SAMSUNG

CATERPILLAR: Iterative Concolic Execution for seed generation

Laurent Simon, Shuying Liang, Amir Rahmati, Mike Grace
{l.simon, s.liang, amir.rahmati, m1.grace}@samsung.com
KNOX Security Team, Mountain View, CA

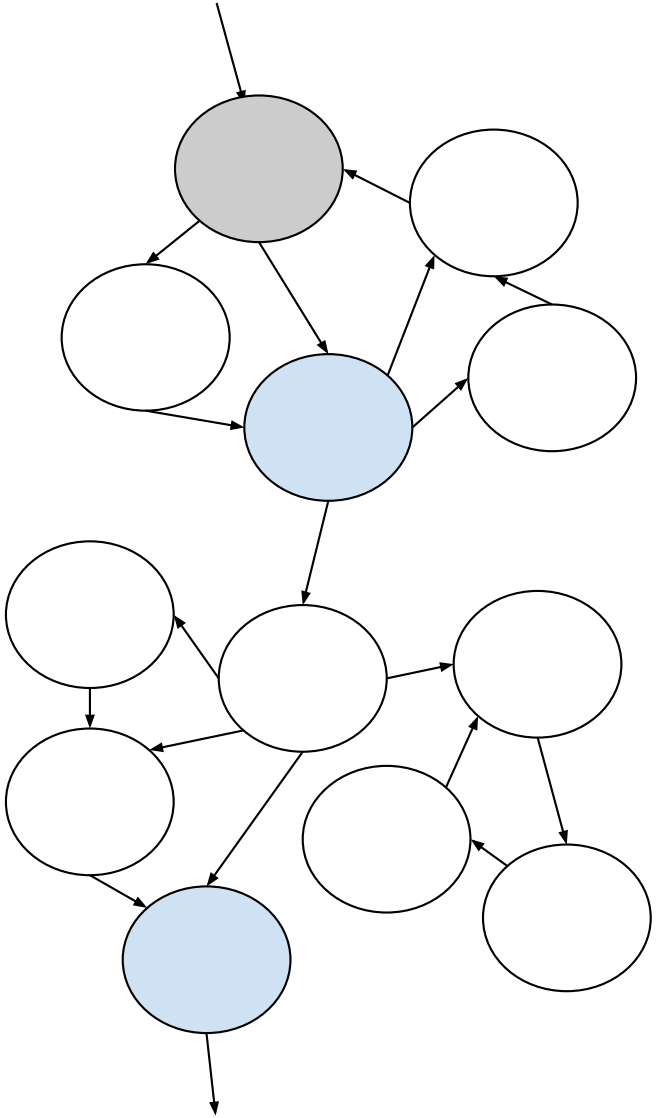
Agenda

1. **Background**
2. Original Idea and Evolution
3. Evaluation

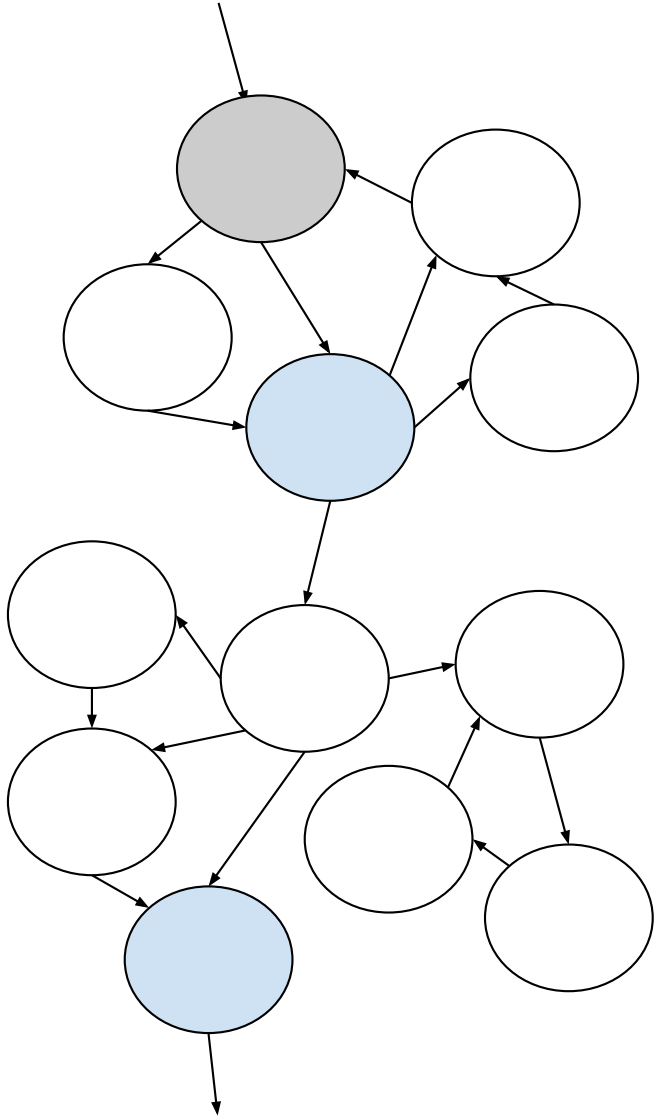
Stateful programs

- Daemon

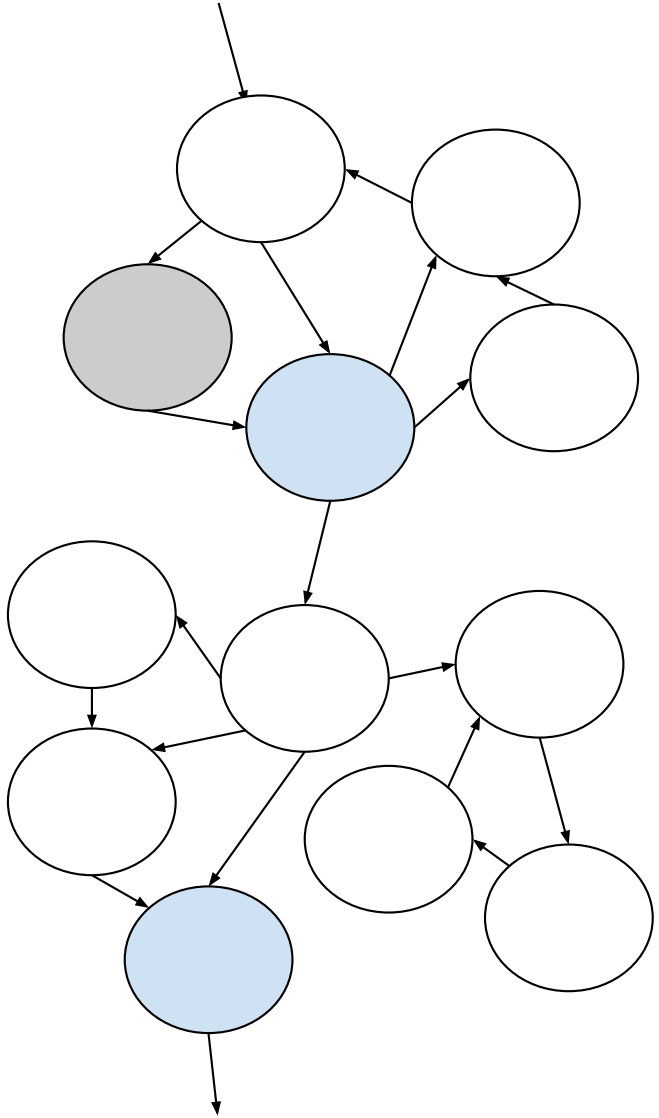
- State changes as the result of processing commands



Request 1

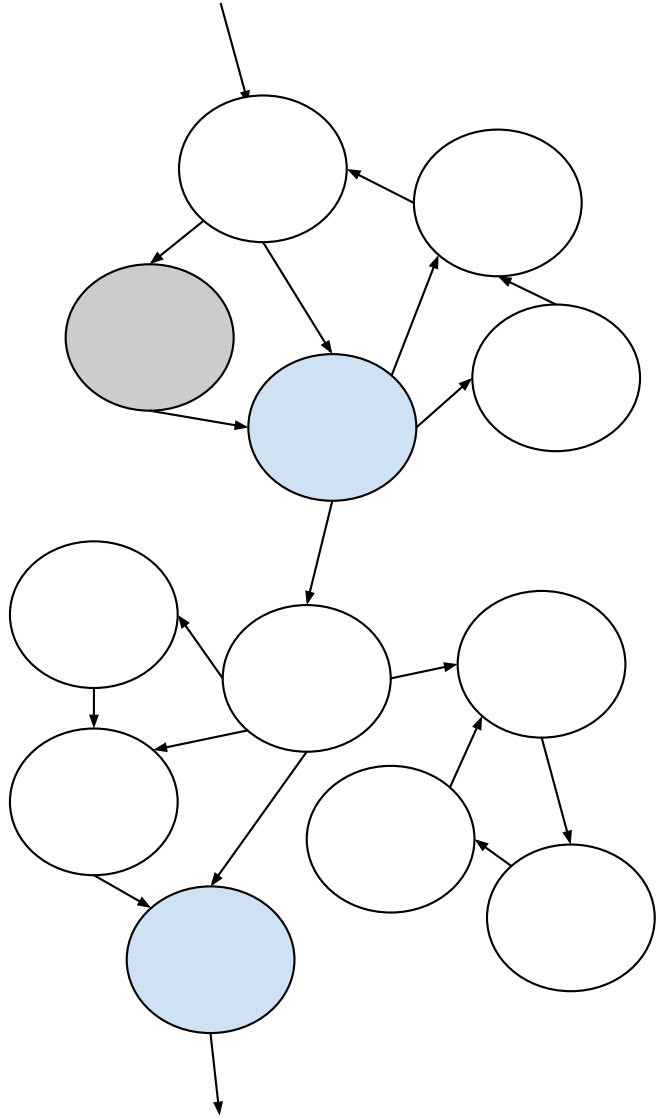


Request 1



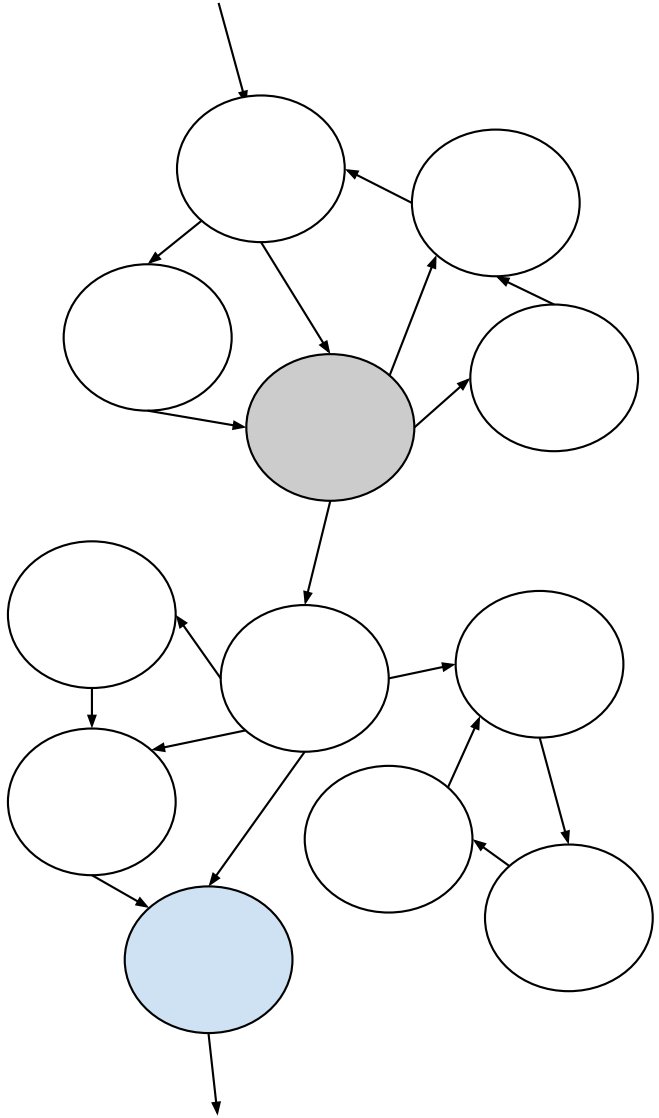
Request 1

Request 2



Request 1

Request 2



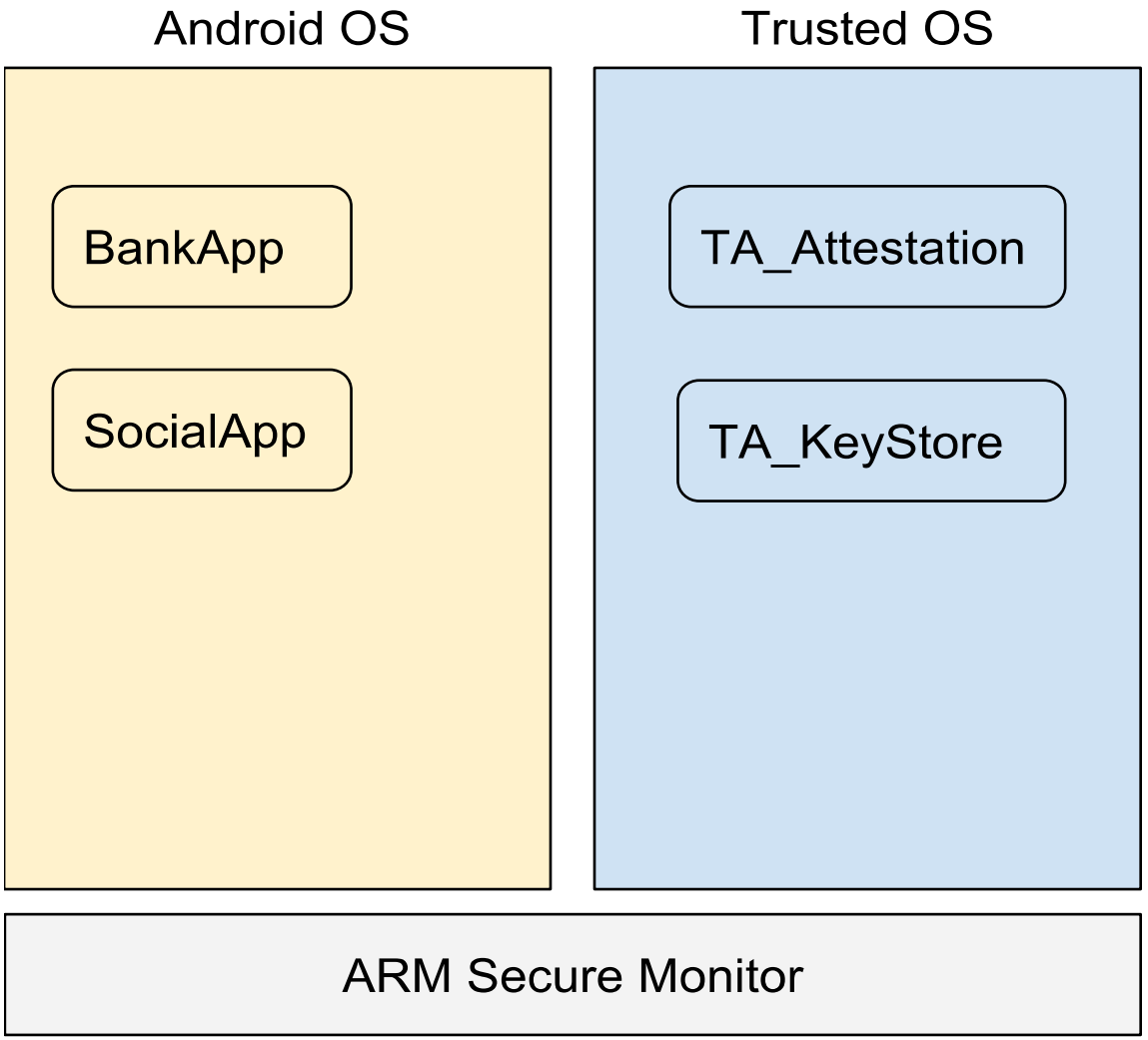
Current Approaches

- KLEE path explosion (plateau after a few hours)
- AFL large input mutation (plateau after 1 hour)
- Guided fuzzing (KLEE w/ AFL: SEFuzzer), often requires an initial seed

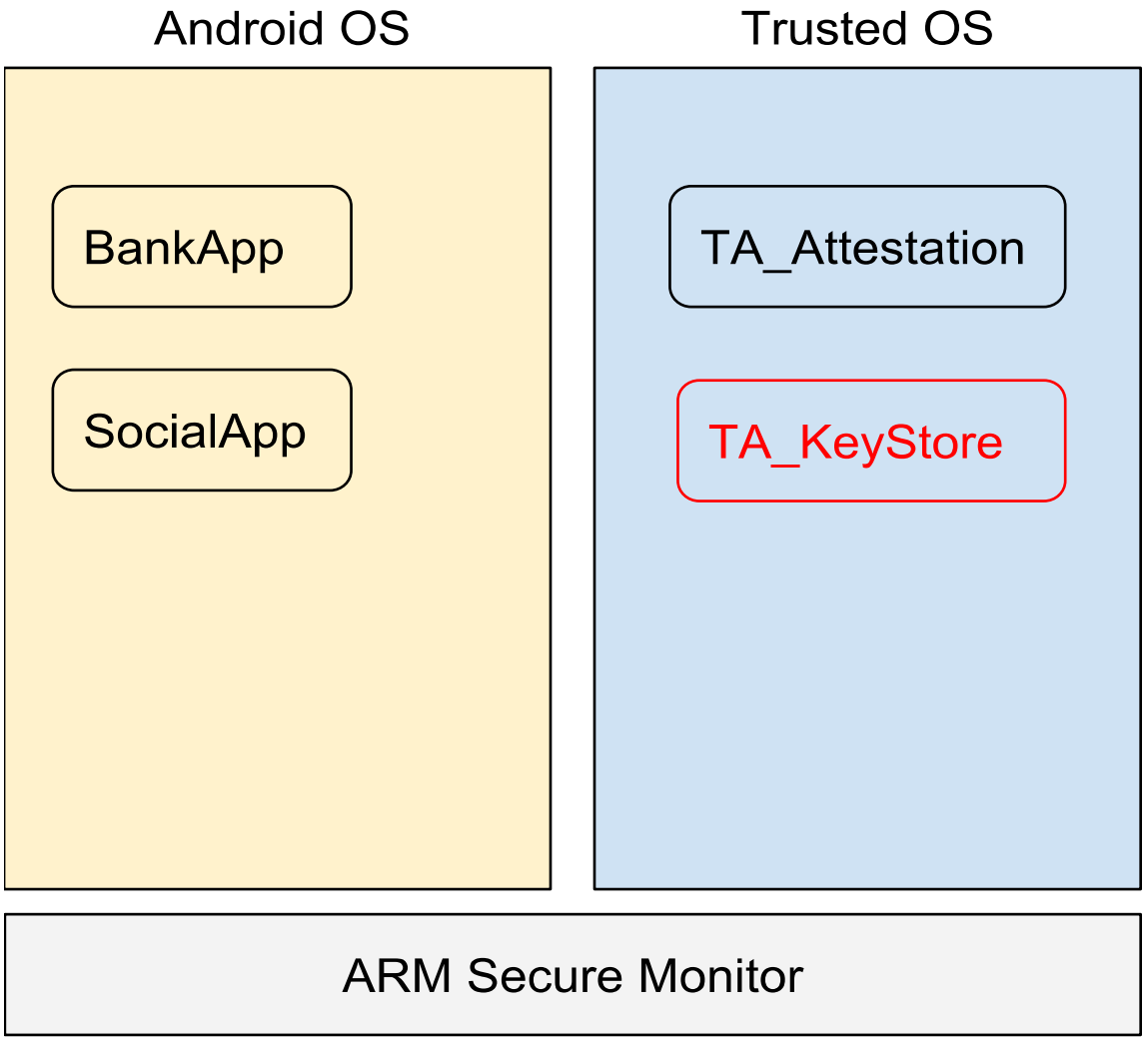
Our goals: CATERPILLAR

- Drive down the execution in a efficient way / navigate state machine
- Generate quality inputs for a fuzzer
- Bonus: find bugs during seed generation too

Stateful programs at Samsung



Stateful programs at Samsung



```
int main() {

        while(1) {

                request_t req = {0};
                getClientRequest( &req );

                switch(req.command) {

                        case CMD_OPEN:
                                OpenSession(req.buf, req.len);
                                break;

                        case CMD_INIT:
                                Init(req.buf, req.len);
                                break;

                        case CMD_WHATEVER:
                                Whatever(req.buf, req.len);
                                break;


                }
        }
}
```

```
int main() {
    while(1) {
        request_t req = {0};
        getClientRequest( &req );

        switch(req.command) {

            case CMD_OPEN:
                OpenSession(req.buf, req.len);
                break;

            case CMD_INIT:
                Init(req.buf, req.len);
                break;

            case CMD_WHATEVER:
                Whatever(req.buf, req.len);
                break;


        }
    }
}
```

```
int main() {

    while(1) {

        request_t req = {0};
        getClientRequest( &req );

        switch(req.command) {

            case CMD_OPEN:
                OpenSession(req.buf, req.len);
                break;

            case CMD_INIT:
                Init(req.buf, req.len);
                break;

            case CMD_WHATEVER:
                Whatever(req.buf, req.len);
                break;


        }
    }
}
```

```
int main() {

        while(1) {

                request_t req = {0};
                getClientRequest( &req );

                switch(req.command) {

                        case CMD_OPEN:
                                OpenSession(req.buf, req.len);
                                break;

                        case CMD_INIT:
                                Init(req.buf, req.len);
                                break;

                        case CMD_WHATEVER:
                                Whatever(req.buf, req.len);
                                break;


                }
        }
}
```

```
int main() {

        while(1) {

                request_t req = {0};
                getClientRequest( &req );

                switch(req.command) {

                        case CMD_OPEN:
                                OpenSession(req.buf, req.len);
                                break;

                        case CMD_INIT:
                                Init(req.buf, req.len);
                                break;

                        case CMD_WHATEVER:
                                Whatever(req.buf, req.len);
                                break;


                }
        }
}
```

```
int main() {

        request_t req = {0};

        // … read client data

        Init(req.buf, req.len);

        OpenSession(req.buf, req.len);

        Decrypt(req.buf, req.len);

}
```

Agenda

1. Background
2. **Original Idea and Evolution**
3. Evaluation

```
int Init(u8 * buf, size_t len) {

    Session_t * session = 0;

    /*
    some complicated logic here, eg
    */

    if ( validateAppData(buf, len) != 0 )
        return ERROR_DATA;

    if ( createNewSession(&session) != 0 )
        return ERROR_CREATE;

    if ( generateSessionId(&session) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_CLOSED;

    // add the session to list of sessions
    addSessionToList(session);

    return OK;
}
```

```
int OpenSession(int sessionId,
                u8 * buf, size_t len) {

    Session_t * session = 0;

    // find session
    if ( findSession(&session, &sessionId) != 0 )
        return ERROR_DATA;

    // check session state
    if ( session->state != SESSION_CLOSED )
        return ERROR_STATE;

    if ( DoOpenSession(session, buf, len) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_OPENED;

    return OK;
}
```

```
int Init(u8 * buf, size_t len) {

    Session_t * session = 0;

    /*
    some complicated logic here, eg
    */

    if ( validateAppData(buf, len) != 0 )
        return ERROR_DATA;

    if ( createNewSession(&session) != 0 )
        return ERROR_CREATE;

    if ( generateSessionId(&session) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_CLOSED;

    // add the session to list of sessions
    addSessionToList(session);

    return OK;
}
```

```
int OpenSession(int sessionId,
                u8 * buf, size_t len) {

    Session_t * session = 0;

    // find session
    if ( findSession(&session, &sessionId) != 0 )
        return ERROR_DATA;

    // check session state
    if ( session->state != SESSION_CLOSED )
        return ERROR_STATE;

    if ( DoOpenSession(session, buf, len) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_OPENED;

    return OK;
}
```

```
int Init(u8 * buf, size_t len) {

    Session_t * session = 0;

    /*
    some complicated logic here, eg
    */

    if ( validateAppData(buf, len) != 0 )
        return ERROR_DATA;

    if ( createNewSession(&session) != 0 )
        return ERROR_CREATE;

    if ( generateSessionId(&session) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_CLOSED;

    // add the session to list of sessions
    addSessionToList(session);

    return OK;
}
```

```
int OpenSession(int sessionId,
                u8 * buf, size_t len) {

    Session_t * session = 0;

    // find session
    if ( findSession(&session, &sessionId) != 0 )
        return ERROR_DATA;

    // check session state
    if ( session->state != SESSION_CLOSED )
        return ERROR_STATE;

    if ( DoOpenSession(session, buf, len) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_OPENED;

    return OK;
}
```

```
int Init(u8 * buf, size_t len) {

    Session_t * session = 0;

    /*
    some complicated logic here, eg
    */

    if ( validateAppData(buf, len) != 0 )
        return ERROR_DATA;

    if ( createNewSession(&session) != 0 )
        return ERROR_CREATE;

    if ( generateSessionId(&session) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_CLOSED;

    // add the session to list of sessions
    addSessionToList(session);

    return OK;
}
```

```
int OpenSession(int sessionId,
                u8 * buf, size_t len) {

    Session_t * session = 0;

    // find session
    if ( findSession(&session, &sessionId) != 0 )
        return ERROR_DATA;

    // check session state
    if ( session->state != SESSION_CLOSED )
        return ERROR_STATE;

    if ( DoOpenSession(session, buf, len) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_OPENED;

    return OK;
}
```

```
int Init(u8 * buf, size_t len) {

    Session_t * session = 0;

    /*
    some complicated logic here, eg
    */

    if ( validateAppData(buf, len) != 0 )
        return ERROR_DATA;

    if ( createNewSession(&session) != 0 )
        return ERROR_CREATE;

    if ( generateSessionId(&session) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_CLOSED;

    // add the session to list of sessions
    addSessionToList(session);

    return OK;
}
```

```
int OpenSession(int sessionId,
                u8 * buf, size_t len) {

    Session_t * session = 0;

    // find session
    if ( findSession(&session, &sessionId) != 0 )
        return ERROR_DATA;

    // check session state
    if ( session->state != SESSION_CLOSED )
        return ERROR_STATE;

    if ( DoOpenSession(session, buf, len) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_OPENED;

    return OK;
}
```

```
int Init(u8 * buf, size_t len) {

    Session_t * session = 0;

    /*
    some complicated logic here, eg
    */

    if ( validateAppData(buf, len) != 0 )
        return ERROR_DATA;

    if ( createNewSession(&session) != 0 )
        return ERROR_CREATE;

    if ( generateSessionId(&session) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_CLOSED;

    // add the session to list of sessions
    addSessionToList(session);

    return OK;
}
```

```
int OpenSession(int sessionId,
                u8 * buf, size_t len) {

    Session_t * session = 0;

    // find session
    if ( findSession(&session, &sessionId) != 0 )
        return ERROR_DATA;

    // check session state
    if ( session->state != SESSION_CLOSED )
        return ERROR_STATE;

    if ( DoOpenSession(session, buf, len) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_OPENED;

    return OK;
}
```

```
int Init(u8 * buf, size_t len) {

    Session_t * session = 0;

    /*
    some complicated logic here, eg
    */

    if ( validateAppData(buf, len) != 0 )
        return ERROR_DATA;

    if ( createNewSession(&session) != 0 )
        return ERROR_CREATE;

    if ( generateSessionId(&session) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_CLOSED;

    // add the session to list of sessions
    addSessionToList(session);

    return OK;
}
```

```
int OpenSession(int sessionId,
                u8 * buf, size_t len) {

    Session_t * session = 0;

    // find session
    if ( findSession(&session, &sessionId) != 0 )
        return ERROR_DATA;

    // check session state
    if ( session->state != SESSION_CLOSED )
        return ERROR_STATE;

    if ( DoOpenSession(session, buf, len) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_OPENED;

    return OK;
}
```

```
int Init(u8 * buf, size_t len) {

    Session_t * session = 0;

    /*
    some complicated logic here, eg
    */

    if ( validateAppData(buf, len) != 0 )
        return ERROR_DATA;

    if ( createNewSession(&session) != 0 )
        return ERROR_CREATE;

    if ( generateSessionId(&session) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_CLOSED;

    // add the session to list of sessions
    addSessionToList(session);

    return OK;
}
```

```
int OpenSession(int sessionId,
                u8 * buf, size_t len) {

    Session_t * session = 0;

    // find session
    if ( findSession(&session, &sessionId) != 0 )
        return ERROR_DATA;

    // check session state
    if ( session->state != SESSION_CLOSED )
        return ERROR_STATE;

    if ( DoOpenSession(session, buf, len) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_OPENED;

    return OK;
}
```

```
int Init(u8 * buf, size_t len) {

    Session_t * session = 0;

    /*
    some complicated logic here, eg
    */

    if ( validateAppData(buf, len) != 0 )
        return ERROR_DATA;

    if ( createNewSession(&session) != 0 )
        return ERROR_CREATE;

    if ( generateSessionId(&session) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_CLOSED;

    // add the session to list of sessions
    addSessionToList(session);

    return OK;
}
```

```
int OpenSession(int sessionId,
                u8 * buf, size_t len) {

    Session_t * session = 0;

    // find session
    if ( findSession(&session, &sessionId) != 0 )
        return ERROR_DATA;

    // check session state
    if ( session->state != SESSION_CLOSED )
        return ERROR_STATE;

    if ( DoOpenSession(session, buf, len) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_OPENED;

    return OK;
}
```

```
int Init(u8 * buf, size_t len) {

    Session_t * session = 0;

    /*
    some complicated logic here, eg
    */

    if ( validateAppData(buf, len) != 0 )
        return ERROR_DATA;

    if ( createNewSession(&session) != 0 )
        return ERROR_CREATE;

    if ( generateSessionId(&session) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_CLOSED;

    // add the session to list of sessions
    addSessionToList(session);

    return OK;
}
```

```
int OpenSession(int sessionId,
                u8 * buf, size_t len) {

    Session_t * session = 0;

    // find session
    if ( findSession(&session, &sessionId) != 0 )
        return ERROR_DATA;

    // check session state
    if ( session->state != SESSION_CLOSED )
        return ERROR_STATE;

    if ( DoOpenSession(session, buf, len) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_OPENED;

    return OK;
}
```

```
int Init(u8 * buf, size_t len) {

    Session_t * session = 0;

    /*
    some complicated logic here, eg
    */

    if ( validateAppData(buf, len) != 0 )
        return ERROR_DATA;

    if ( createNewSession(&session) != 0 )
        return ERROR_CREATE;

    if ( generateSessionId(&session) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_CLOSED;

    // add the session to list of sessions
    addSessionToList(session);

    return OK;
}
```

```
int OpenSession(int sessionId,
                u8 * buf, size_t len) {

    Session_t * session = 0;

    // find session
    if ( findSession(&session, &sessionId) != 0 )
        return ERROR_DATA;

    // check session state
    if ( session->state != SESSION_CLOSED )
        return ERROR_STATE;

    if ( DoOpenSession(session, buf, len) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_OPENED;

    return OK;
}
```

```
int Init(u8 * buf, size_t len) {

    Session_t * session = 0;

    /*
    some complicated logic here, eg
    */

    if ( validateAppData(buf, len) != 0 )
            return ERROR_DATA;

    if ( createNewSession(&session) != 0 )
            return ERROR_CREATE;

    if ( generateSessionId(&session) != 0 )
            return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_CLOSED;

    // add the session to list of sessions
    addSessionToList(session);

    return OK;
}
```

```
int OpenSession(int sessionId,
                u8 * buf, size_t len) {

    Session_t * session = 0;

    // find session
    if ( findSession(&session, &sessionId) != 0 )
            return ERROR_DATA;

    // check session state
    if ( session->state != SESSION_CLOSED )
            return ERROR_STATE;

    if ( DoOpenSession(session, buf, len) != 0 )
            return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_OPENED;

    return OK;
}
```

```
int Init(u8 * buf, size_t len) {

    Session_t * session = 0;

    /*
    some complicated logic here, eg
    */

    if ( validateAppData(buf, len) != 0 )
            return ERROR_DATA;

    if ( createNewSession(&session) != 0 )
            return ERROR_CREATE;

    if ( generateSessionId(&session) != 0 )
            return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_CLOSED;

    // add the session to list of sessions
    addSessionToList(session);

    return OK;
}
```

```
int OpenSession(int sessionId,
                u8 * buf, size_t len) {

    Session_t * session = 0;

    // find session
    if ( findSession(&session, &sessionId) != 0 )
            return ERROR_DATA;

    // check session state
    if ( session->state != SESSION_CLOSED )
            return ERROR_STATE;

    if ( DoOpenSession(session, buf, len) != 0 )
            return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_OPENED;

    return OK;
}
```

```
int Init(u8 * buf, size_t len) {

    Session_t * session = 0;

    /*
    some complicated logic here, eg
    */

    if ( validateAppData(buf, len) != 0 )
            return ERROR_DATA;

    if ( createNewSession(&session) != 0 )
            return ERROR_CREATE;

    if ( generateSessionId(&session) != 0 )
            return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_CLOSED;

    // add the session to list of sessions
    addSessionToList(session);

    return OK;
}
```

```
int OpenSession(int sessionId,
                u8 * buf, size_t len) {

    Session_t * session = 0;

    // find session
    if ( findSession(&session, &sessionId) != 0 )
            return ERROR_DATA;

    // check session state
    if ( session->state != SESSION_CLOSED )
            return ERROR_STATE;

    if ( DoOpenSession(session, buf, len) != 0 )
            return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_OPENED;

    return OK;
}
```

Key ideas

1) Execute Init() symbolically in KLEE

Key ideas

1) Execute Init() symbolically in KLEE

2) Filter out “unsuccessful” states, i.e., that do not return OK

Key ideas

1) Execute Init() symbolically in KLEE
2) Filter out “unsuccessful” states, i.e., that do not return OK
3) Concretize successful states as .ktest files and stop KLEE

Key ideas

1) Execute Init() symbolically in KLEE
2) Filter out “unsuccessful” states, i.e., that do not return OK
3) Concretize successful states as .ktest files and stop KLEE
    i. Randomly pick a .ktest file and replay it concretely in new KLEE instance

Key ideas

1) Execute Init() symbolically in KLEE
2) Filter out “unsuccessful” states, i.e., that do not return OK
3) Concretize successful states as .ktest files and stop KLEE
    i. Randomly pick a .ktest file and replay it concretely in new KLEE instance
    ii. Switch to symbolic exploration in OpenSession()

SYMBOLICALLY

```
int Init(u8 * buf, size_t len) {

    Session_t * session = 0;

    /*
    some complicated logic here, eg
    */

    if ( validateAppData(buf, len) != 0 )
        return ERROR_DATA;

    if ( createNewSession(&session) != 0 )
        return ERROR_CREATE;

    if ( generateSessionId(&session) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_CLOSED;

    // add the session to list of sessions
    addSessionToList(session);

    return OK;
}
```

```
int OpenSession(int sessionId,
                u8 * buf, size_t len) {

    Session_t * session = 0;

    // find session
    if ( findSession(&session, &sessionId) != 0 )
        return ERROR_DATA;

    // check session state
    if ( session->state != SESSION_CLOSED )
        return ERROR_STATE;

    if ( DoOpenSession(session, buf, len) != 0 )
        return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_OPENED;

    return OK;
}
```

```
int Init(u8 * buf, size_t len) {

    Session_t * session = 0;

    /*
    some complicated logic here, eg
    */

    if ( validateAppData(buf, len) != 0 )
            return ERROR_DATA;

    if ( createNewSession(&session) != 0 )
            return ERROR_CREATE;

    if ( generateSessionId(&session) != 0 )
            return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_CLOSED;

    // add the session to list of sessions
    addSessionToList(session);

    return OK;
}
```

CONCRETELY

```
int OpenSession(int sessionId,
                u8 * buf, size_t len) {

    Session_t * session = 0;

    // find session
    if ( findSession(&session, &sessionId) != 0 )
            return ERROR_DATA;

    // check session state
    if ( session->state != SESSION_CLOSED )
            return ERROR_STATE;

    if ( DoOpenSession(session, buf, len) != 0 )
            return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_OPENED;

    return OK;
}
```

CONCRETELY

```
int Init(u8 * buf, size_t len) {

    Session_t * session = 0;

    /*
    some complicated logic here, eg
    */

    if ( validateAppData(buf, len) != 0 )
            return ERROR_DATA;

    if ( createNewSession(&session) != 0 )
            return ERROR_CREATE;

    if ( generateSessionId(&session) != 0 )
            return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_CLOSED;

    // add the session to list of sessions
    addSessionToList(session);

    return OK;
}
```

SYMBOLICALLY

```
int OpenSession(int sessionId,
                u8 * buf, size_t len) {

    Session_t * session = 0;

    // find session
    if ( findSession(&session, &sessionId) != 0 )
            return ERROR_DATA;

    // check session state
    if ( session->state != SESSION_CLOSED )
            return ERROR_STATE;

    if ( DoOpenSession(session, buf, len) != 0 )
            return ERROR_SESSION;

    // and so on and so forth
    ...

    session->state = SESSION_OPENED;

    return OK;
}
```

Advantages

✓ Reduce path explosion

✓ Drive down the execution path efficiently

✗ Lose in soundness (i.e., increase in false negative) which we make up for with a fuzzer

Every program is “stateful”

- Stateful programs not predominant

- Other programs of interest:
  - e.g. X509 ASN1 certificate parsing (https, TLS)

```
int x509_crt_parse_der_core(
        u8 * buf, size_t len)
{
        ...

        if( ( ret = mbedtls_asn1_get_tag( ... ) ) != 0 )
        {
                mbedtls_x509_crt_free( crt );
                return( SOME_ERROR );
        }

        ...

        if( ( ret = mbedtls_asn1_get_tag( ... ) ) != 0 )
        {
                mbedtls_x509_crt_free( crt );
                return( SOME_ERROR );
        }

        ...
        if( ( ret = mbedtls_x509_get_sig_alg( ... ) ) != 0 )
        {
                mbedtls_x509_crt_free( crt );
                return( ret );
        }

        ...

        return OK;
}
```

```
int x509_crt_parse_der_core(
        u8 * buf, size_t len)
{
        ...

        if( ( ret = mbedtls_asn1_get_tag( ... ) ) != 0 )
        {
                mbedtls_x509_crt_free( crt );
                return( SOME_ERROR );
        }

        ...
        if( ( ret = mbedtls_asn1_get_tag( ... ) ) != 0 )
        {
                mbedtls_x509_crt_free( crt );
                return( SOME_ERROR);
        }


        if( ( ret = mbedtls_x509_get_sig_alg( ... ) ) != 0 )
        {
                mbedtls_x509_crt_free( crt );
                return( ret );
        }

        ...

        return OK;
}
```

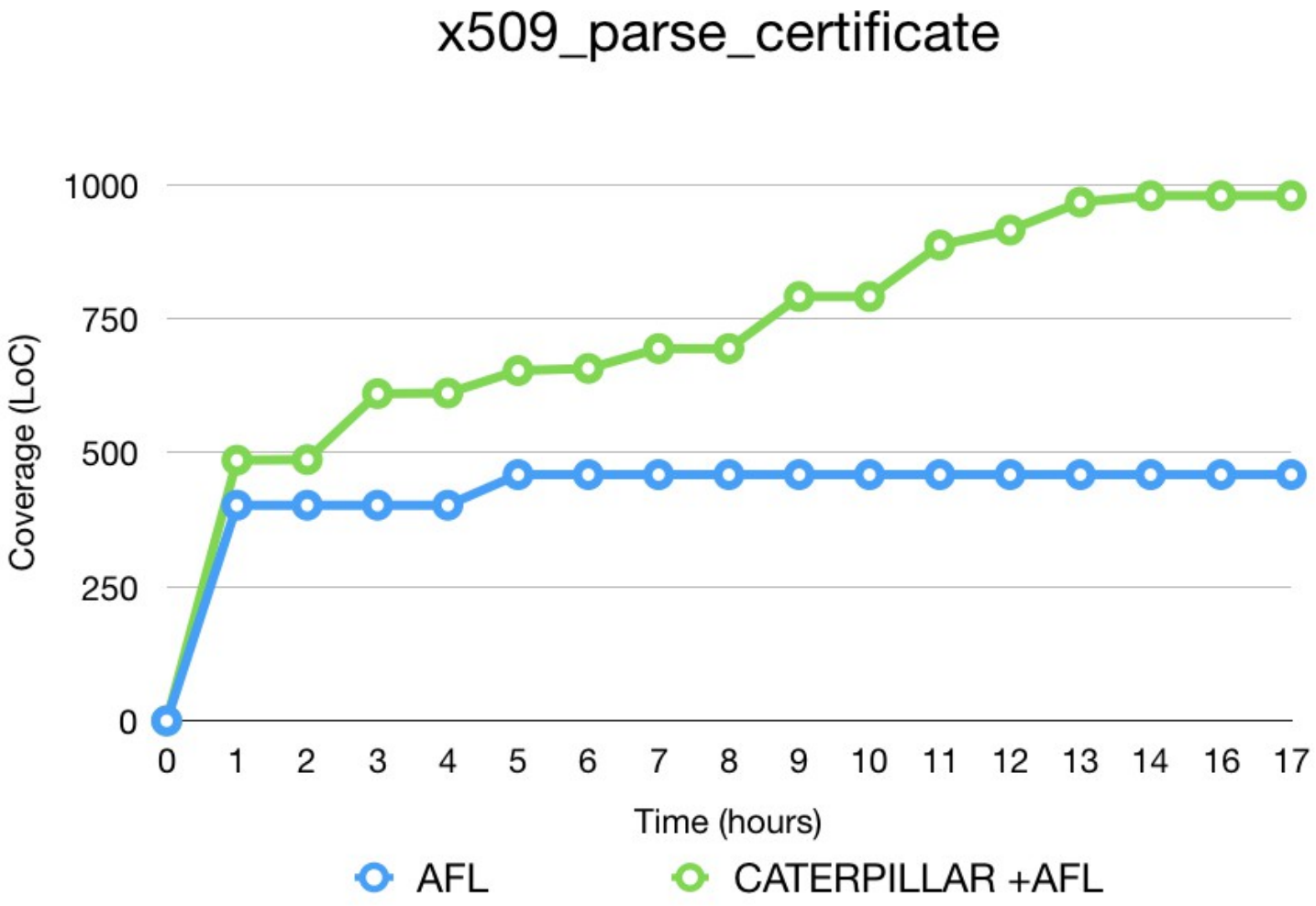
```
int main() {

        ...

        if (Init(req.buf, req.len) != OK)
                return ERROR;




        if (OpenSession(req.buf, req.len) != OK)
                return ERROR;




        if (Decrypt(req.buf, req.len)  != OK)
                return ERROR;




        return OK;
}
```

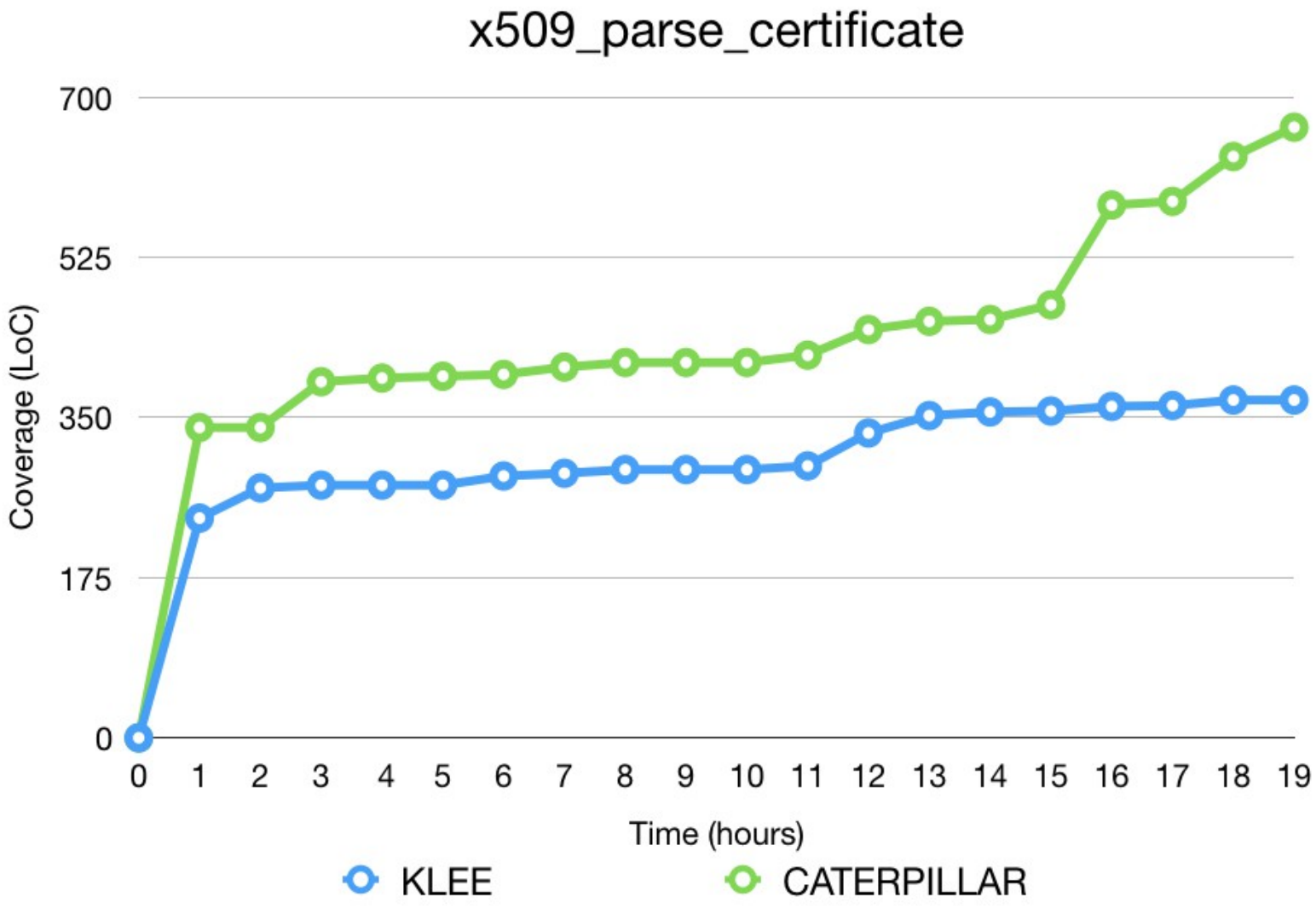
Agenda

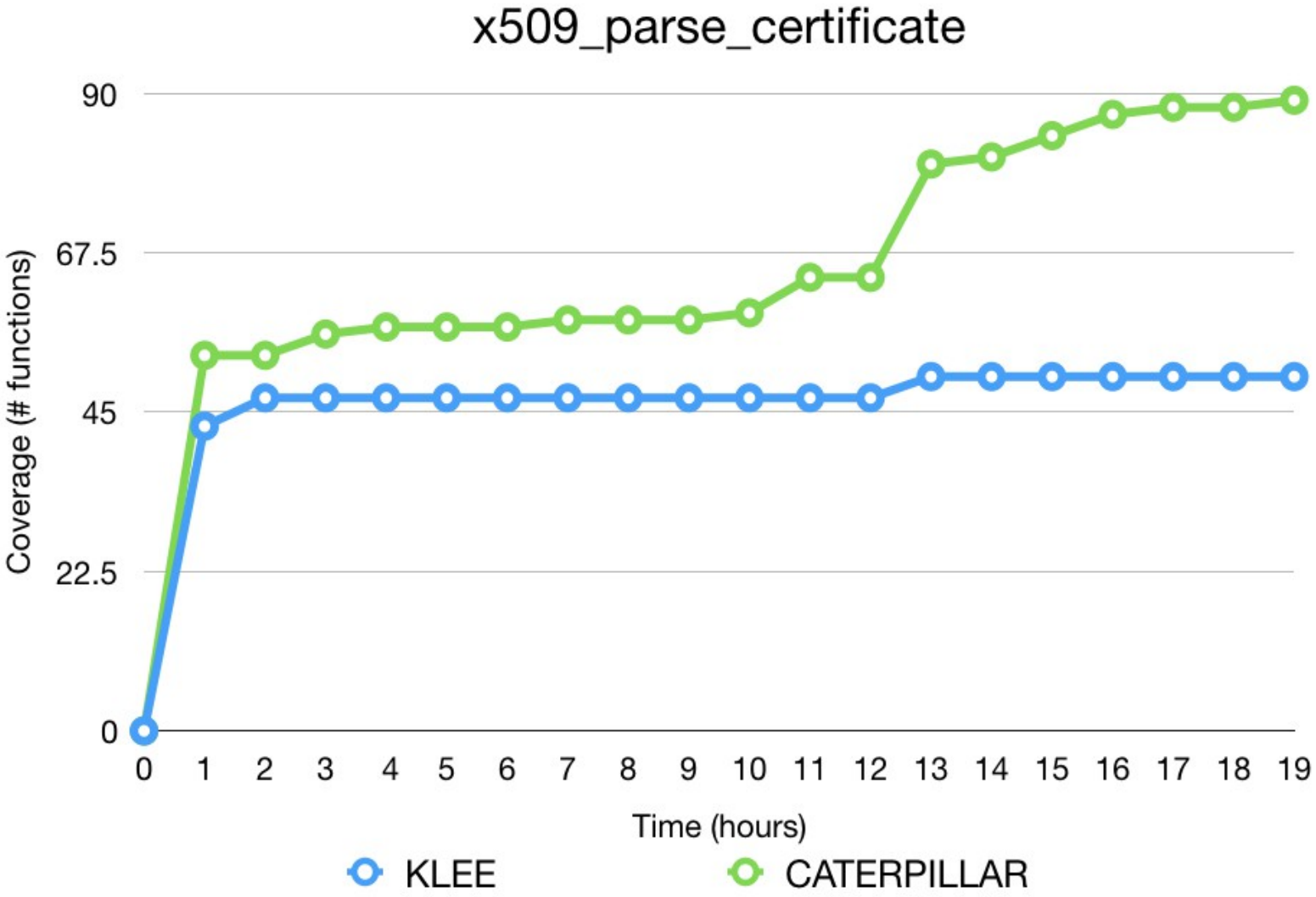
1. Background
2. Original Idea and Evolution
3. **Evaluation**

Setup

- 64-bit Ubuntu 16GB Intel Xeon E5-2650 @ 2GHz (16 cores)
- Analysis
  - KLEE alone
  - AFL alone
  - CATERPILLAR + AFL
    - CATERPILLAR and AFL run in parallel
    - CATERPILLAR (KLEE) seeds → AFL seeds


x509_parse_certificate
1000
750
500
250
0
Coverage (LoC)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 16 17
Time (hours)
AFL
CATERPILLAR +AFL


x509_parse_certificate
700
525
350
175
0
Coverage (LoC)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Time (hours)
KLEE
CATERPILLAR


x509_parse_certificate
Coverage (# functions)
90
67.5
45
22.5
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Time (hours)
KLEE
CATERPILLAR

Future Research Questions/Directions

- Automate function selection for switching ICE between symbolic and concrete execution

Future Research Questions/Directions

- Automate function selection for switching ICE between symbolic and concrete execution
- Permutation of calls

Future Research Questions/Directions

- Automate function selection for switching ICE between symbolic and concrete execution
- Permutation of calls
- Adaptive selection execution between static and dynamic analysis
  - Block level coordination
  - Function level coordination

Future Research Questions/Directions

- Automate function selection for switching ICE between symbolic and concrete execution
- Permutation of calls
- Adaptive selection execution between static and dynamic analysis
  - Block level coordination
  - Function level coordination
- Semi-automated testing: a gap?

SAMSUNG

Questions?

We offer internships :-)

Laurent Simon, Shuying Liang, Amir Rahmati, Mike Grace
{l.simon, s.liang, amir.rahmati, m1.grace}@samsung.com
KNOX Security Team, Mountain View, CA