CMPSC 497: Final Review

Trent Jaeger
Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University
Final

• Format
  ‣ True/False
    • 8 questions – 16 pts
  ‣ Short answer – word/phrase to sentence or two
    • 8 questions – 36 pts
  ‣ Question – conceptual (why?) and constructions (how?)
    • 8 questions – 58 pts
    • Conceptual – can be a bit open-ended
    • Constructions – like questions 11-13 from homework, but fewer parts
  ‣ Time should be less of an issue, but be careful
Final

• Cover entire course

• Know (remember) vulnerability, memory error, no-op sled, canary, buffer overflow, buffer overread, free vulnerabilities, name resolution attack, confused deputy, safe string processing, soundness/completeness in static analysis, path constraints in symbolic execution, fuzz testing

• Different types of fuzzing, ROP chain generation
Homework – Question #1

• Know the definitions of
  ‣ bounds check, control-flow integrity, low-fat pointer, privilege separation, reference monitor concept, authorization hook, software-fault isolation, stack inspection, bytecode verification, noninterference, implicit and explicit flows, and attack surface
Low-Fat Pointers

• Idea
  ‣ Hardware support for fat pointers

• Solutions
  ‣ Forgery – Hardware tags to prevent software from overwriting without detection
  ‣ Limited space – Do not really need entire 64-bit address space – use 46-bit address space and rest for metadata
  ‣ Performance – Hardware instructions to perform desired operations inline

• Result: Memory error protection for 3% overhead
Question #2

• What is spatial memory safety? What is temporal memory safety? Which one does SoftBound aim to address?
Question #2

• What is spatial memory safety? What is temporal memory safety? Which one does SoftBound aim to address?

› Space ➔ area

› Temporal ➔ time

› Which for softbound?
Question #3

• How do you compute a control-flow graph?
Question #3

• How do you compute a control-flow graph?
  ‣ Within a function
    • Sequences
    • Conditionals
    • Indirect jumps
  ‣ Between functions
    • Calls (direct and indirect)
    • Returns
Question #3

• How do you compute a control-flow graph?
  ▸ Indirect calls
  ▸ Returns
Signature-based CFI

• How do we compute the possible targets for function pointers?

• What are the expected targets of an indirect call?
  ‣ Functions with the same type signature as the function pointer
  ‣ Suppose you have a function pointer “int (*fn)(char *b, int n)”
    • Which functions should be assigned to that function pointer?

• Compute the set of functions that share that signature assuming any of these can be a target
  ‣ Fewer than all functions
  ‣ Intuitively seems like an overapproximation
  ‣ Can a function “void foo(void)” be assigned to the fpotr above?
Taint-based CFI

- For each function that may be assigned to a function pointer (e.g., foo)
  - We taint the variables assigned to propagate to find which are used at which indirect call site

- How we propagate the taint depends on the type of LHS
  - function pointer variable
  - function pointer element in an array
  - function pointer field in a structure
Question #4

• What is a shadow stack? How do you track the shadow stack at runtime?
Shadow Stack

• Method for maintaining return targets for each function call reliably

• On call
  ‣ Push return address on the regular stack
  ‣ Also, push the return address on the shadow stack

• On return
  ‣ Validate the return address on the regular stack with the return address on the shadow stack

• Why might this work? Normal program code cannot modify the shadow stack memory directly
Shadow Stack Exceptions

• Challenge: Exceptions
  ‣ There are cases where call-return does not match
  ‣ E.g., Tail calls, thread libraries (setjmp, longjmp)
Question #5

• What do you need to do to privilege separate two functions A and B, where A calls int B(char *buf, int size) to retain their functionality?
Motivation for Partitioning

- Split the application into multiple partitions
- Each partition is isolated using some isolation mechanism such as OS processes

Although some partition of a program has been hijacked, sensitive data can still be protected
PDG-based Partitioning: Example

Sensitive data

Partitioning boundary

Declassification
Selective Pointer Bounds Tracking

Step 1
Find pointers that are sent across the boundary

Step 2
Do backward propagation to find all BR pointers

We need to track the bounds of only the colored pointers
Question #6

- What needs to be protected to achieve the tamperproof guarantee of the reference monitor concept?
Reference Monitor Concept

• **Reference monitor concept** was defined in 1972 by James Anderson to describe design requirements on a “reference validation mechanism” (read authorization system)
  
  ‣ The reference validation mechanism *must always be invoked* (complete mediation).
  
  ‣ The reference validation mechanism must be tamperproof (tamperproof).
  
  ‣ The reference validation mechanism *must be small enough to be subject to analysis and tests*, the completeness of which can be assured (verifiable).
Authorization

• How is authorization integrated into an existing (host) program?

• Authorization system
  ‣ **Authorization hooks** – constructs authorization queries (subject, object, operation) and invokes reference monitor
    • Integrated into host program
  ‣ **Reference monitor module** – processes authorization queries into Y/N decisions using authorization policy
  ‣ **Authorization policy** – essentially a database relating subjects and objects to the operations that subjects are authorized to perform on objects
Tamperproof

- So what do these mean?
  - The reference validation mechanism must be tamperproof (tamperproof).
- The authorization system code, including modules and hooks, and data, including authorization policies, must only be writeable by trusted subjects
  - Ideally, such code and data should be set when the program is initiated (e.g., OS is booted) and remain unchanged throughout its execution.
Question #7

- Instrument a memory load (ld r1, r2(0)) and store (st r1(0), r2) operations to achieve software-fault isolation guarantees with minimal overhead for a data region from 0x50000 to 0x50FFF.
void interp(int pc, reg[], mem[], code[]) {
    while (true) {
        if (pc < CB) exit(1);
        if (pc > CL) exit(1);
        int inst = code[pc], rd = RD(inst), rs1 = RS1(inst),
               rs2 = RS2(inst), immed = IMMED(inst);
        switch (opcode(inst)) {
            case ADD: reg[rd] = reg[rs1] + reg[rs2]; break;
            case LD: int addr = reg[rs1] + immed;
                    if (addr < DB) exit(1);
                    if (addr > DL) exit(1);
                    reg[rd] = mem[addr];
                    break;
            case JMP: pc = reg[rd]; continue;
            ...
        }
        pc++;
    }
}
Enforcing SFI Policy

- Insert monitor code into the target program before unsafe instructions (reads, writes, jumps, ...)

```plaintext
[r3+12] := r4 //unsafe mem write

r10 := r3 + 12
if r10 < DB then goto error
if r10 > DL then goto error
[r10] := r4
```
Special Address Patterns

- Both code and data regions form contiguous segments
  - Upper bits are all the same and form a region ID
  - Address validity checking: only one check is necessary
- Example: DB = 0x12340000; DL = 0x1234FFFF
  - The region ID is 0x1234
  - “[r3+12]:= r4” becomes

```plaintext
r10 := r3 + 12
r11 := r10 >> 16 // right shift 16 bits to get the region ID
if r11 <> 0x1234 then goto error
[r10] := r4
```
Ensure, So No Check

- Force the upper bits in the address to be the region ID
  - Called **masking**
  - No branch penalty
- Example: DB = 0x12340000 ; DL = 0x1234FFFF
  - “[r3+12]:= r4” becomes

```
r10 := r3 + 12
r10 := r10 & 0x0000FFFF
r10 := r10 | 0x12340000
[r10] := r4
```

Force the address to be in data region
One-Instruction Masking

• Idea
  ‣ Make the region ID to have only a single bit on
  ‣ Make the zero-tag region unmapped in the virtual address space

• Benefit: cut down one instruction for masking

• Example: DB = 0x20000000 ; DL = 0x2000FFFF
  ‣ Region ID is 0x2000
  ‣ “[r3+12]:= r4” becomes

    r10 := r3 + 12
    r10 := r10 & 0x2000FFFF
    [r10] := r4

  ‣ Result is an address in DR or in the (unmapped) zero-tag region

• PittSField reported 10% performance gain for this optimization
Question #8

• Why do you need to perform bytecode verification before running a Java applet?
Three Pillars of Java Security

• Class loaders
  ‣ Loading classes into the JVM

• Bytecode verifier
  ‣ Perform dataflow analysis to verify type safety of bytecode

• Security manager
  ‣ Monitors dangerous operations such as file accesses
Bytecode Verification

• Ensure basic safety properties of the bytecode
  ‣ The file is not damaged
  ‣ The code is type safe
    • Which implies memory safety
  ‣ Does not prohibit the bytecode from reading your secret files
    • The job of the security manager

• Bytecode verifier
  ‣ a static verifier
  ‣ bytecode checked only once before running
  ‣ does not assume the bytecode is produced by a Java compiler; the bytecode could be written by an attacker
_bytecode verifier: multiple passes

• **Pass 1**: file integrity check
  ‣ Check the first four bytes is the magic hex number 0xCAFEBABE
  ‣ Check version numbers of the class file
  ‣ Check each structure in the file is of appropriate length

• **Pass 2**: structural checks
  ‣ Final classes are not subclassed
  ‣ Every class has a superclass (except for java.lang.Object)
  ‣ Final methods are not overridden
  ‣ ...

• **Pass 3**: type checking the code
  ‣ The most important and complicated pass
  ‣ Performs data-flow analysis to verify type safety
Question #9

• What is declassification and how does it prevent information flow errors? What is the risk in writing a declassifier?
• One may need to release data from a process that may be affected by secrets

• **Declassification** methods aims to remove secrets from data prior to release
  - How would we declassify medical records?
  - How would we declassify password verification responses?

• A password checking limit is a form of a **declassifier**, as it allows the leakage of some limited amount of secret information
do {  // password hash is secret
    if ( password_hash == hash(password) )
        return {public} authenticated;
    else {
        return {public} !authenticated;
    }
}
foo() {
...

    rtn = check_passwd(username, password, &try);
    ... while ( try < limit );

...}

do { // password hash is secret
    if ( password_hash == hash(password) )
        return {public} authenticated;

    else {
        *try = {public} (*try+1);
        return {public} !authenticated;
    }
}


Question #10

• How does an attack surface relate to a vulnerability?
Vulnerabilities

• … could be anywhere in a program
  ‣ Given the definition of a vulnerability, does that give us any insight into where we should look for vulnerabilities?
    • Software flaw
    • Accessible to an adversary
    • Who can exploit the vulnerability
Attack Surface Estimate

• For a program
• Identify UIDs that are adversaries of the program
• Identify objects (e.g., files) that those UIDs can write
• Identify program entry points that access those objects
  ‣ E.g., by running the program
• That set of entry points is the program attack surface
Answer questions about the code below related to control-flow integrity.

F1() { return; }
F2() { return; }
FA() { F1(); }
FB() { F2(); }
FC(fptr) { fptr(); }
Question #11

• What is the coarse-grained CFI policy for the fptr() indirect call in function FC?

F1() { return; }
F2() { return; }
FA() { F1(); }
FB() { F2(); }
FC(fptr) { fptr(); }
Question #11

• If fptr() can be assigned to either F1 or F2, what is the most accurate CFI target set for the return from F1?

F1() { return; }
F2() { return; }
FA() { F1(); }
FB() { F2(); }
FC(fptr) { fptr(); }
Question #11

• Use nop’s with tags to enforce the CFI policy of (b) for returns from F1 and F2 and any indirect calls.

F1() { return; }
F2() { return; }
FA() { F1(); }
FB() { F2(); }
FC(fptr) { fptr(); }

• Use nop’s with tags to enforce the CFI policy of (b) for returns from F1 and F2 and any indirect calls.
Question #11

• Use `nop`'s with tags to enforce the CFI policy of (b) for returns from `F1` and `F2` and any indirect calls.

```c
F1() { return; if (**ESP == nop X) }  
F2() { return; }  
FA() { F1(); }  
FB() { F2(); }  
FC(fptr) { fptr(); }  
```
Question #11

• Use nop’s with tags to enforce the CFI policy of (b) for returns from F1 and F2 and any indirect calls.

F1() { return; if (**ESP == nop X) }
F2() { return; }
FA() { F1(); nop X; }
FB() { F2(); }
FC(fptr) { fptr(); nop X; }
• Use nop’s with tags to enforce the CFI policy of (b) for returns from F1 and F2 and any indirect calls.

```c
F1() { return; if (**ESP == nop X) }  
F2() { return; if (**ESP == nop X) }  
FA() { F1(); nop X; }  
FB() { F2(); }  
FC(fptr) { fptr(); nop X; }
```
Question #11

- Use nop’s with tags to enforce the CFI policy of (b) for returns from F1 and F2 and any indirect calls.

F1() { return; if (**ESP == nop X) }
F2() { return; if (**ESP == nop X) }
FA() { F1(); nop X; }
FB() { F2(); nop X; }
FC(fp.ptr) { fp.ptr(); nop X; }
Question #11

• What are the inter-function control flows that are allowed by the enforcement of (c) that should not be allowed? Why?

F1() { return; if (**ESP == nop X) }
F2() { return; if (**ESP == nop X) }
FA() { F1(); nop X; }
FB() { F2(); nop X; }
FC(fptr) { fptr(); nop X; }


Question #11

• How would you enforce the CFI policy of (b) to ensure no unauthorized control flows?

F1() { return; }
F2() { return; }
FA() { F1(); }
FB() { F2(); }
FC(fptr) { fptr(); }

• Use HyperSafe tables – F1 return – FA, FC; fptr call: F1, F2; F2 return: FB
Question #12

public boolean connect(NetworkAddr addr) {
    SecurityManager security = System.getSecurityManager();
    if (security != null) {
        ret = security.checkConnect();
        if (ret == false) {
            System.log(addr);
            return;
        }
    }
    if (isAddress(addr)) return connect0(addr);
}

• Suppose an applet invokes the connect code above provided by a Java system class. If System.log writes to a system log file and connect can connect to the specified IP address, which of these operations was possible in the Java 1.0 sandbox?
Question #12

Suppose an applet invokes the connect code above provided by a Java system class. If `System.log` writes to a system log file and `connect` can connect to the specified IP address, which of these operations was possible in the Java 1.1 sandbox if the applets are signed?

```java
public boolean connect(NetworkAddr addr) {
    SecurityManager security = System.getSecurityManager();
    if (security != null) {
        ret = security.checkConnect();
        if (ret == false) {
            System.log(addr);
            return;
        }
    }
    if (isAddress(addr)) return connect0(addr);
}
```

- Suppose an applet invokes the connect code above provided by a Java system class. If `System.log` writes to a system log file and `connect` can connect to the specified IP address, which of these operations was possible in the Java 1.1 sandbox if the applets are signed?

(c) In the Java 2.0 sandbox, suppose class C1 is authorized to the system log file and make connections with a particular IP address. If C1 is invoked by a class C2 that lacks such permissions, will the logging and connection be allowed?

(d) Specify how you would modify the code above to enable the Java system class to log unauthorized connect requests.

(e) Suppose that an adversary produced the bytecode for class C2 without using the Java compiler. Under what conditions would it be allowed to run to access its permissions in a Java 2.0 system?
Question #12

public boolean connect(NetworkAddr addr) {
    SecurityManager security = System.getSecurityManager();
    if (security != null) {
        ret = security.checkConnect();
        if (ret == false) {
            System.log(addr);
            return;
        }
    }
    if (isAddress(addr)) return connect0(addr);
}

• In the Java 2.0 sandbox, suppose class C1 is authorized to the system log file and make connections with a particular IP address. If C1 is invoked by a class C2 that lacks such permissions, will the logging and connection be allowed?
public boolean connect(NetworkAddr addr) 
{  
    SecurityManager security = System.getSecurityManager();  
    if (security != null) 
    {  
        ret = security.checkConnect();  
        if (ret == false) 
        {  
            System.log(addr);  
            return;  
        }  
    }  
    if (isAddress(addr)) return connect0(addr);  
}

- Specify how you would modify the code above to enable the Java system class to log unauthorized connect requests.
Question #12

```java
public boolean connect(NetworkAddr addr) {
    SecurityManager security = System.getSecurityManager();
    if (security != null) {
        ret = security.checkConnect();
        if (ret == false) {
            System.log(addr);
            return;
        }
    }

    if (isAddress(addr)) return connect0(addr);
}
```

- Specify how you would modify the code above to enable the Java system class to log unauthorized connect requests.
- DoPrivileged – where?
Question #12

public boolean connect(NetworkAddr addr) {
    SecurityManager security = System.getSecurityManager();
    if (security != null) {
        ret = security.checkConnect();
        if (ret == false) {
            System.log(addr);
            return;
        }
    }
    if (isAddress(addr)) return connect0(addr);
}

• Suppose that an adversary produced the bytecode for class C2 without using the Java compiler. Under what conditions would it be allowed to run to access its permissions in a Java 2.0 system?
Question #13

extern int send(int fd, char {public} *msg);

int
{public} SendSecretData(Key {secret} key, Name {public}secretfile, Socket {public} sock)
{
    String {secret} data;
    String encrypted_data;
    int error;

    error = read(secretfile, data);
    if (error) return error

    error = encrypt(key, data, encrypted_data);
    if (error) return error;

    send(sock, encrypted_data);

    return error;
}

• What label is inferred for encrypted_data at the send system call if encrypt causes a flow from data to encrypted data?
extern int send(int fd, char {public} *msg);

int {public} SendSecretData(Key {secret} key, Name {public}secretfile, Socket {public} sock)
{
    String {secret} data;
    String encrypted_data;
    int error;

    error = read(secretfile, data);
    if (error) return error

    error = encrypt(key, data, encrypted_data);
    if (error) return error;

    send(sock, encrypted_data);

    return error;
}

• What label should be inferred for the return value from read? Why?
Question #13

extern int send(int fd, char {public} *msg);

int {public} SendSecretData(Key {secret} key, Name {public}secretfile, Socket {public} sock)
{
    String {secret} data;
    String encrypted_data;
    int error;

    error = read(secretfile, data);
    if (error) return error

    error = encrypt(key, data, encrypted_data);
    if (error) return error;

    send(sock, encrypted_data);

    return error;
}

• Assume encrypt returns a secret value. Identify all the locations that may cause security-type errors.
Question #13

extern int send(int fd, char {public} *msg);

int {public} SendSecretData(Key {secret} key, Name {public}secretfile, Socket {public} sock)
{
    String {secret} data;
    String encrypted_data;
    int error;

    error = read(secretfile, data);
    if (error) return error

    error = encrypt(key, data, encrypted_data);
    if (error) return error;

    send(sock, encrypted_data);

    return error;
}

• Assume encrypt returns a secret value. Identify all the locations that may cause security-type errors.
  ‣ Last 3 lines all will report type errors – why?
**Question #13**

```c
extern int send(int fd, char *msg);

int {public} SendSecretData(Key {secret} key, Name {public}secretfile, Socket {public} sock)
{
    String {secret} data;
    String encrypted_data;
    int error;

    error = read(secretfile, data);
    if (error) return error

    error = encrypt(key, data, encrypted_data);
    if (error) return error;

    send(sock, encrypted_data);

    return error;
}
```

- **How would you fix these security-type errors? Be specific.**
Question #13

extern int send(int fd, char {public} *msg);

int {public} SendSecretData(Key {secret} key, Name {public}secretfile, Socket {public} sock)
{
    String {secret} data;
    String encrypted_data;
    int error;

    error = read(secretfile, data);
    if (error) return error

    error = encrypt(key, data, encrypted_data);
    if (error) return error;

    send(sock, encrypted_data);

    return error;
}

• How would you fix these security-type errors? Be specific.
  ‣ Declassify encrypted_data and error – create new vars
  ‣ Should initialize ‘error’
Question #13

- How would you fix these security-type errors? Be specific.
  - Declassify encrypted_data and error – create new vars

- Is this program secure?
  
  ```
  do {  // password hash is secret
      if ( password_hash == hash(password) )
          return {public} authenticated;
      else {
          *try = {public} (*try+1);
          return {public} !authenticated;
      }
  }
  while ( try < limit );
  ```
Question #13

extern int send(int fd, char {public} *msg);

int {public} SendSecretData(Key {secret} key, Name {public}secretfile, Socket {public} sock)
{
    String {secret} data;
    String encrypted_data;
    int error;

    error = read(secretfile, data);
    if (error) return error

    error = encrypt(key, data, encrypted_data);
    if (error) return error;

    send(sock, encrypted_data);

    return error;
}

• Do we need to label the variable encrypted_data to secret statically to get the same effect (as declassification)?
extern int send(int fd, char {public} *msg);

int {public} SendSecretData(Key {secret} key, Name {public}secretfile, Socket {public} sock)
{
  String {secret} data;
  String encrypted_data;
  int error;

  error = read(secretfile, data);
  if (error) return error;

  error = encrypt(key, data, encrypted_data);
  if (error) return error;

  send(sock, encrypted_data);

  return error;
}

• Do we need to label the variable encrypted data to secret statically to get the same effect (as declassification)?

  ‣ Or public?