Welcome to Hacknight

Offensive Security, Incident Response, and Internet Security Laboratory

Room 219 of Rogers Hall,
5 MetroTech Center, Brooklyn

osiris@osiris.cyber.nyu.edu

Come hang out with us!

Hiring partners:

Google, Facebook, IBM, Bloomberg, Capsule8, BAE Systems, Raytheon, NSA, Palo Alto Networks, Riot Games, Square, Salseforce, JPMorgan Chase & Co, Etsy, MWR Labs, Include Security, NCC Group, Uber, Cisco, Etsy, CTFd, Synack, FBI, OPTIV, TD Bank, Red Balloon Security, Vector35 LLC, Trail of Bits, New York City Cyber Command....

OSIRIS

Kent Ma

About me

- CSAW CTF Lead
- Email: kent@osiris.cyber.nyu.edu
- I like to cheat in video games

OSIRIS

Hacknight

- Lectures and workshops
  - RH217 every Thursday at 7 PM
  - Workshops will be hosted on wargames.osiris.cyber.nyu.edu
  - You're expected to bring your laptop
- We will cover material both from and beyond the scope of the following classes:
  - CS4773 - Penetration Testing and Vulnerability Analysis
  - CS3943 - Intro to Offensive Security

Warning

This course will be teaching potentially dangerous techniques. By staying here, you agree that all of the knowledge gained will be used in an ethical and safe manner.

The material taught in this course can cause harm to people and computer systems. If you cause harm either accidentally or intentionally, **it will get you fined or in jail.**

There are many other fields of security, but our focus will be on offensive application security

Why?

There are many other fields of security, but our focus will be on offensive application security

Why?

- We can make developers cry

There are many other fields of security, but our focus will be on offensive application security

Why?

- We can make developers cry
- How can you defend against something you don't understand?

OSIRIS

Overall Outline

Week 1	Introduction
Week 2, 3	Server-Side Web Security
Week 4	Client-Side Web Security
Week 5, 6, 7	Reverse engineering
Week 8, 9, 10	Memory Corruption
Week 11, 12	Heap Exploitation
Week 13	Kernel Exploitation

OSIRIS

Week 1 - Introduction

Kent Ma

OSIRIS

Terminology

Vulnerability - A bug that has potential to be used in an exploit

Exploit (noun) - Crafted data of that uses a bug to make software do something Unintended

0day - Unknown or unpatched vulnerability that can be used by an exploit

CVE - Public database of vulnerabilities run by the MITRE corporation. You'll see vulnerabilities indexed as CVE-<year discovered>-<some ID number> e.x. CVE-2014-6271

Outline

- What are we actually looking for?
  - Types of bugs
- Where do we look?
  - Targets
  - Boundaries
- How do we look?
  - Tools
  - Methods of looking

OSIRIS

What are we looking for?

Types of Flaws

The rest of Hack Night will be focused on these bugs in depth with more advanced techniques

- Design flaws
- Operational flaws
- Implementation bugs

OSIRIS

Design Flaws

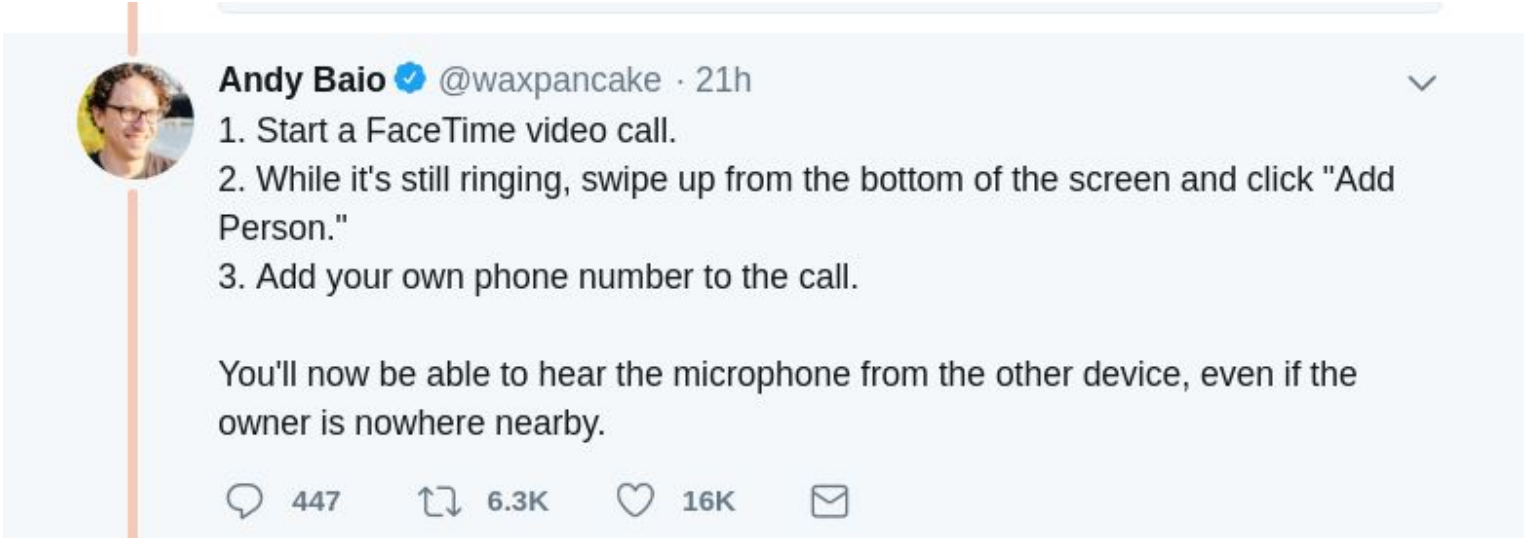
- Fundamental flaws in application architecture
- Application is working exactly as designed - for better or worse
- These are often subtle, devastating, and a massive pain to fix

Types of Design Flaws

- Authentication flaws
- Authorization flaws
- Business logic flaws
- Insecure cryptography

OSIRIS

Authentication Bypass - Facetime



OSIRIS

Authorization - Dominos App

- Authorization for payment of order is done client-side

  Change request sent to the server for payment from:

  ```
  <reason>DECLINED</reason>
  ```

  Changed to

  ```
  <reason>ACCEPTED</reason>
  ```

Design Flaws - Authentication & Authorization

When do they happen?

- Lack of authentication/authorization
- Backdoors/Default creds
- Granting privilege to lower privileged users
- Confusion between authorization contexts
- Logic mistakes

Business Logic Bugs

- Multi-step processes happening out of order
- Race conditions
- Time of check vs time of use (TOCTOU)

Business Logic Bug - libssh Auth Bypass

“**By presenting the server an SSH2_MSG_USERAUTH_SUCCESS** message in place of the SSH2_MSG_USERAUTH_REQUEST message which the server would expect to initiate authentication, the attacker could successfully authentciate [sic] without any credentials.” (CVE-2018-10933)

- What does this mean?

OSIRIS

Business Logic Bug - libssh Auth Bypass

Normal interaction

- Client: Hello. I want to log in. Here's my username and password
- Server: Your username and password is correct. We can start talking
  Server: Ok, What's your command?
- Client: Here's my command
- Server: Ok, What's your command?

Business Logic Bug - libssh Auth Bypass

????

- Client: Hello. Your username and password is correct. We can start talking
  Client: Ok, What's your command?
- Server: Ok, What's your command?
- Client: Here's my command
- Server: Ok, What's your command?

OSIRIS

Business Logic Bugs - Starbucks

- Starbucks.com has personal accounts for gift cards
- API endpoints to hit to transfer money:

```
starbucks.com/step1?amount=<amount>&from=wallet1&to=wallet2
```

OSIRIS

Business Logic Bugs - Starbucks

`starbucks.com/step1?amount=<amount>&from=wallet1&to=wallet2`

The code might look something like this:

```
if from.balance > amount:
    from.balance -= amount
    to.balance += amount
```

OSIRIS

Time of Check vs Time of Use (TOCTOU)

What happens if multiple transactions happen at once?

wallet1 has 10 dollars on it

```
starbucks.com/step1?amount=9&from=wallet1&to=wallet2
starbucks.com/step1?amount=9&from=wallet1&to=wallet2
```

OSIRIS

Time of Check vs Time of Use (TOCTOU)

```
from has 10
to has 0

if from.balance > amount:
     from.balance -= amount
     to.balance += amount
```

```
if from.balance > amount:
     from.balance -= amount
     to.balance += amount
```

OSIRIS

Time of Check vs Time of Use (TOCTOU)

```
from has 10
to has 0
```

```
if from.balance > amount:
     from.balance -= amount
     to.balance += amount
```

```
if from.balance > amount:
     from.balance -= amount
     to.balance += amount
```

OSIRIS

Time of Check vs Time of Use (TOCTOU)

```
from has 10
to has 0

if from.balance > amount: # 10 > 9
     from.balance -= amount
     to.balance += amount
```

```
if from.balance > amount:
     from.balance -= amount
     to.balance += amount
```

OSIRIS

Time of Check vs Time of Use (TOCTOU)

```
from has 1
to has 0

if from.balance > amount:
     from.balance -= amount
     to.balance += amount
```

```
if from.balance > amount:
     from.balance -= amount
     to.balance += amount
```

OSIRIS

Time of Check vs Time of Use (TOCTOU)

```
from has 0
to has 9

if from.balance > amount:
     from.balance -= amount
     to.balance += amount

Done!
```

```
if from.balance > amount:
     from.balance -= amount
     to.balance += amount
```

OSIRIS

Time of Check vs Time of Use (TOCTOU)

```
from has -8
to has 9

if from.balance > amount:
     from.balance -= amount
     to.balance += amount

Done!
```

```
if from.balance > amount:
     from.balance -= amount
     to.balance += amount
```

OSIRIS

Time of Check vs Time of Use (TOCTOU)

```
from has -8
to has 18

if from.balance > amount:
     from.balance -= amount
     to.balance += amount

Done!
```

```
if from.balance > amount:
     from.balance -= amount
     to.balance += amount

Done!
```

OSIRIS

Time of Check vs Time of Use (TOCTOU)

```
from has -8
to has 18

if from.balance > amount:
     from.balance -= amount
     to.balance += amount

Done!
```

```
if from.balance > amount:
     from.balance -= amount
     to.balance += amount

Done!
```

Real bug in Starbucks: https://sakurity.com/blog/2015/05/21/starbucks.html

Operational Flaws

Dumb mistakes that get you bug bounties

- Misconfigurations
- Insecure defaults
- Exposing sensitive external components
- Missing mitigations

Operational Flaws

- We won’t focus too much on these
- If you notice potential for one when using/auditing a web application, scan the internet for it for $$$
- https://www.shodan.io/

Implementation Bugs

- Failures in implementation that cause the application to perform incorrectly
  - Invalid input causing **undefined behavior**
  - Misuse of API
  - Miscalculation
  - No verification

Injection Bugs

Several bugs classes follow this same idea

- Special characters which have no meaning in one context can have meaning in others
- Look for communication/context switches between different execution

OSIRIS

Injection Bugs - Command Injection

```
os.system(“ping ” + ip)
```

OSIRIS

Injection Bugs - Command Injection

```
os.system(“ping ” + ip)
ip = “127.0.0.1”
```

OSIRIS

Injection Bugs - Command Injection

```
os.system(“ping ” + ip)
```

```
Our command is now: “ping 127.0.0.1”
```

OSIRIS

Injection Bugs - Command Injection

```
os.system(“ping ” + ip)
ip = “127.0.0.1; cat /etc/passwd”
```

OSIRIS

Injection Bugs - Command Injection

```
os.system(“ping ” + ip)
```

```
Our command is now: “ping 127.0.0.1; cat /etc/passwd”
```

OSIRIS

Injection Bugs - Command Injection

```
os.system("ping " + ip)
```

```
Our command is now: "ping 127.0.0.1; cat /etc/passwd"
```

OSIRIS

Injection Bugs - Command Injection

```
os.system("ping " + ip)
```

```
Our command is now: "ping 127.0.0.1; cat /etc/passwd"
```

What gets run?

- `ping 127.0.0.1` <- completes our original ping command

OSIRIS

Injection Bugs - Command Injection

```
os.system("ping " + ip)
```

```
Our command is now: "ping 127.0.0.1; cat /etc/passwd"
```

What gets run?

- `ping 127.0.0.1` <- completes our original ping command
- `cat /etc/passwd` <- reads password file

OSIRIS

Injection Bugs - Command Injection

```
os.system(“ping ” + ip)

Our command is now: “ping 127.0.0.1; cat /etc/passwd”
```

- The ; character allows multiple commands to be run in sequence

OSIRIS

Injection Bugs - Command Injection

```
os.system(“ping ” + ip)

Our command is now: “ping 127.0.0.1; cat /etc/passwd”
```

- The ; character allows multiple commands to be run in sequence
- Occurs because the IP string is moved to a different context (shell with system()) where it has different behavior

OSIRIS

Integer Bugs - Underflow

```
unsigned int level = 0;
level--;
print(level);
```

OSIRIS

Integer Bugs - Underflow

```
unsigned int level = 0;
level--;
print(level);
```

4294967295

OSIRIS

Integer Bugs - Overflow

```
int level = 2147483647;
level++;
print(level);
```

OSIRIS

Integer Bugs - Overflow

```
int level = 2147483647;
level++;
print(level);
```

-2147483648

OSIRIS

Integer Underflow?

```
unsigned short level = 1;
if ((level - 5) < 0) {
    login();
}
```

- Does `login()` get called?
- `level` should underflow to **`MAX_INT`**, shouldn't it?

OSIRIS

Integer Underflow?

```
unsigned short level = 1;
if ((level - 5) < 0) {
    login();
}
```

- Trick question! Login *is* called
- What type is everything?

OSIRIS

Integer Underflow?

```
unsigned short level = 1;
if ((level - 5) < 0) {
    login();
}
```

- level is unsigned short

OSIRIS

Integer Underflow?

```
unsigned short level = 1;
if ((level - 5) < 0) {
    login();
}
```

- level is unsigned short
- What type is 5? It's an int

OSIRIS

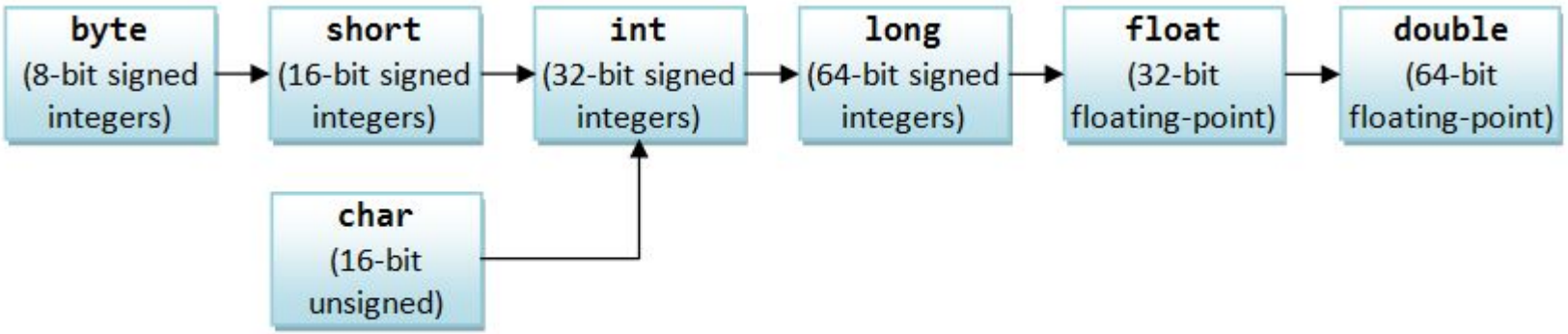
Integer Underflow?

```
unsigned short level = 1;
if ((level - 5) < 0) {
    login();
}
```

- level is unsigned short
- What type is 5? It's an int
- What happens when you do something with two types of different sizes?

Integer Promotions

- Smaller type is converted to “Larger” type **even if sign-ness is different**

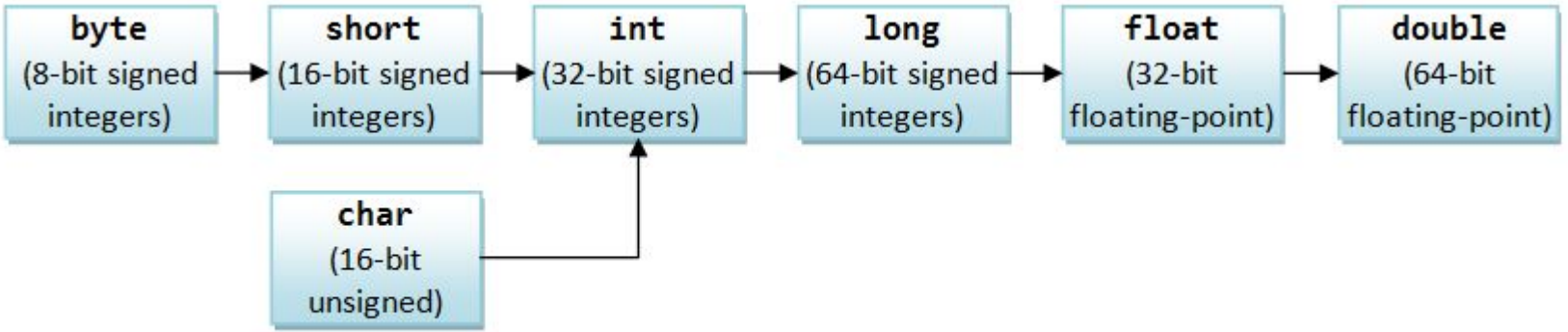


Orders of Implicit Type-Casting for Primitives

OSIRIS

Integer Promotions

- Smaller type is converted to “Larger” type **even if sign-ness is different**
- In our case of `(level - 5)`,
  - `level` - unsigned short
  - `5` - int



Orders of Implicit Type-Casting for Primitives

Memory Corruption

- The most classic implementation bug
- We'll have several Hack Night weeks dedicated to this topic
- For now, just keep in mind that segfaults in your code aren't "just errors"

OSIRIS

Where do we look?

There’s never enough time

- There’s a lot of code
- You start with little knowledge of the code
- People write ugly, hard-to-read code
- Timed engagements (pen testing times, deadlines, CTF end times, etc)

Smelly Traditional Targets

- Input sources
- Authentication/filtering mechanisms
- Complex things (protocols, parsing, etc)
- Coupled objects/classes/boundaries

Components and Boundaries

- What components are there?
- How do different parts talk?
- What separates different parts?

OSIRIS

Coupled Components - What can go wrong?

Two (or more) codependent parts that modify each other's state

- If one of them is used without the other, it might modify the other anyways
- Large and complex code bases often have these

Roles and Goals

- How are privileges defined?
- Which components need which privileges?
- How are users and groups used? (Authentication, identification)
- Are there ambiguities in roles for users?

Smelly Meta Targets

- Old/unmaintained code (check those copyright dates)
- Timing of code checked in
  - Near release dates
  - Around the same time as other known buggy code
- Developers that wrote bad code write more bad code elsewhere
- Previously buggy code
  - Don’t trust fixes!
- Things that directly tell you that the code sucks
  - FIXME, TODO, XXX, swearing, typos, github issues, etc

OSIRIS

How do we look?

Source Code Auditing

Reading source code to find vulnerabilities

- Can be tedious and kind of a pain
- Requires thorough understanding of the language

Tools:

- Source navigator - http://sourcenav.sourceforge.net/
- OpenGrok - https://oracle.github.io/opengrok/
- Scitools Understand - https://scitools.com/

Tactics - Search

Search through the code base for bad functions

Pros:

- Easy to do, takes little effort
- Fast
- Sometimes works

Cons:

- Tracing back to exploitable inputs is a pain/impossible
- Lots of noise
- Won’t give you more understanding of the application

Tactics - Input Tracing

Look at all sources of input and trace where they flow through the application

Ask yourself:

- What's the point of the input?
- How is the input used/modified?
- What checks are/aren't done?
- Where does the input go?

OSIRIS

Tactics - Modeling Boundaries

- Look at separate components of a target and diagram where they communicate
- Note authorization levels for communication between each component

Fuzz testing (Fuzzing)

Basic idea: run program with random inputs and record errors

- Works surprisingly well - lots of research into optimizing/improving fuzzing
- Think of it as searching over all possible inputs for any cases that are bugs

We won't go too deeply into fuzzing today, but you're free to chat with me afterwards about it

See https://labs.mwrinfosecurity.com/blog/what-the-fuzz/

OSIRIS

Workshop 1

wargames.osiris.cyber.nyu.edu