Agenda

- Shellphish
- The DARPA Cyber Grand Challenge
- Shellphish’s Cyber Reasoning System
- Automatic Vulnerability Discovery
  - Angr → Live demonstration!
- Towards the Cyber Grand Challenge Finals
Agenda

● Shellphish

● The DARPA Cyber Grand Challenge

● Shellphish’s Cyber Reasoning System

● Automatic Vulnerability Discovery
  ○ Angr → Live demonstration!

● Towards the Cyber Grand Challenge Finals
Shellphish

- Who are we?
  - a team of security enthusiasts
    - do research in system security
    - play Capture the Flag competitions
Shellphish

- Started (in 2004) at:
  - SecLab: University of California, Santa Barbara
Shellphish

- expanded to:
  - Northeastern University: Boston
  - Eurecom: France
  - ...

A Dozen Years of Shellphish – from DEFCON to the Cyber Grand Challenge
CTF competitions

- Security competitions

- Different challenges
  - exploit a vulnerable service
  - exploit a vulnerable website
  - reversing a binary
  - ...

- Different formats
  - Jeopardy – Attack-Defense
  - Online – Live
We do not only play CTFs

We also organize them!

- UCSB iCTF
  - Attack-Defense format
  - every year, since 2002!

References:
- [ictf.cs.ucsb.edu](http://ictf.cs.ucsb.edu)
- [github.com/ucsb-seclab/ictf-framework](https://github.com/ucsb-seclab/ictf-framework)
○ If you want to know more about Shellphish:
  ■ Attend the talk of my “colleague”:
    Yan Shoshitaishvili

  ■ Saturday, August 29th (14:20 – 15:10)
    HITCON Community
Agenda

● Shellphish

● The DARPA Cyber Grand Challenge
  ● Shellphish’s Cyber Reasoning System
  ● Automatic Vulnerability Discovery
    ○ Angr → Live-demonstration!

● Towards the Cyber Grand Challenge Finals
Cyber Grand Challenge (CGC)

- **2014**: DARPA Cyber Grand Challenge
  - Autonomous hacking!
Cyber Grand Challenge (CGC)

- Started in 2014

- Qualification event: June 3rd, 2015, online
  - ~70 teams → 7 qualified teams

- Final event: August 4th, 2016 @ DEFCON (Las Vegas)
CGC – Rules

- Attack-Defense CTF
- No human intervention
- Develops a system that automatically
  - Exploit vulnerabilities in binaries
  - Patch binaries, removing the vulnerabilities
CGC Qualification Event – Rules

• Every team has to:
  ○ Generate exploits
    ■ an input to a binary
    ● the binary crashes (invalid memory access)
    ● encoded as a list of recv/send/… operations
  ○ Patch binaries
    ■ fix the vulnerabilities
    ■ preserve the original binary’s functionality
    ■ performance impact is evaluated
      ● CPU time, memory consumption, disk space
CGC Qualification Event – Rules

- Architecture: Intel x86, 32bit

- Operating System: DECREE
  - Linux-like
  - only 7 syscalls
    - terminate (exit)
    - transmit (write)
    - receive (read)
    - fdwait (select)
    - allocate (mmap)
    - deallocate (munmap)
    - random
  - no signal handling, no not-executable stack, no ASLR, ...

- DECREE VM
  - standard Linux ELF binaries
  - CGC binaries
Agenda

- Shellphish
- The DARPA Cyber Grand Challenge
- Shellphish’s Cyber Reasoning System
  - Automatic Vulnerability Discovery
    ○ Angr → Live demonstration!
- Towards the Cyber Grand Challenge Finals
Shellphish CRS

A Dozen Years of Shellphish – from DEFCON to the Cyber Grand Challenge
Shellphish CRS

20

A Dozen Years of Shellphish – from DEFCON to the Cyber Grand Challenge
Agenda

- Shellphish
- The DARPA Cyber Grand Challenge
- Shellphish’s Cyber Reasoning System
- **Automatic Vulnerability Discovery**
  - Angr → Live demonstration!
- Towards the Cyber Grand Challenge Finals
Automatic Vulnerability Discovery

“How do I crash a binary?”

“How do I trigger a condition X in a binary?”

Dynamic Analysis/Fuzzing  Symbolic Execution
Dynamic Analysis/Fuzzing

- How do I trigger the condition: “You win!” is printed?

```python
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```
Dynamic Analysis/Fuzzing

● How do I trigger the condition: “You win!” is printed?

```python
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```

● Try “1” → “You lose!”
● Try “2” → “You lose!”
● …
● Try “10” → “You win!”
Dynamic Analysis/Fuzzing

- How do I trigger the condition: “You win!” is printed?

```python
x = int(input())
if x >= 10:
    if x == 123456789012:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```
Symbolic Execution

- Interpret the binary code, using symbolic variables for user-input

```python
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```

<table>
<thead>
<tr>
<th>State A</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Variables</strong></td>
</tr>
<tr>
<td>x = ???</td>
</tr>
<tr>
<td><strong>Constraints</strong></td>
</tr>
<tr>
<td>{}</td>
</tr>
</tbody>
</table>
Symbolic Execution

- Follow all feasible paths, tracking "constraints" on variables

```python
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```
Symbolic Execution

- Follow all feasible paths, tracking "constraints" on variables

```python
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```

<table>
<thead>
<tr>
<th>State AA</th>
<th>State AB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Variables</strong></td>
<td><strong>Variables</strong></td>
</tr>
<tr>
<td><code>x = ???</code></td>
<td><code>x = ???</code></td>
</tr>
<tr>
<td><strong>Constraints</strong></td>
<td><strong>Constraints</strong></td>
</tr>
<tr>
<td><code>{x &gt;= 10}</code></td>
<td><code>{x &lt; 10}</code></td>
</tr>
</tbody>
</table>
Symbolic Execution

- Follow all feasible paths, tracking "constraints" on variables

```python
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```

<table>
<thead>
<tr>
<th>State AA</th>
<th>Variables</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x = ???</td>
<td>{x &gt;= 10}</td>
</tr>
</tbody>
</table>
Symbolic Execution

- Follow all feasible paths, tracking "constraints" on variables

```python
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```

<table>
<thead>
<tr>
<th>State AA</th>
<th>Variables</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x = ???</td>
<td>{x &gt;= 10}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State AAA</th>
<th>Variables</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x = ???</td>
<td>{x &gt;= 10, x &lt; 100}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State AAB</th>
<th>Variables</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x = ???</td>
<td>{x &gt;= 10, x &gt;= 100}</td>
</tr>
</tbody>
</table>
Symbolic Execution

- Concretize the constraints on the symbolic variables

```python
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```

State AAA

<table>
<thead>
<tr>
<th>Variables</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = ???</td>
<td>{x &gt;= 10, x &lt; 100}</td>
</tr>
</tbody>
</table>

Concretization

State AAA

<table>
<thead>
<tr>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 99</td>
</tr>
</tbody>
</table>
Symbolic Execution

- How did we use Symbolic Execution for CGC?
Symbolic Execution

- How did we use Symbolic Execution for CGC?
- Symbolically execute the binaries checking if one of these two conditions is true

- Memory accesses outside allocated regions
- "Unconstrained" instruction pointer (e.g., controlled by user input)
Symbolic Execution

- How did we use Symbolic Execution for CGC?
- Symbolically execute the binaries checking if one of these two conditions is true
  
  - Memory accesses outside allocated regions
  - “Unconstrained” instruction pointer (e.g., controlled by user input)
  
- We used the symbolic execution engine of Angr:
  the binary analysis platform developed at UCSB
Agenda

- Shellphish
- The DARPA Cyber Grand Challenge
- Shellphish’s Cyber Reasoning System
- Automatic Vulnerability Discovery
  - Angr → Live demonstration!
- Towards the Cyber Grand Challenge Finals
Angr

- Binary analysis platform, developed at UCSB

- Open-source: https://github.com/angr (please “star” it!)

- Written in Python!
  - installable with one single command!
  - interactive shell (using IPython)

- Architecture independent
  - x86 (ELF, CGC, PE), amd64, mips, mips64, arm, aarch64, ppc, ppc64
Angr – Demonstration

- CADET_00001
### Angr – Demonstration

**CADET_00001**

```
08048080 push    ebp
08048081 mov     ebp, esp
08048083 sub     esp, 48h
08048086 mov     eax, 1
0804808B lea     ecx, ascii_8048530 ; "\nWelcome to Palindrome Finder\n\n"
08048091 mov     edx, 1Fh
08048096 mov     dword ptr [ebp-4], 0
0804809D mov     dword ptr [esp], 1
080480A4 mov     [esp+4], ecx
080480A8 mov     dword ptr [esp+8], 1Fh
080480B0 mov     [ebp-0Ch], eax
080480B3 mov     [ebp-10h], edx
080480B6 call    sub_8048360

08048360 push    ebp
08048361 mov     ebp, esp
08048363 push    esi
08048364 sub     esp, 34h
08048367 mov     eax, [ebp+10h]
0804836A mov     ecx, [ebp+0Ch]
0804836D mov     edx, [ebp+8]
08048370 mov     [ebp-0Ch], edx
08048373 mov     [ebp-10h], ecx
08048376 mov     [ebp-14h], eax
08048379 mov     dword ptr [ebp-18h], 0
08048380 mov     dword ptr [ebp-1Ch], 0
08048387 cmp     dword ptr [ebp-10h], 0
0804838E jnz     loc_80483A0
```
Angr – Demonstration

- CADET_00001: a classic buffer overflow

```c
int check()
{
    char string[64];
    receive_delim(0, string, 128, '
')

    //check if the string is palindrome
    //...

    return result;
}
```
Angr – Demonstration

- CADET_00001: a classic buffer overflow

```python
import angr
p = angr.Project("CADET_00001")
pg = p.factory.path_group(immutable=False,
  save_unconstrained=True)
while len(pg.unconstrained)==0:
  pg.step()
  crash_state = pg.unconstrained[0].state
  crash_state.posix.dumps(0)
```
CADET_00001: triggering the "Easter Egg"

```c
#define EASTEREgg "\n\nEASTER EGG!\n\n"

// the "caret" character ("^") triggers the Easter Egg
if (string[0] == '^'){
    transmit_all(1, EASTEREgg, sizeof(EASTEREgg) - 1)
}
```
Angr – Demonstration

- CADET_00001: triggering the “Easter Egg”
Angr – Demonstration

- CADET_00001: triggering the “Easter Egg”

```python
import angr
p = angr.Project("CADET_00001")
pg = p.factory.path_group(immutable=False)
pg.explore(find=0x804833E)
pg.found[0].state.posix.dumps(0)
```
Agenda

- Shellphish
- The DARPA Cyber Grand Challenge
- Shellphish’s Cyber Reasoning System
- Automatic Vulnerability Discovery
  - Angr → Live-demonstration!
- Towards the Cyber Grand Challenge Finals
CGC Quals – Results

- 7 teams passed the qualification phase
- Shellphish is one of them! :-)
- We exploited 44 binaries out of 131
- Every qualified team received 750,000$!
The system will need to be 100% automated
- no possibility of bug fixing after competition’s start

Partially different rules
- An exploit needs to
  - set a specific register to a specific value
  - leak data from a specific memory region
  - we need to implement more “realistic” exploits
    - **Angr** automatic ROP-chain builder!

- Network-level monitoring and defenses
CGC Finals

- Every team will have access to a cluster of:
  - 1280 cores
  - 16 TB of RAM
  - 128 TB of storage
CGC Finals

- Money prices!
  - First place: 2,000,000$
  - Second place: 1,000,000$
  - Third place: 750,000$

- The winning team will compete against human teams at DEFCON CTF Finals :-)}
Questions?

References:
this presentation: http://goo.gl/3ulxRa
angr: https://github.com/angr/angr
HITCON Community talk: Saturday, August 29th (14:20 – 15:10)
emails: antoniob@cs.ucsb.edu – yans@cs.ucsb.edu