DARPA’s Cyber Grand Challenge: Creating a League of Extra-Ordinary Machines*

Ben Price and Michael Zhivich

ACSAC
December 10, 2015

* This work was sponsored by the Defense Advanced Research Projects Agency (DARPA) under US Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the author and are not necessarily endorsed by the United States Government.
Could a Purpose-Built Supercomputer Play DEF CON Capture-the-Flag (CTF)?
What is Capture-the-Flag?

Source: DARPA
What is Capture-the-Flag?

Source: DARPA
What is Capture-the-Flag?

Source: DARPA
What is Capture-the-Flag?
What is Capture-the-Flag?

Source: DARPA
What is Capture-the-Flag?
Cyber Grand Challenge: Create Cyber Reasoning Systems (CRS)

Source: DARPA
A League of Extra-Ordinary Machines

Chess Grandmasters

World Class Computer Science

Dedicated Systems

Chess
A League of Their Own

1977: NWU-Chess – Grandmaster Michael Stean defeated by a computer

1970: First all-computer tournament

1970 to 1977: An innovation explosion through measurable dominance:
- Chess hash tables
- Iterative deepening
- Bit boards
- Opening books
- Endgame databases

Key
- Software, General Purpose Hardware
- Single Purpose Hardware

Creating a League of Extra-Ordinary Machines - 11
BP, MZ 10/30/15
A League of Their Own

“In the past Grandmasters came to our computer tournaments to laugh. Today they come to watch. Soon they will come to learn.”

Monroe Newborn,
President, International Computer Chess Association, 1977
June 3, 2015: In the Beginning...

"We held the world's biggest [#capturetheflag] and all the contestants were robots." #cybersecurity #DARPACGC
Cyber Grand Challenge Timeline

- **Qualification Phase**: Jun 3, 2014 to Jun 3, 2015
- **Finalist Site Visits**: Jun 10, 2015 to Jul 17, 2015
- **Finals Phase**: Jul 18, 2015 to Aug 4, 2016
- **Trials**: Mar 14, 2016 to Apr 3, 2016
- **Cyber Grand Challenge Qualification Event**: Jun 3, 2015
- **Technical Paper**: Aug 8, 2016

The qualification phase is designed to identify the most effective automated systems that locate and mitigate security flaws. Participation in these Scored Events is optional and success in these events will not be evaluated as part of CGC scoring. Each Scored Event is an opportunity for competitors to gain an understanding of the format, procedure, and scoring mechanism to be used during the CQE.

Scored Events:
- Dec 2, 2014
- Apr 16, 2015
- Jul 18, 2015

Technical Paper:
- Mar 5, 2015
- Mar 5, 2015

Cyber Grand Challenge Final Event:
- Aug 4, 2016

Source: DARPA
Challenge for Infrastructure Team

Goal: Build a game that incentivizes improvements in automated program analysis and cyber reasoning

CRS: Cyber Reasoning System
CB: Challenge Binary
pcap: Packet capture
RB: Replacement Binary
PoV: Proof of Vulnerability
Challenge for Infrastructure Team

Goal: Build a game that incentivizes improvements in automated program analysis and cyber reasoning

Alternative Ecosystem
How do we make the test corpus representative of real-world challenges but not tainted by prior knowledge?
Challenge for Infrastructure Team

Goal: Build a game that incentivizes improvements in automated program analysis and cyber reasoning

Real-World Incentives
How do we evaluate “patched” replacement binaries to encourage solutions that will stand up to real-world pressures?
Challenge for Infrastructure Team

Goal: Build a game that incentivizes improvements in automated program analysis and cyber reasoning

Repeatable, Scalable Experiments

How do we ensure measurement system is scalable, consistent, and robust?
Challenge for Infrastructure Team

Alternative Ecosystem
How do we make the test corpus representative of real-world challenges but not tainted by prior knowledge?

No known protocols
No code reuse
### DARPA Experimental Cyber Research Evaluation Environment

<table>
<thead>
<tr>
<th>Syscall Name</th>
<th>Syscall #</th>
</tr>
</thead>
<tbody>
<tr>
<td>allocate</td>
<td>5</td>
</tr>
<tr>
<td>deallocallocate</td>
<td>6</td>
</tr>
<tr>
<td>transmit</td>
<td>2</td>
</tr>
<tr>
<td>receive</td>
<td>3</td>
</tr>
<tr>
<td>fdwait</td>
<td>4</td>
</tr>
<tr>
<td>random</td>
<td>7</td>
</tr>
<tr>
<td>_terminate</td>
<td>1</td>
</tr>
</tbody>
</table>
Scoping the Problem: DECREE

<table>
<thead>
<tr>
<th>OS</th>
<th># of System Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux 3.19</td>
<td>322</td>
</tr>
<tr>
<td>FreeBSD 11</td>
<td>549</td>
</tr>
<tr>
<td>Mac OS X</td>
<td>504</td>
</tr>
<tr>
<td>Win 8 SP 1</td>
<td>1469</td>
</tr>
<tr>
<td>DECREE</td>
<td>~7 (lower than 1)</td>
</tr>
</tbody>
</table>

The chart compares the number of system calls across different operating systems, showing DECREE has the lowest number of system calls.
**Scoping the Problem: DECREE**

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Process Creation</th>
<th>File System</th>
<th>Env Variables</th>
<th>Shared Libraries</th>
<th>Shared memory</th>
<th>Network sockets</th>
<th>IPC messaging</th>
<th>Sources of non-determinism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Mac OS X</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Win 8</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td><strong>DECREE</strong></td>
<td>❖</td>
<td>✗</td>
<td>✗</td>
<td>❖</td>
<td>❖</td>
<td>❖</td>
<td>✔</td>
<td>✗</td>
</tr>
</tbody>
</table>

- Processes and network sockets are managed by CGC process launcher
- Only source of non-determinism is a pseudo-random number generator
DECREE:
Limited Scope, Limitless Possibilities

- Basic messaging application
- Simple particle physics simulator
- RAM-based filesystem
- Radio receiver after demodulation
- Spreadsheet program
- Basic virtual machine
- Industrial control system
- A new way to pay for everything
- …and 123 more…
Challenge for Infrastructure Team

Real-World Incentives
How do we evaluate “patched” replacement binaries to encourage solutions that will stand up to real-world pressures?
Real-World Constraints

- Security solutions should not break functionality
  - Antivirus delivers a false-positive detection…in SVCHOST.exe
    (https://support.microsoft.com/en-us/kb/2025695)
Real-World Constraints

- Security solutions should not break functionality

- Significant performance degradation will not be tolerated
  - “CPU and memory cost below 5%” (Microsoft BlueHat Contest, “Practical and Functional” criterion)
Real-World Constraints

- Security solutions should not break functionality
- Significant performance degradation will not be tolerated
- Security solutions must mitigate attacks
  - “Relying solely on perimeter defenses is now passé – and naively dangerous” (Kelly Jackson Higgins, “Damage Mitigation as the New Defense”)
Real-World Constraints

- Security solutions should not break functionality

- Significant performance degradation will not be tolerated

- Security solutions must mitigate attacks

- Rewarding Proof of Vulnerability (PoV) discovery enables fixing bugs sooner
  - “SAGE found ~ 1/3 of all bugs found by file fuzzing in Windows 7”
Scoring Algorithm

\[
\text{SubScore}(RB, PoV) = \text{Availability}(RB) \times \text{Security}(RB, PoV) \times \text{Evaluation}(PoV)
\]

- Security solutions should not break functionality
- Significant performance degradation will not be tolerated
- Security solutions must mitigate attacks
- Rewarding PoV discovery enables fixing bugs sooner
Scoring Algorithm

\[ \text{SubScore}(RB, \text{PoV}) = \text{Availability}(RB) \times \text{Security}(RB, \text{PoV}) \times \text{Evaluation}(\text{PoV}) \]

- ✓ Security solutions should not break functionality
- ✓ Significant performance degradation will not be tolerated
- ❑ Security solutions must mitigate attacks
- ❑ Rewarding PoV discovery enables fixing bugs sooner
Scoring Algorithm

\[ \text{SubScore}(RB, PoV) = \text{Availability}(RB) \times \text{Security}(RB, PoV) \times \text{Evaluation}(PoV) \]

- 0 if no reference PoVs mitigated
- \(1 + \frac{1}{2} \times (\text{Reference} + \text{Consensus})\) otherwise

- ✓ Security solutions should not break functionality
- ✓ Significant performance degradation will not be tolerated
- ❌ Security solutions must mitigate attacks
- ❌ Rewarding PoV discovery enables fixing bugs sooner
Scoring Algorithm

\[
\text{SubScore}(RB, \text{PoV}) = \text{Availability}(RB) \times \text{Security}(RB, \text{PoV}) \times \text{Evaluation}(\text{PoV})
\]

0 if no reference PoVs mitigated

\[
1 + \frac{1}{2} \times (\text{Reference} + \text{Consensus})
\]

otherwise

\[
\frac{\# \text{ reference PoVs mitigated}}{\# \text{ reference PoVs}}
\]

0 if any competitor PoV succeeded

1 otherwise

✓ Security solutions should not break functionality
✓ Significant performance degradation will not be tolerated
✓ Security solutions must mitigate attacks
☐ Rewarding PoV discovery enables fixing bugs sooner
Scoring Algorithm

\[
\text{SubScore}(RB, \text{PoV}) = \text{Availability}(RB) \times \text{Security}(RB, \text{PoV}) \times \text{Evaluation}(\text{PoV})
\]

1. if PoV was unsuccessful
2. if PoV was successful

- Security solutions should not break functionality
- Significant performance degradation will not be tolerated
- Security solutions must mitigate attacks
- Rewarding PoV discovery enables fixing bugs sooner
SubScore(RB, PoV) = Availability(RB) × Security(RB, PoV) × Evaluation(PoV)

- Availability (FuncFactor): How many tests does the service pass?
- Availability (PerfFactor): How much CPU and memory does the service use?
- Security (Reference): Do reference PoVs crash the replacement service?
- Security (Consensus): Do submitted PoVs crash the replacement service?
- Evaluation: Does submitted PoV crash the vulnerable service?
Measurement Framework

Beware the cracks in the abstraction layer

- **Availability** *(FuncFactor)*: How many tests does the service pass?
- **Availability** *(PerfFactor)*: How much CPU and memory does the service use?
- **Security** *(Reference)*: Do reference PoVs crash the replacement service?
- **Security** *(Consensus)*: Do submitted PoVs crash the replacement service?
- **Evaluation**: Does submitted PoV crash the vulnerable service?
void echo() {
    char buf[64];
    while (receive(STDIN, &buf, 128, NULL) == 0) {
        transmit(STDOUT, &buf, 64, NULL);
    }
}
Evaluating PoVs: Proof of Vulnerability

- Accessing unmapped memory (SIGSEGV)
- Executing illegal instruction (SIGILL)

```c
void echo() {
    char buf[64];
    while (receive(STDIN, &buf, 128, NULL) == 0) {
        transmit(STDOUT, &buf, 64, NULL);
    }
}
```
Evaluating PoVs: Audience Poll

```c
void echo() {
    char buf[64];
    while (receive(STDIN, &buf, 128, NULL) == 0) {
        transmit(STDOUT, &buf, 64, NULL);
    }
}
```

Send 128 bytes to this service – does it crash?
Evaluating PoVs: Audience Poll

```c
void echo() {
    char buf[64];
    while (receive(STDIN, &buf, 128, NULL) == 0) {
        transmit(STDOUT, &buf, 64, NULL);
    }
}
```

Send 128 bytes to this service – does it crash?

...That depends on **packet size** and **timing**!
void echo() {
    char buf[64];
    while (receive(STDIN, &buf, 128, NULL) == 0) {
        transmit(STDOUT, &buf, 64, NULL);
    }
}

**Single write, single read: Service will crash**
void echo() {
    char buf[64];
    while (receive(STDIN, &buf, 128, NULL) == 0) {
        transmit(STDOUT, &buf, 64, NULL);
    }
}

transmit(64 bytes)

\[
\text{tx\_bytes} = 64
\]

transmit(64 bytes)

\[
\text{tx\_bytes} = 64
\]

receive(up to 128 bytes)

✔ Multiple writes, single read: Service will crash
void echo() {
    char buf[64];
    while (receive(STDIN, &buf, 128, NULL) == 0) {
        transmit(STDOUT, &buf, 64, NULL);
    }
}

\[ \text{transmit(64 bytes)} \]
\[ \text{tx\_bytes} = 64 \]

\[ \text{receive(up to 128 bytes)} \]
\[ \text{rx\_bytes} = 64 \]

\[ \text{\textbf{✗ Multiple writes, multiple reads: Service won't crash!}} \]
void echo() {
    char buf[64];
    while (receive(STDIN, &buf, 128, NULL) == 0) {
        transmit(STDOUT, &buf, 64, NULL);
    }
}

Send 128 bytes to this service – does it crash?
...
That depends on packet size and timing!

This behavior is non-deterministic!
Evaluating PoVs: High(er) Reliability

- Change CB to avoid non-deterministic behavior
- Verify reference PoVs and polls work with different write chunk sizes and random seeds
- Re-run competitor PoVs several times with different random seeds; if it’s ever successful, count as successful evaluation

```c
void echo() {
    char buf[64];
    while (receive_until(STDIN, &buf, '\n') == 0) {
        transmit(STDOUT, &buf, 64, NULL);
    }
}
```
Challenge for Infrastructure Team

Repeatable, Scalable Experiments
How do we ensure the measurement system is scalable, consistent, and robust?
Challenge for Infrastructure Team

Repeatable, Scalable Experiments

How do we ensure the measurement system is scalable, consistent, and robust?
Challenge for Infrastructure Team

Repeatable, Scalable Experiments
How do we ensure the measurement system is scalable, consistent, and robust?
CQE Distribution System

Amazon S3
- Team 1 Bucket
- Team 2 Bucket
- Team n Bucket

Firewall

Limbo

Encrypted CQE Bundle

Database & NFS

Post

Submissions Queue

Availabilty Queue

Security Queue

Evaluation Queue

Scoring Taskmaster

DECREE VM
Each team has equal access to the challenge bundle.

Contents of the challenge bundle remain secret until CQE begins.
CQE Distribution System

- Each team has equal access to the challenge bundle
- Contents of the challenge bundle remain secret until CQE begins

Key distribution: Twitter, SMS, Email

PASS: Ultimately, what separates a winner from a loser at the grandmaster level is the willingness to do the unthinkable. 5844659ce9891a09
Creating a League of Extra-Ordinary Machines - 52

BP, MZ 10/30/15

CQE Submission System

CRS can only access its own bucket
CQE Submission System

- Submission system must be “high-availability”
- Each submission must be time-stamped
- Only scoring system can read submissions
- Submission uploads must be atomic operations
- Submission uploads must be allowed only during CQE
Submission system:
Amazon S3

Submission system must be “high-availability”
Submission system: Amazon S3

- Submission system must be “high-availability”
- Each submission must be time-stamped
CQE Submission System

- Submission system must be “high-availability”
- Each submission must be time-stamped
- Only scoring system can read submissions

Encrypt submissions using pre-shared, per-team keys
CQE Submission System

- Submission system must be “high-availability”
- Each submission must be time-stamped
- Only scoring system can read submissions
- Submission uploads must be allowed only during CQE
- Submission uploads must be allowed only during CQE
Creating a League of Extra-Ordinary Machines - 58
BP, MZ 10/30/15
# CQE Scoring System Statistics

<table>
<thead>
<tr>
<th>Type of Test</th>
<th>Number of Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality and Performance</td>
<td>16,167,316</td>
</tr>
<tr>
<td>Reference Security</td>
<td>186,720</td>
</tr>
<tr>
<td>Consensus Security</td>
<td>438,760</td>
</tr>
<tr>
<td>Proof of Vulnerability Evaluation</td>
<td>52,600</td>
</tr>
<tr>
<td><strong>Total Tests</strong></td>
<td><strong>16,845,396</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hosts</th>
<th>Total RAM (GB)</th>
<th>Total CPU Cores</th>
<th>DECREE VMs</th>
<th>Time to Provision</th>
<th>Time to Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>3,072</td>
<td>240</td>
<td>192</td>
<td>40 min</td>
<td>16 hours</td>
</tr>
</tbody>
</table>

Results confirmed by independent (HW and SW) scoring system
Hey @amazon, we're about to shut down our CRS. Let us know if you hear a pitch change in the datacenter. #DARPACGC
Qualifying Event

- 6,000+ Functions
- 190,000+ LOC .h
- 7,000+ LOC .cc
- 200,000+ LOC .c
- 131 Challenge Sets
- 5 IPC Challenge Sets
- 590 Embedded Vulnerabilities
- 50+ Distinct CWEs

CWE = Common Weakness Enumeration
IPC = Inter-Process Communication
LOC = Lines of Code
Common Weaknesses in CQE

- CWE-122: Heap-based Buffer Overflow
- CWE-121: Stack-based Buffer Overflow
- CWE-787: Out-of-bounds Write
- CWE-476: NULL Pointer Dereference
- CWE-120: Buffer Copy without Checking Size of Input
- CWE-190: Integer Overflow or Wraparound
- CWE-129: Improper Validation of Array Index
- CWE-125: Out-of-bounds Read
- CWE-193: Off-by-one Error
- CWE-131: Incorrect Calculation of Buffer Size
- CWE-20: Improper Input Validation
- CWE-416: Use After Free
- CWE-788: Access Memory Location After End of
- CWE-134: Uncontrolled Format String
- CWE-843: Access of Resource Using Incompatible
- CWE-119: Improper Restriction of Operations within
- CWE-824: Access of Uninitialized Pointer
- CWE-674: Uncontrolled Recursion
- CWE-457: Use of Uninitialized Variable
- CWE-195: Signed to Unsigned Conversion Error
- CWE-822: Untrusted Pointer Dereference
- CWE-908: Use of Uninitialized Resource
- CWE-839: Numeric Range Comparison Without
- CWE-704: Incorrect Type Conversion or Cast
- CWE-191: Integer Underflow (Wrap or Wraparound)
CQE from Perspective of CRS

[Diagram showing the flow of an encrypted CQE bundle through a CRS system connected to Amazon S3.]
Grand Challenge for CRS Creators

Program Analysis
- Challenge Binaries
- Triage
- Unpack
- Fuzzing
- Post-Mortem Analysis
- Static Analysis
- Symbolic Execution
- SMT/SAT
- Capture & Replay
- Trace, Monitor, Prioritize
- Program Path DB

Network Analysis
- Fingerprint Scanners
- Guards
- Signatures

Defense Generation

Source: DARPA
CRS Strategy 1: Fuzz and Fix

- Look at network traffic capture
- Fuzz to find a PoV
- Patch the observed crash
CRS Strategy 2: Generic Hardening

- Analyze CB for possible memory corruption
- Patch to validate pointers before memory access
CRS Strategy 3: Symbolic Execution

- Symbolically execute CB to collect path constraints
- Solve for possible memory corruption
- Verify via concrete execution
- Patch confirmed crash sites
Successful PoV & Patch
Successful PoV
Successful Patch
Unsuccessful Submission

Legend
- Approved for Public Release, Distribution Unlimited
Example Challenge: YAN01_00012

- A simple stack-based machine that uses 32-bit words
  - 3 bits for opcode
  - 29 bits for immediate values

- Vulnerability is a missing check on writes to a heap-allocated buffer

- This challenge was designed to test a CRS’ ability to
  - Monitor the heap (allocate and deallocate system calls)
  - Support bit-wise operations for dependency analysis
  - Identify VM instructions that can be used to cause a VM stack overflow and patch them
## YAN01_00012 Instruction Set

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>PUSH n</td>
<td>Push specified 32-bit number onto the stack</td>
</tr>
<tr>
<td>01</td>
<td>POP</td>
<td>Pop a 32-bit number from the stack</td>
</tr>
<tr>
<td>02</td>
<td>PUSHPC</td>
<td>Push program counter onto the stack</td>
</tr>
<tr>
<td>03</td>
<td>JMPZ</td>
<td>Pop two 32-bit values off the stack; if the first one is equal to 0, jump to the second value</td>
</tr>
<tr>
<td>04</td>
<td>SWAP n</td>
<td>Swap nth stack entry with the top one</td>
</tr>
<tr>
<td>05</td>
<td>DUP n</td>
<td>Duplicate nth stack entry and push it to the top of the stack</td>
</tr>
<tr>
<td>06</td>
<td>ADD</td>
<td>Pop top two numbers off of the stack, add them, and push the sum back onto the stack</td>
</tr>
<tr>
<td>07</td>
<td>SUB</td>
<td>Pop the top two numbers off of the stack, subtract them, and push the difference back onto the stack</td>
</tr>
<tr>
<td>FFFFFFFF</td>
<td>RET</td>
<td>End of instruction stream – the topmost value on stack is the return value</td>
</tr>
</tbody>
</table>
### YAN01_00012 Instruction Set

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>PUSH n</td>
<td>Push specified 32-bit number onto the stack</td>
</tr>
<tr>
<td>01</td>
<td>POP</td>
<td>Pop a 32-bit number from the stack</td>
</tr>
<tr>
<td>02</td>
<td>PUSHPC</td>
<td>Push program counter onto the stack</td>
</tr>
<tr>
<td>03</td>
<td>JMPZ</td>
<td>Pop two 32-bit values off the stack; if the first one is equal to 0, jump to the second value</td>
</tr>
<tr>
<td>04</td>
<td>SWAP n</td>
<td>Swap nth stack entry with the top one</td>
</tr>
<tr>
<td>05</td>
<td>DUP n</td>
<td>Duplicate nth stack entry and push it to the top of the stack</td>
</tr>
<tr>
<td>06</td>
<td>ADD</td>
<td>Pop top two numbers off of the stack, add them, and push the sum back onto the stack</td>
</tr>
<tr>
<td>07</td>
<td>SUB</td>
<td>Pop the top two numbers off of the stack, subtract them, and push the difference back onto the stack</td>
</tr>
<tr>
<td>FFFFFFFF</td>
<td>RET</td>
<td>End of instruction stream – the topmost value on stack is the return value</td>
</tr>
</tbody>
</table>
# YAN01_00012 Instruction Set

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>PUSH n</td>
<td>Push specified 32-bit number onto the stack</td>
</tr>
<tr>
<td>01</td>
<td>POP</td>
<td>Pop a 32-bit number from the stack</td>
</tr>
<tr>
<td>02</td>
<td>PUSHPC</td>
<td>Push program counter onto the stack</td>
</tr>
<tr>
<td>03</td>
<td>JMPZ</td>
<td>Pop two 32-bit values off the stack; if the first one is equal to 0, jump to the second value</td>
</tr>
<tr>
<td>04</td>
<td>SWAP n</td>
<td>Swap nth stack entry with the top one</td>
</tr>
<tr>
<td>05</td>
<td>DUP n</td>
<td>Duplicate nth stack entry and push it to the top of the stack</td>
</tr>
<tr>
<td>06</td>
<td>ADD</td>
<td>Pop top two numbers off of the stack, add them, and push the sum back onto the stack</td>
</tr>
<tr>
<td>07</td>
<td>SUB</td>
<td>Pop the top two numbers off of the stack, subtract them, and push the difference back onto the stack</td>
</tr>
<tr>
<td>FFFFFFFF</td>
<td>RET</td>
<td>End of instruction stream – the topmost value on stack is the return value</td>
</tr>
</tbody>
</table>
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE pov SYSTEM "/usr/share/cgc-docs/replay.dtd">
<pov>
  <cbid>YAN01_00012</cbid>
  <replay>
    <delay>500</delay>
    <write>
      <data>
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        ...
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
      </data>
    </write>
  </replay>
</pov>
Second Proof of Vulnerability Attempt (Successful)

<write>
  <data format="hex">
    000000000
    000000000
    000000000
    ffffffff7f
    000000000
    050000000
    000000000
    000000000
    030000000
    ...
    ffffffffff
    ...
  </data>
</write>

Resulting Program

0. PUSH 0
1. PUSH 0
2. PUSH 0
3. SUB
4. PUSH 0
5. DUP 0
6. PUSH 0
7. PUSH 0
8. JMPZ

GOTO Line 0

RET
Original YAN01_00012: PUSH Instruction
Defended YAN01_00012: PUSH Instruction (1/2)

```assembly
loc_0048401:         ; BEGIN PUSH
    mov    eax, [ebp+curTop]
    add    eax, 1
    mov    [ebp+curTop], eax
    mov    eax, [ebp+insn]
    shr    eax, 3
    mov    ecx, [ebp+curTop]
    jmp    check_ptr_stack

check_ptr_stack:
    mov    edx, [ebp+stack]
    movd   xmm0, eax
    lea    eax, [edx+ecx*4]
    sub    eax, ds:stack_top
    cmp    eax, 4092
    jbe    short do_push

    lea    eax, [edx+ecx*4]
    mov    ds:ret_addr, offset do_push
    jmp    check_ptr
```
Defended YAN01_00012: PUSH Instruction (2/2)

```
check_ptr:
  mov  ds:temp_stack, esp
  lea  esp, temp_stack
  push ecx
  push edx
  push 6
  push 4
  push eax
  call check_ptr_full
  jmp ret_to_app

; START OF FUNCTION CHUNK FOR main_loop
ret_to_app:
  pop   edx
  pop   ecx
  pop   esp
  jmp   ds:ret_addr
```
Time to First Defended Solution
Time to First Defended Binary

Fraction of Challenge Sets (cumulative)

0% 10% 20% 30% 40% 50% 60% 70% 80% 90% 100%

Time to First Successful PoV

Fraction of Challenge Sets (cumulative)

- 12:00
- 13:00
- 14:00
- 15:00
- 16:00
- 17:00
- 18:00
- 19:00
- 20:00
- 21:00
- 22:00
- 23:00
- 0:00
- 1:00
- 2:00
- 3:00
- 4:00
- 5:00
- 6:00
- 7:00
- 8:00
- 9:00
- 10:00
- 11:00
- 12:00
Time to First Successful PoV
Machines Think Differently

**Proofs of Vulnerability**

*Intended* (bugs inserted by design)  
59%

*Unintended* (bugs first found by machines)  
41%

For example – it turns out *not reading* from a socket can cause a buffer overflow if writer doesn’t check available buffer space.
June 3, 2015: In the Beginning...

CRS received perfect scores on 18% of challenges

"We held the world's biggest [#capturetheflag] and all the contestants were robots." #cybersecurity #DARPACGC
“We can only see a short distance ahead, but we can see plenty that needs to be done.”

~ Alan Turing
Envisioned Road Map

- **8 Months**: CGC Final Event (CFE)
- **1–3 Years**: Analysis Tools
- **≤ 10 Years**: Auto Custom Patching
- **≤ 20 Years**: Game Theory
Short Term: Assisting the Software Analyst

• Automated unpacking
• Vulnerability discovery
• Taint tracing
  – Functional equivalency of standard routines
• Anomaly Detection
  – Can compare with specification/expectations and look for divergence (e.g., old vs new variants of program)
• Currently available tools
  – Mcsema
  – angr (management)
  – BAP
  – BitBlaze
Mid-term: Custom Patching

• On-demand custom patching
  – Reduced time to patch
  – Not dependent on vendor
  – Tailored to specific workload/inputs
  – Update unsupported legacy software

• Use CRS to remove/modify functionality
  – Remove remote tracking
  – Don’t load images in email client
  – Don’t turn URLs into links

Side Effect: software diversity prevents widespread attacks
Automated 3rd-Party Repairs Are Close

- Fun With Shellshock: http://blog.regehr.org/archives/1187 (Oct. 11 2014)
  - “We simply inserted an exploit that attempted to cat a “passwd” file into a GET request”

```
GET /appstore/index.php HTTP/1.1
User-Agent: { }{ ;}; /bin/cat /home/mitll/passwd > /tmp/hello.txt
Host: 155.98.38.76:7701
Accept: */*
```

- A3 able to remove bash functionality and mitigate vulnerability
- “…A3’s mandatory mediation blocked the attack …”
- “A3 took ~2 minutes to find a repair …”
- “A3 took an additional ~1.5 minutes to find a source code repair …”
Long term:
CRS Interactive with Opponent

• Machine vs machine competition adds complexity and a ‘Game Theoretic’ aspect, where CRS may:
  – Make decisions on what type of patch to deploy
  – Learn what kind of analysis is being used
  – Intentionally misinform opponent
  – Set up weaker defenses to see how opponent reacts

Source: DARPA

• Could a CRS build an adversary profile based on visible artifacts?
• “Interactive honeypot” backed by a CRS?
Envisioned Road Map

- 8 Months: CGC Final Event (CFE)
- 1–3 Years: Analysis Tools
- ≤ 10 Years: Auto Custom Patching
- ≤ 20 Years: Game Theory
ENTER _start
    call main
    pushl %eax
    call _terminate
END _start
Lesson 1

Relevant solutions require real-world constraints

- Security solutions cannot break functionality
- Significant performance degradation will not be tolerated
- Security solutions must mitigate attacks
Lesson 2

There’s no substitute for the real event

• Integration and scale issues are hidden until you ‘go live’
• Practice like you play
Lesson 3

Beware the cracks in the abstraction layer

• Low-level artifacts can affect determinism of higher-level behavior

• Resources are finite
“Be conservative in what you do, be liberal in what you accept from others.”

[ RFC 793 ]
Lesson 4

Don’t trust; verify

“Be conservative in what you do, be liberal extremely conservative in what you accept from others.”

[CGC mantra]

• Be explicit in specification, validate ruthlessly
• Solve the halting problem (watchdog timer)

If you can’t repeat it, it didn’t happen
(if it’s not automated, you can’t repeat it)

- Automated unit tests for everything
- Building and testing challenge sets
- Scoring cluster provisioning and push-button scoring
Lesson 6

Give people a challenge, and they will surprise you

“Machines take me by surprise with great frequency.”
– Alan Turing
Source Code and Walkthroughs:
https://github.com/CyberGrandChallenge

Packages, VMs, and Scoring Data:
http://repo.cybergrandchallenge.com
Acknowledgements
Meet the Finalists

ForAllSecure
Pittsburgh, PA

Deep Red
Arlington, VA

TECHx
Charlottesville, VA

Shellphish
Santa Barbara, CA

disekt
Athens, GA

Codejitsu
Berkeley, CA

CSDS
Moscow, ID
Save The Date: CGC Final Event

August 4, 2016
DEF CON
Las Vegas, NV
Dashboard, complete with automatic haiku generator, ready #DARPACGC

TechXCRS says: early morning, in the pond, sudo cb-test

Leaders 192.168.8.3 192.168.8.4 192.168.8.6
VM Load 0 / 297 0 0
Errors 0
Bundles 0 Crashes 0 [a-g] 0 POVs 0 [a-g] 0
Protected Binaries 0 / 0 Submissions 0 / 0
Live IPs 300 / 300 PCAPs 0 / 0

6:41 PM - 2 Jun 2015
phenomenal nrfin_05, in the formless dusk, clever smell arises from the clearing

#DARPACGC

12:45 PM - 4 Jun 2015
do fuzz. or do not. there is no try.

#DARPACGC

12:45 AM - 4 Jun 2015
More input Stephanie!

11:59 AM - 3 Jun 2015
CGC quals, Shellphish-style: the guy in charge of our testing infrastructure just accidentally ran "rm -rf /cgc"...
go ahead POV, make my day #DARPACGC
Hey @amazon, we're about to shut down our CRS. Let us know if you hear a pitch change in the datacenter. #DARPACGC
#CRSFacts We had 28 nodes in GCE, 10 in AWS, and 2 local servers for a total of 804 cores. We crashed 81 CBs and patched 128 CBs. #DARPACGC
Save The Date: CGC Final Event

August 4, 2016
DEF CON
Las Vegas, NV
Meet the Finalists: CodeJitsu

CodeJitsu is based at the University of California Berkeley and led by Professor Dawn Song. The CodeJitsu cyber reasoning system is based on automated binary analysis and hardening.
The Center for Secure and Dependable Systems at the University of Idaho is proud to sponsor team **CSDS**. This self-funded team consists of Dr. Jia Song, a postdoc, and Dr. Jim Alves-Foss, director of CSDS. Although a small team, they are building from scratch a **new and innovative custom tool suite** to participate in CGC.
Deep Red is composed of a small team of specialized engineers from Raytheon Corporation. The Deep Red team is inventing new ways to analyze software that builds on the team’s uniquely rich heritage in computer security.
Meet the Finalists: disekt

disekt is a computer security team that participates in various Capture the Flag security competitions hosted by other teams, universities and organizations from around the world.
Meet the Finalists: ForAllSecure

ForAllSecure’s technology is the result of more than a decade of program analysis research at Carnegie Mellon University by Professor David Brumley, Thanasssis Avgerinos, and Alex Rebert.
Shellphish started at the University of California Santa Barbara as the SecLab hacking team. As members graduated and moved, the team expanded to include other locations such as France, United Kingdom, and other exotic locations. Shellphish has participated in more DEF CON CTF events than any other team.
The **TECHx** team consists of leading software analysis experts from GrammaTech, Inc. and the University of Virginia. The team is led by Dr. David Melski, Professor Jack Davidson, and Professor John Knight. GrammaTech and UVA are co-developers of an automatic software-hardening technology called **PEASOUP**.