CPP-Summit 2019

全球C++软件技术大会

C++ Development Technology Summit

Boolan

高端IT互联网教育平台

关注“博览Boolan”服务号

发现更多 会议·课程·活动

Secure Coding Best Practices

Your First Line Is The Last Line Of Defense

MATTHEW BUTLER

CPP-SUMMIT • NOVEMBER 2, 2019

About Me

- Software architect and security researcher
  - Started with C++ professionally in 1990 (Borland C++ 1.0)
  - C++Now and CppCon staff
  - International conference speaker and trainer

- Areas of expertise:
  - Network and applications security
  - Safety critical systems
  - Real-time data analysis
  - Embedded systems

- Member of the ISO C++ Standards Committee
  - Evolution Working Group (EWG)
  - SG12 – Software vulnerabilities and safety critical systems
  - SG14 – Low Latency, embedded
  - SG21 – Contracts

Flying the Unfriendly Skies

The Most Dangerous Piece Of Real Estate

- Perimeter Security Will Not Protect You
  - They will get inside the wire on you
- There Are No Safe Spaces Any More
  - The rise of zero-trust environments
- Racing to the dark side
  - No longer just about money
  - Penetrations are becoming much more destructive

The Three Lies We Tell Ourselves

- We have firewalls
  - Fixed fortifications do not keep threat actors out
  - The focus now is on preventing data exfiltration
- But it's been code reviewed…and tested!
  - Most code reviews don't focus on secure coding
  - Security vulnerabilities require a special type of testing
- We're too <fill in the blank> to be a target
  - Large companies have more to steal
  - Small companies have weak security
  - Everyone has money and software scales really well

The Usual Suspects

- Nation States
  - Anyone with geo-political or geo-military ambitions
  - Spend billions developing zero-day exploits
  - And then they lose containment on them

- Criminal Intent
  - Use the zero-day exploits developed by nation states
  - They make weaponized exploits fashionable and affordable
  - Coming to a server near you

The Usual Suspects

- Spy vs Spy
  - Corporate espionage is rampant and profitable
  - Nortel Networks and Advanced Persistent Threats
  - Trickle down economics on the dark side

- Insiders
  - Insiders are still the largest source of data breaches world-wide
  - The weakest point in security is always human beings

Critical Systems

- What Is A Critical System?
  - The system itself
  - Any other system that can interact with it no matter how low the priority
    - Unrelated processes
    - Hardware (printers)
    - External systems

Attack Vectors

- Arbitrary Code Execution
  - Buffer overruns
  - Code pointer exploits
- Denial Of Service
  - Undefined behavior
- Privilege escalation
- Others
  - SQL Injection

Attack Surfaces

- Any External Facing Interface
  - Network connections
  - User interfaces
  - Authentication points
  - USB
- Also Internal Interfaces
  - IPC interfaces
  - Database engines
  - CLIs
- Security Is Built In Layers
  - The last layer is the code itself

Running with Scissors

What's The Vulnerability?


```
#define LOG_INPUT_SIZE 40

// saves the file name to a log file
int outputFilenameToLog(char *filename, int length)
{
        int success;

        // buffer with size set to maximum size for input to log file
        char buf[LOG_INPUT_SIZE];

        // copy filename to buffer
        strncpy(buf, filename, length);

        // save to log file
        success = saveToLogFile(buf);

        return success;
}
```

Arbitrary Code Execution

High
Likely
Medium

```
#define LOG_INPUT_SIZE 40

// saves the file name to a log file
int outputFilenameToLog(char *filename, int length)
{
        int success;

        // buffer with size set to maximum size for input to log file
        char buf[LOG_INPUT_SIZE];

        // copy filename to buffer
        strncpy(buf, filename, length);

        // save to log file
        success = saveToLogFile(buf);

        return success;
}
```

std::string


```
// saves the file name to a log file
int outputFilenameToLog(char *filename, int length)
{
        int success;

        // buffer is replaced with std::string so it's on the heap
        std::string buf = filename;

        // save to log file
        success = saveToLogFile(buf.c_str());

        return success;
}
```

Check & Terminate


```
#define LOG_INPUT_SIZE 40

// saves the file name to a log file
int outputFilenameToLog(char *filename, int length)
{
        int success;

        if (length > LOG_INPUT_SIZE - 1)
                return -1;

        // ...

        strncpy(buf, filename, length);
        buf[length] = 0;

        // ...

        return success;
}
```

Best Practices

- Maintain Situational Awareness

What's The Vulnerability?


```
enum EnumType {
        First,
        Second,
        Third
};

void f(int intVar)
{
        EnumType enumVar = static_cast<EnumType>(intVar);

        if (enumVar < First || enumVar > Third)
        {
                // Handle error
        }
}
```

Undefined Behavior

Medium
Unlikely
Medium

```
enum EnumType {
        First,
        Second,
        Third
};

void f(int intVar)
{
        EnumType enumVar = static_cast<EnumType>(intVar);

        if (enumVar < First || enumVar > Third)
        {
                // Handle error
        }
}
```

Strongly Typed Enumerations


```
enum class EnumType : int {
        First,
        Second,
        Third
};

void f(int intVar)
{
        EnumType enumVar = static_cast<EnumType>(intVar);
}
```

Best Practices

- **M**aintain Situational Awareness
- **S**tudy The Standard

What's The Vulnerability?


```
#include <algorithm>
#include <vector>

void f(const std::vector<int> &src)
{
        std::vector<int> dest;
        std::copy(src.begin(), src.end(), dest.begin());
        // ...
}
```

Undefined Behavior

High
Likely
Medium

```
#include <algorithm>
#include <vector>

void f(const std::vector<int> &src)
{
        std::vector<int> dest;
        std::copy(src.begin(), src.end(), dest.begin());
        // ...
}
```

back_inserter()


```
#include <algorithm>
#include <vector>

void f(const std::vector<int> &src)
{
        std::vector<int> dest;
        dest.reserve(src.size());

        std::copy(src.begin(), src.end(), std::back_inserter(dest));
        // ...
}
```

Direct Construction


```
#include <algorithm>
#include <vector>

void f(const std::vector<int> &src)
{
        std::vector<int> dest(src);
        // ...
}
```

Best Practices

- **M**aintain Situational Awareness
- **S**tudy The Standard
- **W**arnings Are Errors

What's The Vulnerability?


```
#include <cstdarg>

int add(int first, int second, ...)
{
        int r = first + second;
        va_list va;
        va_start(va, second);

        while (int v = va_arg(va, int)) {
                r += v;
        }
        va_end(va);

        return r;
}
```

Undefined Behavior

High
Likely
Medium

```
#include <cstdarg>

int add(int first, int second, ...)
{
        int r = first + second;
        va_list va;
        va_start(va, second);

        while (int v = va_arg(va, int)) {
                r += v;
        }
        va_end(va);

        return r;
}
```

Variadic templates


```
#include <type_traits>

template <typename Arg,
typename std::enable_if<std::is_integral<Arg>::value>::type * = nullptr>
        int add(Arg f, Arg s) {
                return f + s;
        }

template <typename Arg,
        typename... Ts,
        typename std::enable_if<std::is_integral<Arg>::value>::type * = nullptr>
        int add(Arg f, Ts... rest) {
                return f + add(rest...);
        }
```

Use Braced Initializer List Expansion


```
#include <type_traits>

template <typename Arg,
        typename... Ts,
        typename std::enable_if<std::is_integral<Arg>::value>::type * = nullptr>
        int add(Arg i, Arg j, Ts... all) {
                int values[] = { j, all... };
                int r = i;

                for (auto v : values) {
                        r += v;
                }

                return r;
        }
```

Best Practices

- **M**aintain Situational Awareness
- **S**tudy The Standard
- **W**arnings Are Errors
- **C**omplexity Is The Enemy

What's The Vulnerability?


```
auto g() {
   int i = 12;
   return [&]() {
      i = 100;
      return i;
   };
}

void f3() {
   int j = g()();
}
```

Undefined Behavior

High
Likely
High

```
auto g() {
   int i = 12;
   return [&]() {
      i = 100;
      return i;
   };
}

void f3() {
   int j = g()();
}
```

Capture By Value


```
auto g() {
   int i = 12;
   return [=]() mutable {
      i = 100;
      return i;
   };
}

void f() {
   int j = g()();
}
```

Best Practices

- **M**aintain Situational Awareness
- **S**tudy The Standard
- **W**arnings Are Errors
- **C**omplexity Is The Enemy
- **G**row Bug Bounty Hunters

This Is Not Bug Bounty Hunting

What I Look For When I Penetrate A System

- Anytime time you copy or move memory
  - You're likely to get it wrong
- Anytime you don't validate your data or verify who I am
  - I can send you anything and you'll choke on it
- The open source libraries you're using
  - Their weaknesses are your weaknesses

What I Look For When I Penetrate A System

- Internal interfaces you aren't guarding
  - IPC and command-line interfaces are great access points
- Any place I find complexity in your design
  - Complexity breeds vulnerabilities

The Art of War

Static Analysis Tools

- What Not To Rely On (For Security Testing)
  - Core Guidelines – Incomplete and not security centric
  - CppCheck – Some security checks mostly for undefined behavior
  - Clang Tidy/SA – Mostly language correctness, some security checks

	strncpy()	EnumType	std::copy	Variadic Function	Lambda [&]
Core Guidelines	✗	✗	✗	✗	✗
CppCheck	✗	✗	✗	✗	✗
Clang Tidy/SA*	✗	✗	✗	✓	✗

- These work best for code clarity and correctness

Secure Coding Static Analysis Tools

- Look for code & API errors
  - Much like an automated code review
  - Systematic
  - Can product a lot of false positives
- Tools
  - Coverity & Sonar, PVS-Studio – Commercial products
  - CERT Thread Safety Analysis – Already part of Clang
- Experimental tools
  - Rose Checkers – Specifically enforces the CERT C/C++ Coding Standard
  - AIR Integer Model – Integer overflow & truncation, proposed
  - Compiler–Enforced Buffer Overflow Elimination – In research stages

Secure Coding Static Analysis Tools

- Static Analysis is not a silver bullet
  - Static checkers are only as good as their rules
  - A lot of behavior is beyond the capabilities of static checkers
  - Can't detect environmental or 3rd party library vulnerabilities
- These work best for simple coding & API mistakes

Dynamic Analysis Tools

- Look for run-time vulnerabilities
  - Finds vulnerabilities in the run-time environment
  - Systematic
- Tools (GCC & Clang)
  - Asan – Address Sanitizer (buffer overflows, memory leaks)
  - Tsan – Thread Sanitizer (concurrency bugs)
  - Msan – Memory Sanitizer (uninitialized memory)
  - Ubsan – Undefined Behavior Sanitizer (UB)
- Can be used any time but really shine during performance, stress and scalability testing

Fuzz Testing

- Interface testing
  - Tests input interfaces for robustness with "random" data
  - Covers broad test ranges, cheaply
  - Can be difficult to deploy
- Tools
  - OSS-Fuzz – Google's continuous fuzzing service
  - libFuzzer – Coverage-guided fuzzing
  - American Fuzzy Lop – Uses genetic algorithms to generate test cases
- These work best for testing situational awareness

Penetration Testing

- Thinking like a hacker
  - Actively look for vulnerabilities in attack surfaces
  - Requires a significant level of security knowledge

- Tools
  - Metasploit Framework – Open source pen testing framework
  - Nmap – Attack surface mapping

Penetration Testing

- Other options
  - Employ white hat testers (see Black Hat)
  - Let the hackers help
- These work best for testing security models, external interfaces and architectures

Best Practices

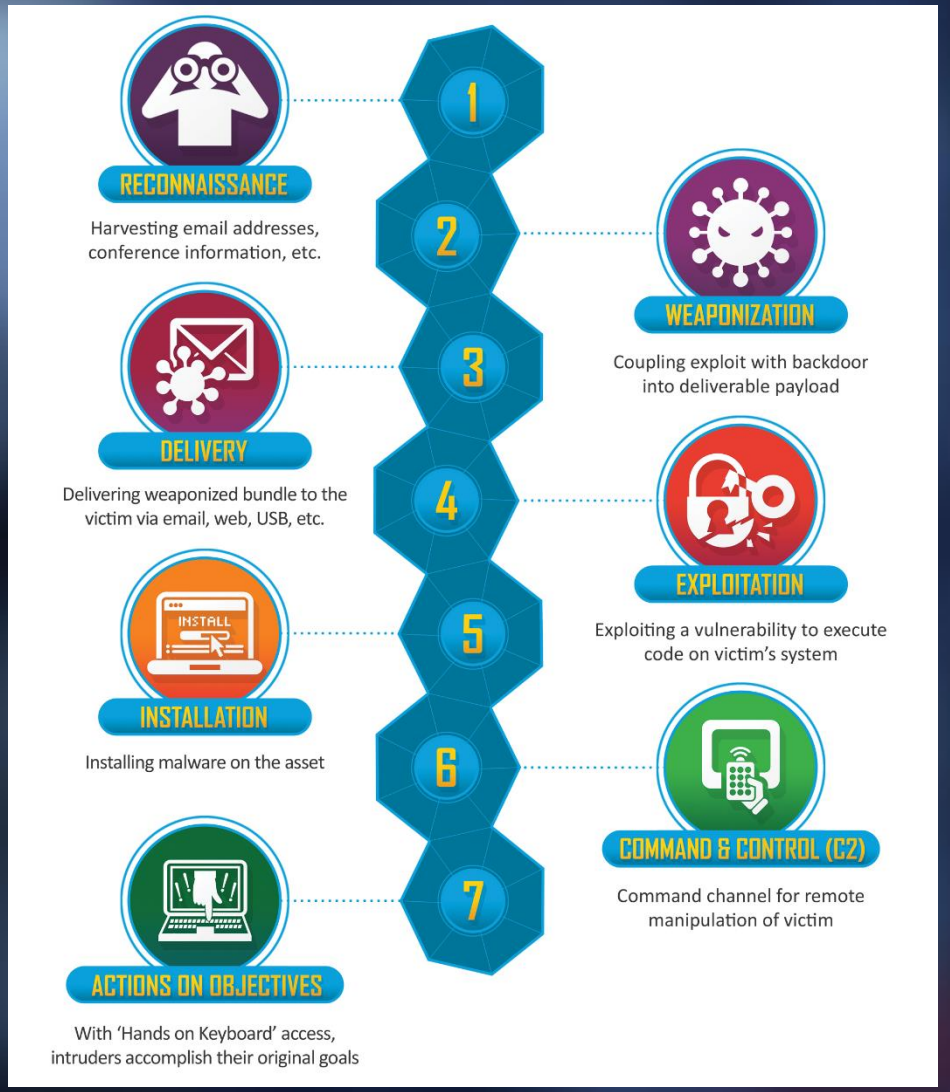
- **M**aintain Situational Awareness
- **S**tudy The Standard
- **W**arnings Are Errors
- **C**omplexity Is The Enemy
- **G**row Bug Bounty Hunters
- **A**ll Testing Is Asymmetrical

Threat Hunting

- Threat Hunting is about understanding the ***nature*** of threats
  - We do this every time we go somewhere unfamiliar
  - Darkness, isolation, insecurity, vulnerability
- We need to understand the ***nature*** of the attackers

Intrusion Kill Chains

- Reconnaissance
- Weaponization
- Delivery
- Exploitation
- Installation
- Command & Control (C^2)
- Actions On Objectives



Lockheed Martin Cyber Kill Chain

Threat Modeling Stages

- Scope Definition
  - Too narrow and we miss threats
  - Too broad and we get information overload
  - Use an iterative process – start narrowly and expand out
- Model Creation
  - How the data moves, where it pools
  - Serves as a reference design for later work
- Threat Identification
  - Where are the trust boundaries?
  - Are we holding data we don't need?
  - Where are we allowing access?

Threat Modeling Stages

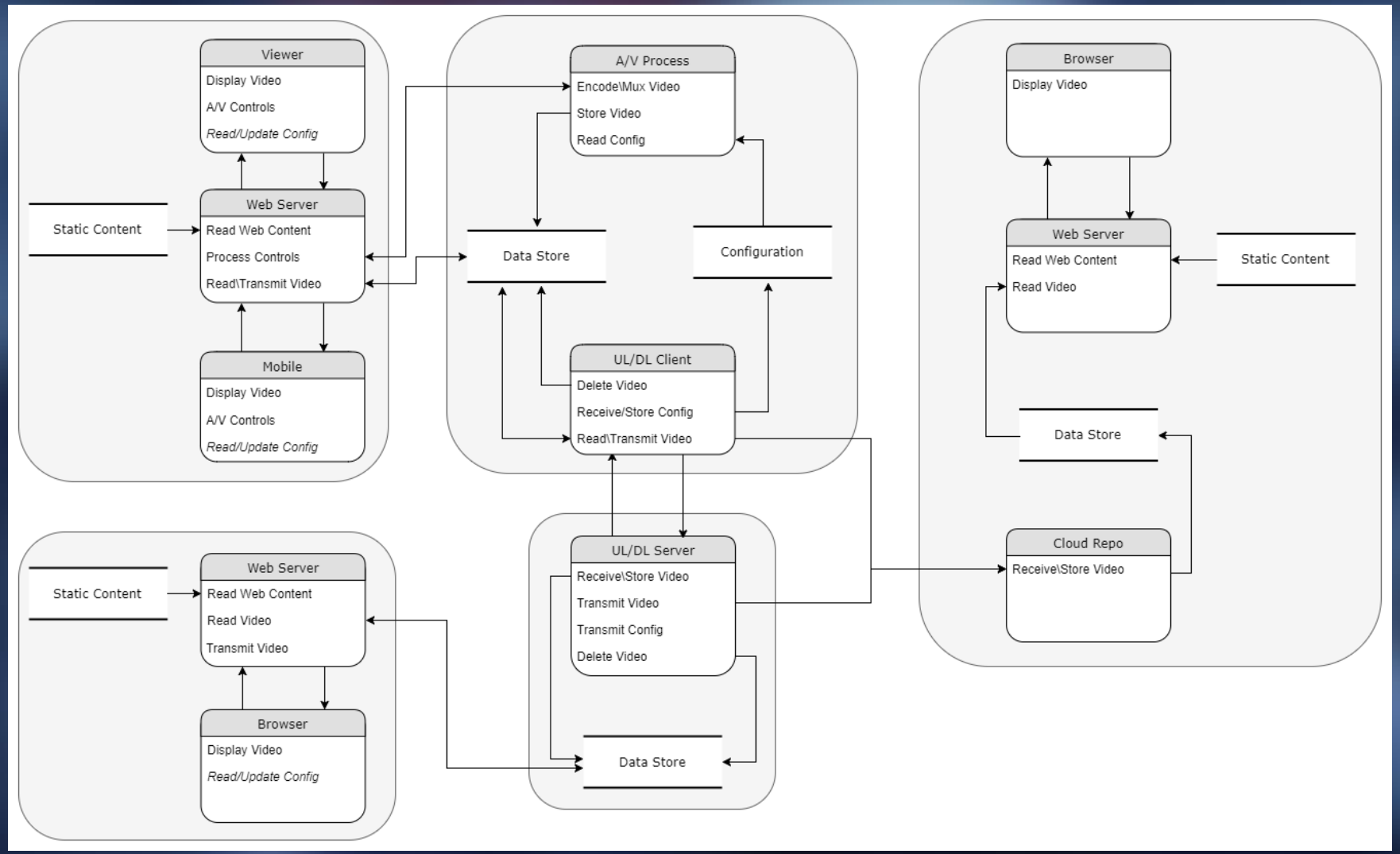
- Threat Classification
  - How much data can be exposed by this threat?
  - How much control can be gained by penetrating the system?
  - How easy it is to exploit what other systems can be affected?
- Mitigation Planning
  - Is this already mitigated or can we change the design?
  - Can we accept the risk of not fixing it?
  - Can we fix this in an acceptable timeframe (one iteration, for example)?
- Validation
  - Did this fix the problem, did it mitigate the threat?
  - Did this create other attack surfaces or new threats?
  - Did mitigating this threat expose other potential threats?

Trust Boundaries

- Trust Boundaries are fundamental to understanding risk
  - Different trust levels carry different risks
  - Trust and risk are inversely proportionate
  - The internet has high risk because it is low trust
  - Internal networks have low risk because we trust them

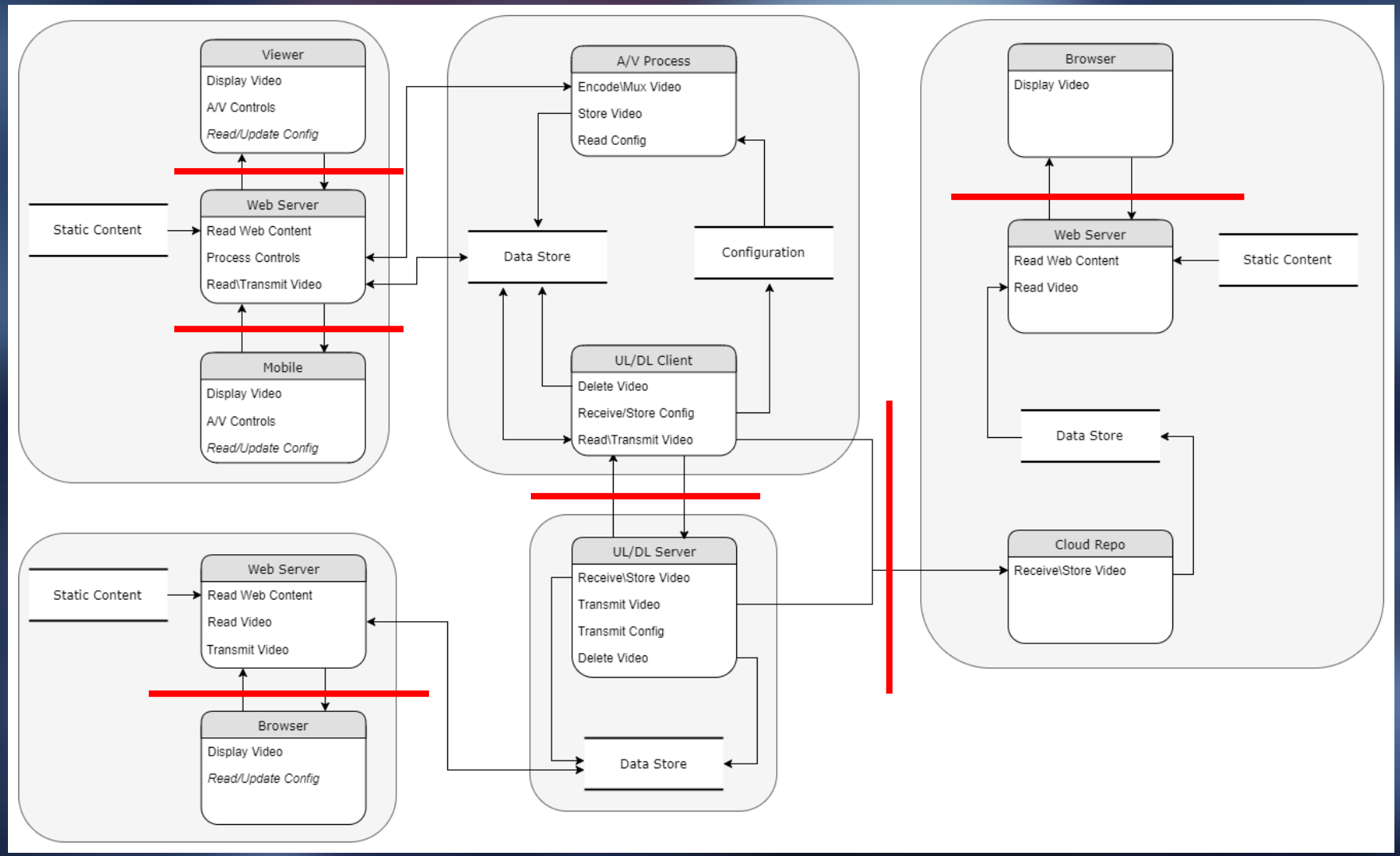
- Trust Boundaries occur when:
  - Data crosses from one level of trust to into another
  - Data crosses from an area were we control security to an area we don't

Remote Security DVR System



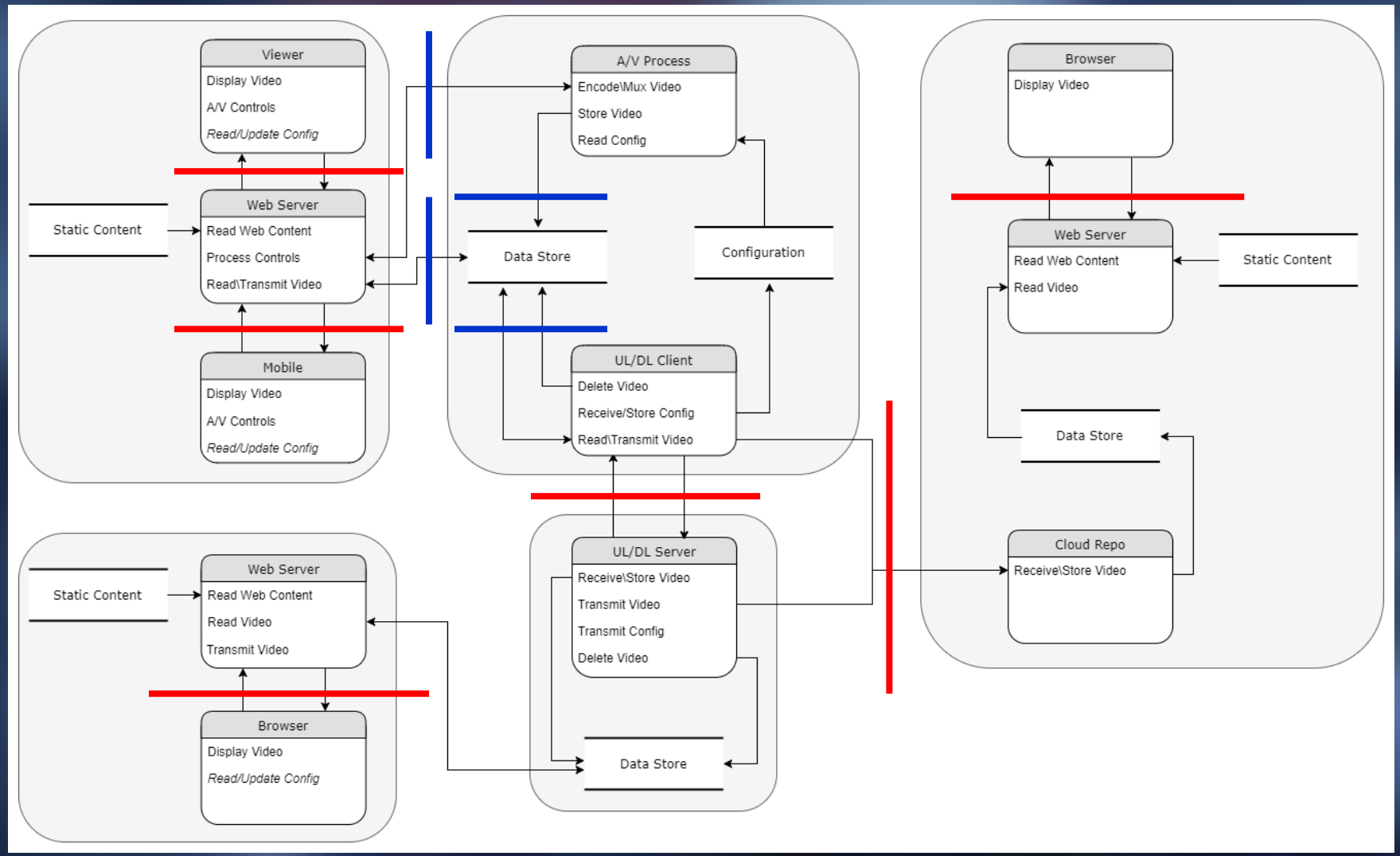
Where Are The Trust Boundaries?

Remote Security DVR System



Where Are The Trust Boundaries?

Remote Security DVR System

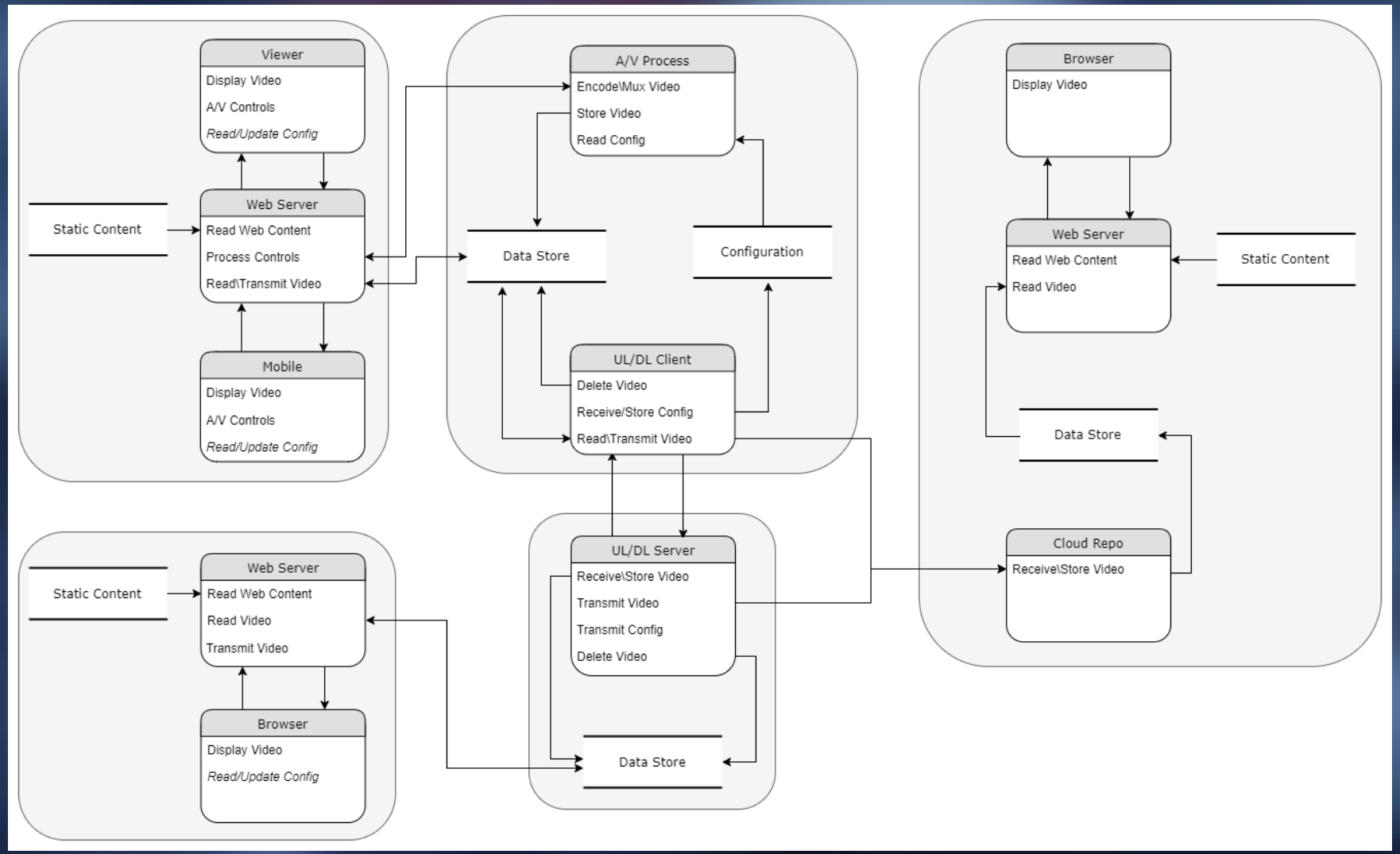


Where Are The Trust Boundaries?

STRIDE

- **S**poofing
  - Masquerading as another user
- **T**ampering
  - Sabotaging a system
- **R**epudiation
  - Acting without evidence of action
- **I**nformation Disclosure
  - Exfiltrating data out of a system
- **D**enial of Service
  - Rendering a system unusable
- **E**levation of Privilege
  - Gaining elevated access

Remote Security DVR System



Spoofing

Tampering

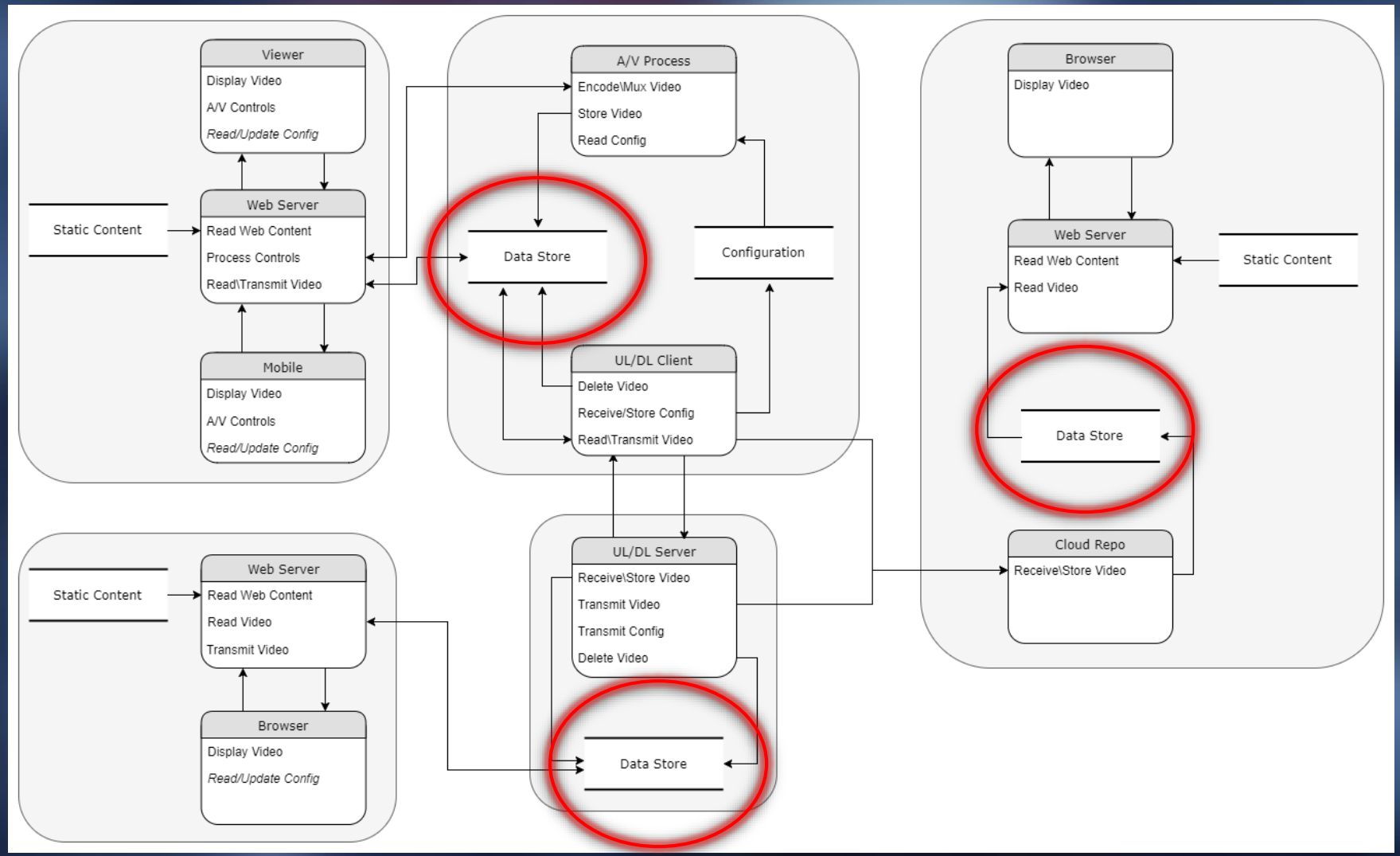
Repudiation

Information **D**isclosure

Denial of Service

Elevation of Privilege

Remote Security DVR System



Spoofing

Tampering

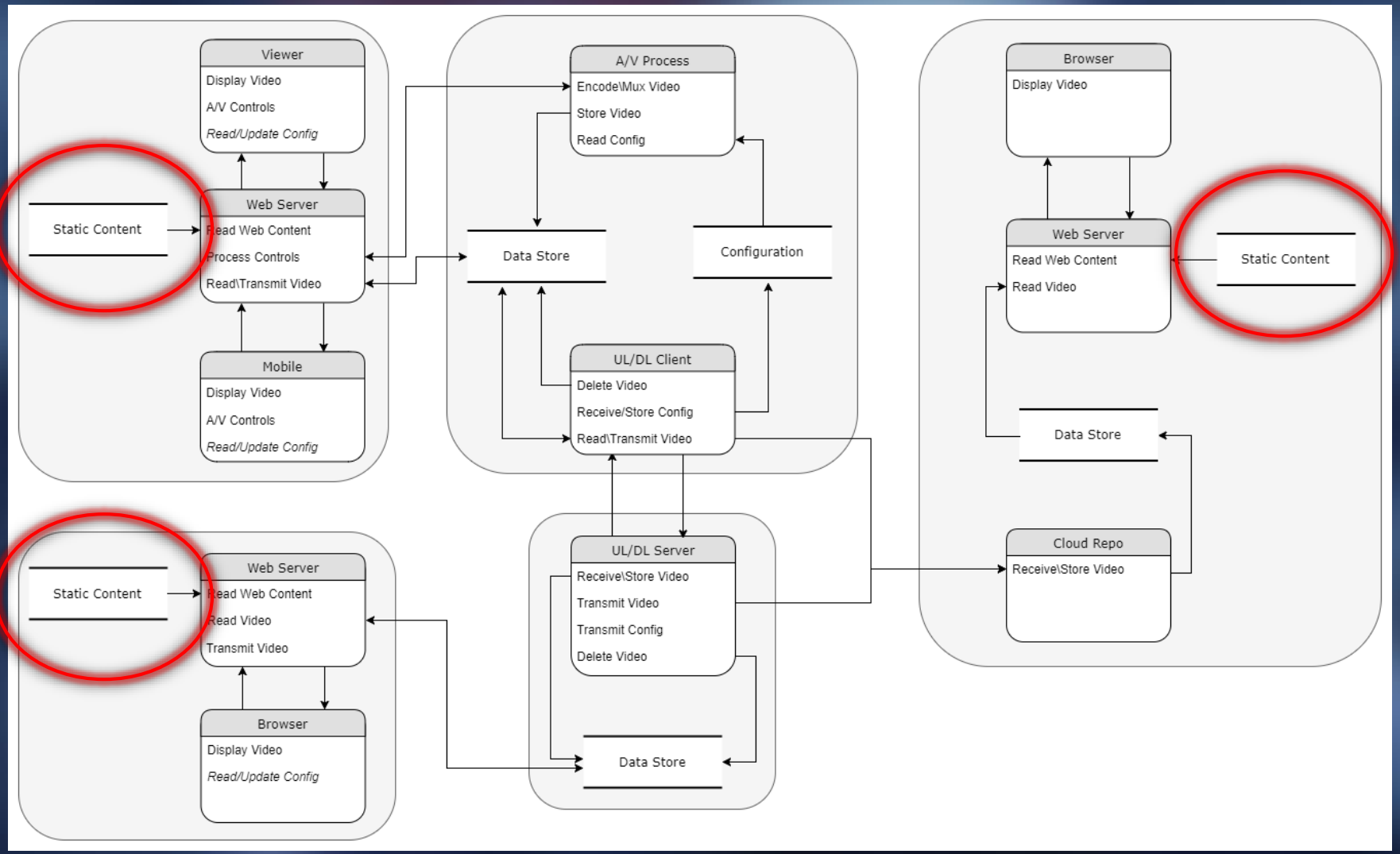
Repudiation

Information **D**isclosure

Denial of Service

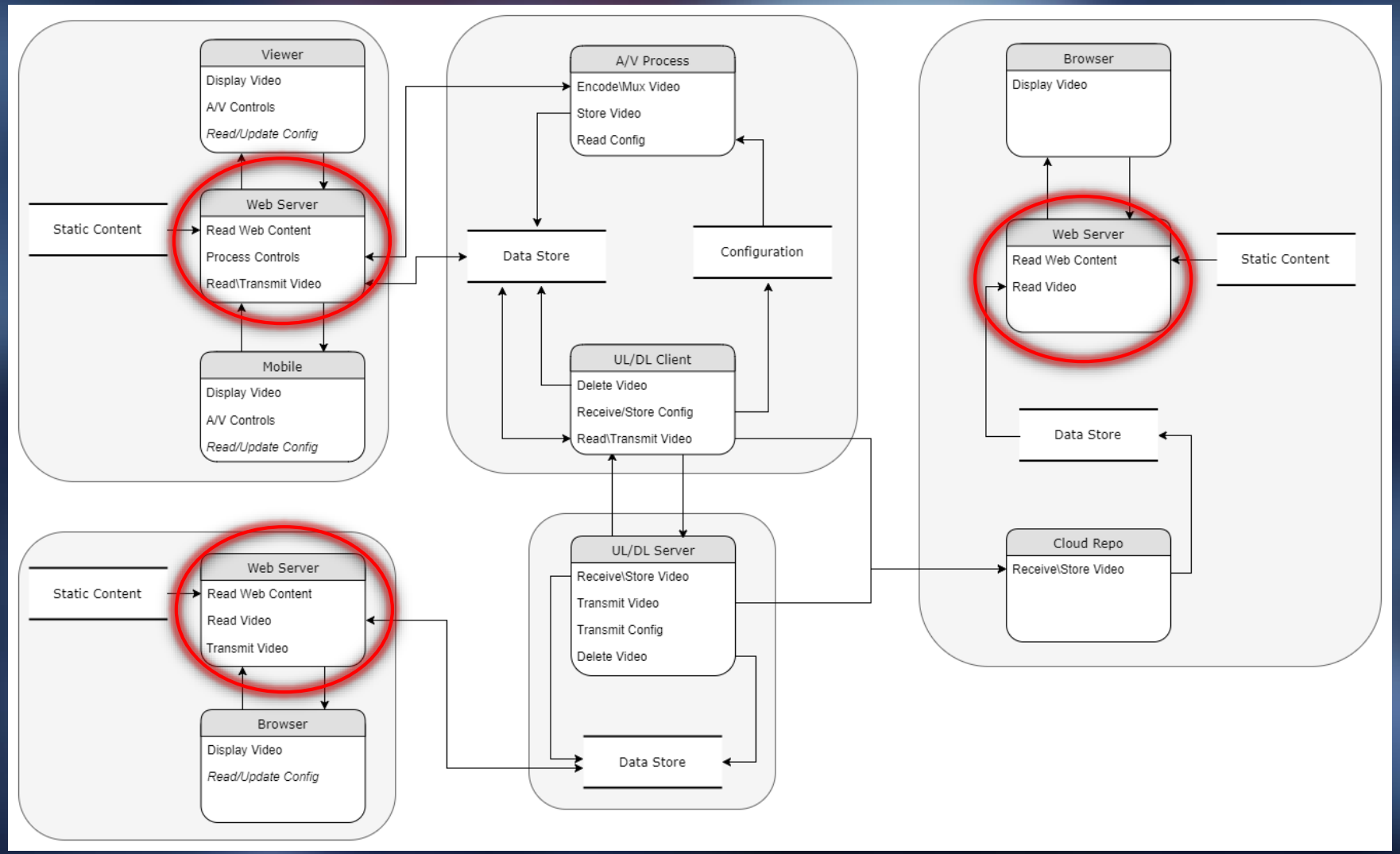
Elevation of Privilege

Remote Security DVR System



- Spoofing
- Tampering
- Repudiation
- **Information Disclosure**
- Denial of Service
- Elevation of Privilege

Remote Security DVR System



Spoofing

Tampering

Repudiation

Information Disclosure

Denial of Service

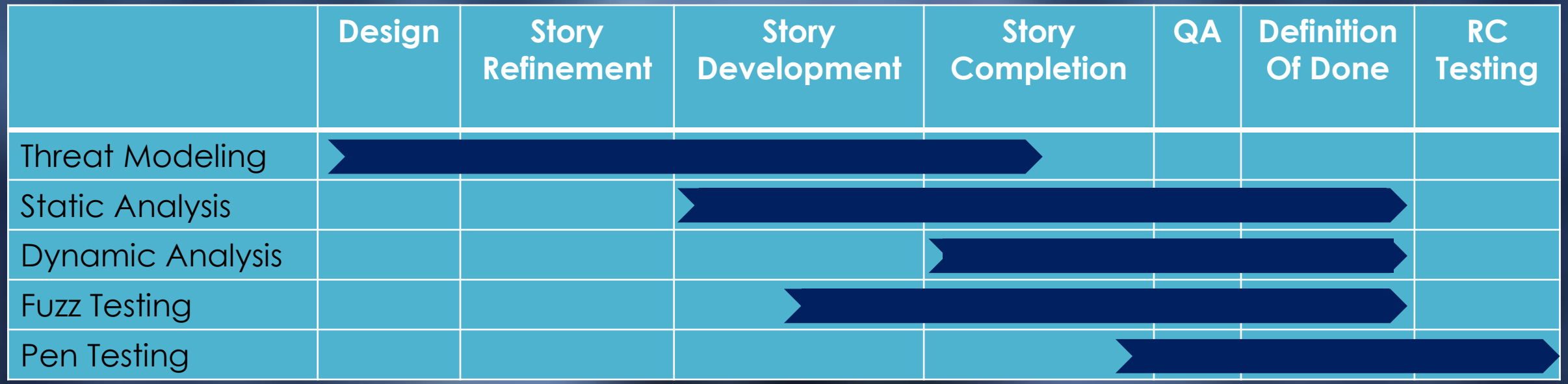
Elevation of Privilege

Take-Aways

- Zero trust environments
  - There are no safe spaces or trusted zones
- Exfiltration is becoming as important than infiltration
  - Focus on how data gets out of a system
- Complexity is the enemy
  - When you have to document it you find out how complex it really is
  - Complexity produces emergent behavior
- Threat Modeling does more than model threats
  - Exposes emergent behavior
  - Everyone now knows the system from end to end

When To Use What

There are no silver bullets…



Defense In Depth

- Build security in layers
- We live in zero-trust environments
- Don't mix trust levels
- Build security wrappers for 3rd party libraries
- Ex-filtration is now more important than infiltration

Security Layers

- Principle Of Least Privilege
  - Use the minimum privileges necessary
  - Grant & revoke privileges only as needed
  - Watch for exceptions and multiple returns
  - Privilege control is only one layer of protection
- Complexity Is The Enemy
  - Creates emergent behavior
  - Makes it hard to reason about our code
  - Understandability

Logging

- Log Memory From Segmentation Faults & Exceptions
  - The pattern of corrupted memory tells us something
  - Where/when it was corrupted tells us something as well
  - Digging through core files is non-trivial
- Log Security
  - We really can't afford to encrypt our logs
  - They're often a treasure trove of information
  - Sanitize what you put in them
- Audit Trails
  - Use security specific exceptions
  - Treat logging as an essential part of your security model

Best Practices

- **M**aintain Situational Awareness
- **S**tudy The Standard
- **W**arnings Are Errors
- **C**omplexity Is The Enemy
- **G**row Bug Bounty Hunters
- **A**ll Testing Is Asymmetrical
- **S**ecure Coding Starts With Secure Designs

Code Reviews

- Most reviews focus on form not function
  - Should focus on correctness, performance and security
  - +1s should not come from incremental commits
  - Fixing one bug may create another vulnerability
  - Be ruthless

- Legacy Code
  - Often has the worst vulnerabilities
  - Is generally not reviewed for security (if at all)
  - Implement security reviews specific to legacy code

Best Practices

- **M**aintain Situational Awareness
- **S**tudy The Standard
- **W**arnings Are Errors
- **C**omplexity Is The Enemy
- **G**row Bug Bounty Hunters
- **A**ll Testing Is Asymmetrical
- **S**ecure Coding Starts With Secure Designs
- **R**uthlessness Is A Virtue

The Problem With Third Party Libraries

- Libraries You Didn't Write
  - Open Source
  - Trust but verify with code reviews
  - Prefer libraries that have continual security audits
- Security Wrappers
  - Design wrapper classes for libraries you don't own
  - Include exception handling

Best Practices

- **M**aintain Situational Awareness
- **S**tudy The Standard
- **W**arnings Are Errors
- **C**omplexity Is The Enemy
- **G**row Bug Bounty Hunters
- **A**ll Testing Is Asymmetrical
- **S**ecure Coding Starts With Secure Designs
- **R**uthlessness Is A Virtue
- **Y**ou're Only As Strong As Your weakest Third Party Library

Best Practices

Grow Bug Bounty Hunters

Maintain Situational Awareness

Secure Coding Starts With Secure Designs

All Testing Is Asymmetrical

Be Committed

You're Only As Strong As Your weakest Third Party Library

Ruthlessness Is A Virtue

Complexity Is The Enemy

Study The Standard

Warnings Are Errors

Predator

Prey