contributor

1.41 MB Download History

View raw

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

narf



Ben took a break from writing 180 PoVs to pose for this picture.

narf

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.

Takeaways

- **ToB PoV hit Shellphish but not CSDS.**
- **CSDS and ToB's patches were likely generic.**
- **Shellphish may have been generic, but was probably specific.**
- Challenges: indirection, many possible control flow paths
- **ToB's performance (memory, CPU usage) score likely dragged them down.**

narf

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

- **GREYMATTER / NRFIN_00011**
  - emulates **MS08-067**
  - PoV'ed by: ForAllSecure, DeepRed
  - PA challenge: strings are not NULL-terminated; state reasoning across calls to strncat()

narf

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

SOMECHECKS:

- all but the last vuln led to stack buffer overflow
- the CRS would also certainly demand a dictionary to solve level 5 & 6

INTERPRETTHIS:

- NaCl = Native Client, the same-process privilege separation system available in Chrome.

narf

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**

Many of our CBs did quite well at differentiating:

- 21 is the max for cqe_score
- 12 is the max for eval_score

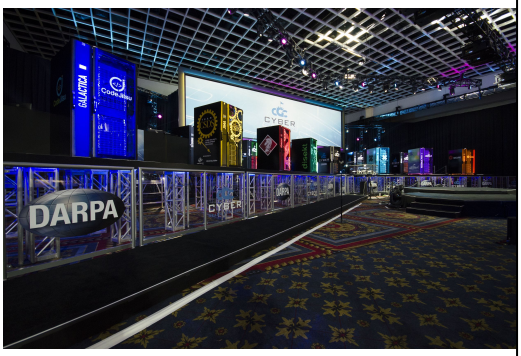
Notes:

- Only finalists were considered
- Evaluation score is differentiating competitors' ability to PoV.
- If you're looking for more stories in the CQE archive or interesting CBs to develop a CRS against, these are likely good candidates.

narf

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

Any vuln that permits PoV type 1 also permits PoV type 2.

narf

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	

- CFE: each CB fielded for at least 10 rounds
- all rounds compete autonomously
- once removed, CBs are not re-fielded

Narf's CFE CBs

of the **21 Narf CBs** fielded during CFE:

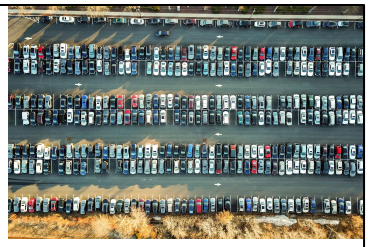
- **only 3 were PoV'ed**
  - we'll talk about these 3
  - we made the CBs too hard?
- **all competitors patched at least 1 CB**
- Shellphish patched all but 2 CBs
- CSDS only patched 2 CBs
- **ForAllSecure & CSDS had comparatively conservative patching strategies**

That one CB that everyone patched

@randyfath on Unsplash

narf

Overflow Parking NRFIN_00052



- 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
#ifdef PATCHED_1
if (total > INT_MAX) {
    return NULL;
}
#endif
```

46

https://archive.darpa.mil/CyberGrandChallenge_Corpus/challenges/NRFIN_00052/

Challenges

Firstly, this CB contains *many* integer overflows, the vast majority of which are unimportant, and even in some instances required behavior exercised by the pollers.

Systems attempting to explore or patch all of these overflows will likely encounter difficulty attempting to address them all, making techniques similar to -fsanitize=integer problematic.

Secondly, the vulnerability is contained within a standard library function rather than a non-standard function.

While this may make patching easier, as some teams may replace malloc altogether, it will make

root cause analysis more difficult.

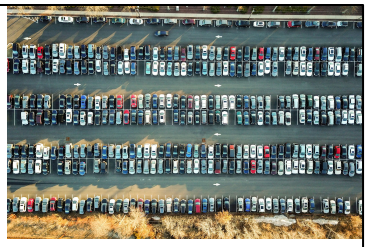
Thirdly, the protocol this CB uses is a mixture of line-delimited and length:data formats, which can confuse protocol reconstruction attempts.

Finally, proving vulnerability via heap corruption is a non-trivial task, even for humans.

Leveraging the somewhat narrow constraints available to achieve a write-what-where primitive, and using that to gain code execution, will likely prove quite challenging for an autonomous system.

narf

Overflow Parking NRFIN_00052



@omerrana on Unsplash

- **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**

The arrows denote a successful PoV: attacker -> defender.

narf

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**

https://archive.darpa.mil/CyberGrandChallenge_Corpus/challenges/NRFIN_00059/

In postage.c:get_new_stamp(), the serial number of each stamp is a series of 3 bytes from the flag page.

Each consecutive stamp gets the next 3 bytes from the flag page.

When it reaches the end of the flag page, it wraps back to the start..

If more than 1 stamp is purchased, the serial numbers can be concatenated to create longer sequences of bytes from the flag page.

Challenges

- The information leakage only leaks 3 bytes at a time. This means the CRS has to learn which

- the CRS has to learn that the bytes being leaked in each subsequent stamp are consecutive. Then the CRS can concatenate the bytes from 2 consecutive stamps together to create a single POV TYPE2_VALUE.
- All of the commands return 2, 3, or 8 byte buffers. So, the CRS will have to use taint tracking to determine which returned buffer contains leaked data and associate that returned buffer with a specific input command. Directly replaying the bytes returned from running a command in dumb fuzzing will not work.
- There are only 11 valid 4 byte command buffers. So, blind fuzzing will be of little use. Smart fuzzing that takes advantage of known command buffers can likely get some interaction with the CB. However, the result of each command depends on the internal state of the mailboxes. So, the CRS will have to model that internal state as well.

Difficulty

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

Snail Mail NRFIN_00059

@alex19perz on Unsplash

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**

narf

CFE: On Sale NRFIN_00063

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

@iiying on Unsplash

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

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

estimated difficulty:

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

52

https://archive.darpa.mil/CyberGrandChallenge_Corpus/challenges/NRFIN_00063/

Challenges

- There are a large number of products preloaded into the inventory and there are 100 predefined sale functions. This was an effort to make the binary larger in case that causes problems for any analysis tools.
- The do_update() function can be accessed by a fuzzer once the fuzzer learns the command to reach that function. This should be relatively easy.
- It should be relatively easy to figure out where the buffer overflow is happening on the receive call in do_update().
- What is more challenging is that the analysis tools have to know to call the do_list()

- function to get a list of valid barcodes that can then be used in the do_update() function.
- Additionally, the do_list() function takes a set of options. Those options determine how much data is returned for each product. When testing those options the code only checks to determine if they are odd or even. This will make a large number of valid inputs for these values that will give widely differing outputs. This should provide some distraction for analysis tools.
- Fixing the overflow should be pretty easy.

Difficulty

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

narf

CFE: On Sale NRFIN_00063

@iiying on Unsplash

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**

narf

CFE: On Sale NRFIN_00063

@iiying on Unsplash

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

narf

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

narf

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: **Hack time**
- ❖ **Franklin's BBQ Lunch**

Hack

Wednesday 10/8/2014: **Hack time**
- ❖ **There will be a team outing at a time and place TBD**

Hack

Thursday 10/9/2014: **Hack time**

Hack

Friday 10/10/2014: **Hack time**

Hack

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

narf

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

We did not observe libcgcdwarf used in game; recommended removal from game infrastructure

narf

Hackathon Evaluation Recommendations & Results 3

- **hardening recommendations** (all fixed):
  - many critical-path ELFs lacked PIE -> not full ASLR protection
  - many critical-path ELFs lacked stack protection
  - recommended Clang upgrade for --fstack-protector-strong
  - many critical-path ELFs 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)

From our report to DARPA:

1) **Problem**: ASLR isn't fully enabled, greatly reducing this mitigation's effectiveness. Specifically, critical-path ELFs are not produced as PIEs, and the kernel is not configured to fully randomize process address space.

Solution: Compile ELFs as fully-ASLR compatible. To do this, add "-fPIE -pie" to CFLAGS/LDFLAGS (as appropriate) for each project. Without this flag, the .text segment of the ELF is never randomized, irrespective of kernel configuration (modulo grsecurity/PaX's RANDEXEC).

Turn up the ASLR in the Linux kernel. Right now it's set to 1, meaning brk() managed memory is never randomized in process address space. For more:

http://securityetalii.es/2013/02/03/how-effective-is-aslr-on-linux-systems/ To fix this temporarily, write "2" to /proc/sys/kernel/randomize_va_space as root. To fix permanently (through reboots), add "kernel.randomize_va_space = 2" to /etc/sysctl.conf.

2) **Problem**: Stack protection is absent from many (all?) of the critical-path ELFs.

Solution: Enable stack protection by adding "-fstack-protector-all" to CFLAGS in all critical projects. I would recommend "-fstack-protector-strong", but the gcc included in DECREE is too old (requires gcc 4.9; DECREE has 4.7.2).

3) **Problem**: RELRO isn't bound, which means the GOT is writable at runtime. This is great for control of EIP). See more: http://tk-blog.blogspot.com/2009/02/relro-not-so-well-known-memory.html.

Solution: Add the following flags to CFLAGS/LDFLAGS (as appropriate): "-Wl,-z,relro,-z,now" in each project.

4) **Concern**: The Linux kernel presents a huge attack surface and there's no need for critical-path ELFs to interact with most of it.

Solution: Prior to processing any user-supplied input, issue a call to seccomp() or seccomp_bpf() to reduce kernel attack surface to only what is necessary to accomplish the task at hand. For example, AFAIK, there's no reason for

cgcef_verify to bind sockets, so why allow it to do so?

Some links:
http://blog.chromium.org/2012/11/a-safer-playground-for-your-linux-and.html
http://lwn.net/Articles/475043/
https://lwn.net/Articles/494252/
http://sourceforge.net/projects/libseccomp/
(Or just look at Chromium source).

5) **Concern**: The Linux kernel fails pretty hard at even basic exploit mitigations. For example, the DECREE kernel doesn't have stack cookies enabled. This isn't entirely surprising because not enabling them is the default setting.

Solution: Consider the grsecurity/PaX patches.

6) **Concern**: The ability to execute ELF syscalls equates to kernel compromise in the DECREE VM due to passwordless sudo and a lack of security boundary between root and kernelspace. Kernelspace execution greatly expands the hypervisor attack surface.

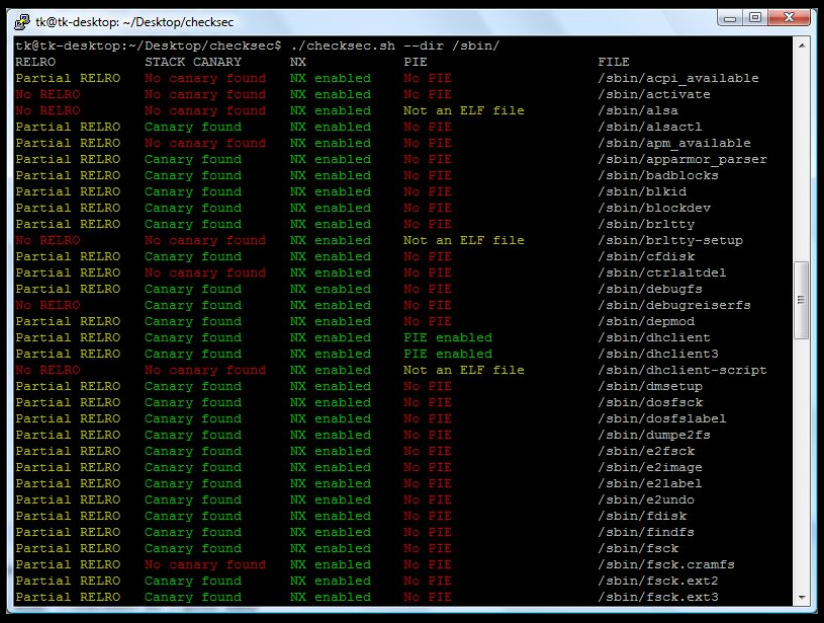
Solution: Change to nobody (or some unprivileged user) everywhere in the critical path that handles user input - prior to handling this input. We haven't audited to what (if any) extent this is currently happening.

7) **Problem**:
Multiple security issues in libcgcef/libcgcdwarf, which are based on outdated versions of libelf and libdwarf. There are multiple integer overflows in libcgcef that allow for heap

overflows, as well as at least two zero-sized mallocs that can lead to simple type-confusion vulnerabilities. Some of these vulnerabilities are reachable in cgcef_verify, and most are accessible in readcgcef. From our limited analysis, it appears likely that these would allow for execution of arbitrary code in the DECREE VM.

Rather than trying to discover and fix all the issues with these libraries, the best approach is likely to either a.) replace all necessary CGCEF parsers with Python-based parsers instead or b.) update to more recent forks of libelf/libdwarf and add a call to seccomp immediately after opening the file to be parsed, allowing only further reads, memory allocations, and exit. libcgcdwarf might be able to be removed entirely, since it doesn't appear that anything is currently using it.

narf



checksec.sh screenshot (not taken in DECREE VM)

narf

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

RB = Replacement Binary (a patched binary)

narf

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

We would have liked to have built a symbolic component to our “mock CRS”, but time would not allow for it.

narf

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

narf

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

narf

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

narf

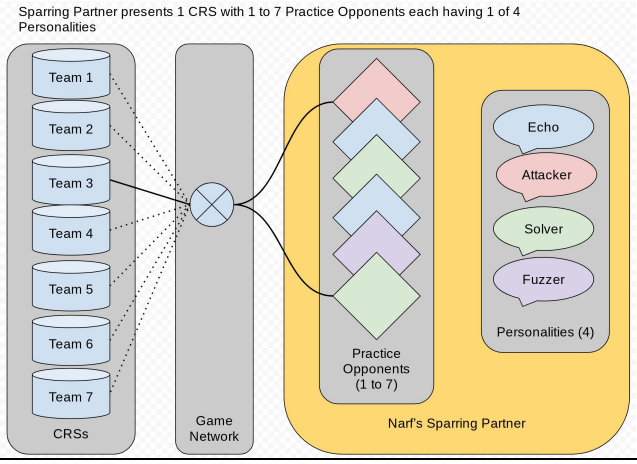
Competition Integrity

Lady Justice

- “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**

narf

Sparring Partner “Hadouken”



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

narf

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

Of these “cheeseball” defenses, **triple allocate was most effective** in deterring competitor PoVs.

Cheeseball defenses we never got to:

- Coin Flip: randomly drop connections
  - would have adverse effect on availability
- **Memory Games: mapping all memory to avoid faults**
  - this should actually work - it wouldn’t affect perf to simply allocate, only to page in memory
- PoV Denial of Service: put sleep()s along paths to known vulns

- we didn’t want additional reliance on wall time

narf

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
```


-O3 had highest yield in terms of PoV breakage during SE-II

Triple allocate had second highest yield.

Takeaway: just compile your software with odd flags, catch 0day in the wild.

narf

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)

* All credit for this analysis belongs to Mike Thompson

- Re “stealth” PoVs’ ineffectiveness:
  - perhaps competitors were not equipped well enough to repurpose thrown PoVs
  - some threw stealth in a nonsensical manner (after throwing non-stealth for the same vuln)
    - stealth would have no benefit in such situations

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

* All credit for this analysis belongs to Mike Thompson

narf

CFE Was Really Interesting*

NRFIN_00063 differentiated CRS' generic patching methodology:

1. CodeJitsu got EIP; used (obfuscated) payload to PoV Type 2 via transmit()
   - 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
   - ROP payload doesn't crash the service, but CRSs have already seen it crash

* All credit for this analysis belongs to Mike Thompson

Re generic defenses:

- return pointer encryption
- ASLR
- extended mallocs (triple alloc)
- protect indirect calls / jmps
- zero uninitialized stack
- W^X not possible with DECREE syscalls

CGC Impact

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

narf

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)
- 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)**

This is not a complete list!

narf

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

Will there be a CGC 2.0? Mike Walker encouraged others to take up the mantle.

narf

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

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

@neonbrand on Unsplash

Will there be a CGC 2.0? Mike Walker encouraged others to take up the mantle.

narf

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

* Many authors; enumerated at the link

This is by no means an exhaustive list. Links provided here a jumping off point to dive deeper into each team's methodology.

Didn't find technical publications by:

1. **DeFENCE** (SRI)
2. "Rubeus" (**DeepRed** Team @ Raytheon)
3. "CRSPY" (**Disekt**)

Other publications:
https://github.com/CyberGrandChallenge/
https://github.com/trailofbits/cb-multios
https://archive.darpa.mil/CyberGrandChallenge/
https://arxiv.org/pdf/1702.06162.pdf
https://archive.darpa.mil/CyberGrandChallenge_CompetitorSite/
https://vimeo.com/180882544
https://cgcdist.s3.amazonaws.com/cfe/darpa-cgc-cfe-event_log.pdf

https://github.com/CyberGrandChallenge/cgc-humint

narf

Thank You!

Questions?

A DARPA Cyber Grand Challenge
White Team Retrospective

Presented by: Paul Makowski

narf

Bonus Slides

narf

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]]
```


1. Constify one index into the recv buffer by XORing with itself
2. Use a 2nd index in recv buffer to access a 3rd index, XOR the 3rd index with itself.
3. Use a 4th index in recv buffer to access a 5th and (confusingly) XOR the 5th with itself.
4. Now indices XOR_CONST_OFF, 1+rx_buf[XOR_CONST_OFF_PTR] and 1+rx_buf[XOR_CONST_OFF_PTR*2] should be constified

narf

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);
```


1. Use quirks of floats to constify at index FLOAT_CONST_OFF
2. Modify the value in index FLOAT_PROP_OFF by way of jumping through float hoops

narf

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);
```

I super over-engineered this …

1. Use quirks of floats to constify at index FLOAT_CONST_OFF
2. Modify the value in index FLOAT_PROP_OFF by way of jumping through float hoops

narf

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);
```

70

1. Call deallocate on an invalid value; error return code will be static.
2. Reflect a value via the random() syscall, the number of random bytes will be equal to the number requests. Ignore the random bytes.
3. (Not displayed here), if any of the constified values differs from what it should be (this cannot happen), then recurse. This is intended to throw CRS' CFG reconstruction off.
4. Anyway, after all of this, 4 bytes are taken from input, including those propa

narf

CQE: TAINTEDLOVE NRFIN_00019

#3: heisenberg_hooey() the vuln

```
// 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
    }
```


Anyway, after all of this, 2 bytes from input are taken (those propagated via float and syscall operations and compared to the gate.

If the the gate is passed, attempt to write to the NULL page, causing a NULL ptr deref.