D-TIME
Distributed Threadless Independent Malware Execution for Runtime Obfuscation

13th USENIX Workshop on Offensive Technologies

Jithin Pavithran
Milan Patnaik
Chester Rebeiro
Advanced Malware Detection Techniques

Behavioral Detection  Run time behavior

connect(socket, ai_addr ...)

GetAsyncKeyState(char)

send(socket, ...)

13 Aug 2019
D-TIME
Distributed Threadless Independent Malware Execution
D-TIME

Chunks (set of instructions)

Split
D-TIME

<table>
<thead>
<tr>
<th>Time</th>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Target System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process 1</td>
</tr>
<tr>
<td>Process 2</td>
</tr>
</tbody>
</table>
Difficulties of Distribution

➢ Sequence of Chunks
➢ Context
D-TIME

Offline Phase

Online Phase
D-TIME – Offline Phase

... ;

cmp [a], [b] ; if(a==b) {

jne a_unequal_b ;

mov [a], 0 ; a = 0

a_unequal_b:

} ;

mov [c], [d] ; c = d

... ;

Splitting technique adopted from MalWASH (WOOT’16)
D-TIME – Offline Phase

- Chunk 1:
  ```
  cmp [a], [b] ; if(a==b){
  jne a_unequal_b ;
  ```

- Chunk 2:
  ```
  mov [a], 0 ; a = 0
  ```

- Chunk 3:
  ```
  mov [c], [d] ; c = d
  ...
  ```

Splitting technique adopted from MalWASH (WOOT'16)
D-TIME – Offline Phase

chunk 1

... ;
cmp [a], [b] ; if(a==b){
jne a_unequal_b ;
mov ebx, 2
jmp END
jmp a_unequal_b:
mov ebx, 3
chunk 2

mov [a], 0 ; a = 0
mov ebx, 3
chunk 3

mov [c], [d] ; c = d
...

Splitting technique adopted from MalWASH (WOOT’16)
<table>
<thead>
<tr>
<th>Emulator</th>
<th>Inter-emulator Comm. Channels</th>
<th>SCBC</th>
</tr>
</thead>
</table>

D-TIME – Online Phase
D-TIME – Online Phase

Emulator

Inter-emulator Comm. Channels

Virtualization Layer - Emulator

OS

SCBC

Time

T1 T2 T3 T4 T5 T6

Process 1 Process 2 Process 3

13th USENIX Workshop on Offensive Technologies
## D-TIME – Online Phase

<table>
<thead>
<tr>
<th>Emulator</th>
<th>Inter-emulator Comm. Channels</th>
<th>SCBC</th>
</tr>
</thead>
<tbody>
<tr>
<td>➢ A Function</td>
<td>➢ Gets executed in benign thread’s context</td>
<td>➢ Creates the virtualization layer</td>
</tr>
<tr>
<td>➢ Execute a chunk</td>
<td>➢ Maintaining</td>
<td>➢ Ensures completeness of malware</td>
</tr>
<tr>
<td>✓ Sequence of chunks</td>
<td>✓ Continuity of state (stack, registers, heap)</td>
<td></td>
</tr>
</tbody>
</table>
APC
(Asynchronous Procedural Call)
APC
(Asynchronous Procedural Call)
APC
(Asynchronous Procedural Call)

Thread 1

Function 1

Alertable Wait State

Function 2

APC Queue

SleepEx
SignalObjectAndWait
MsgWaitForMultipleObjectsEx
WaitForMultipleObjectsEx
WaitForSingleObjectEx
## D-TIME

### Emulator

#### Steps:

1. Connect to Comm. Channel
2. Retrieve
   - a) State info
   - b) Next chunk to execute
3. Execute next chunk
4. Broadcast via Comm. Channel
   - a) Current state
   - b) Next chunk Id
5. Regenerate itself

### Inter-emulator Comm. Channels

**Online Phase:** A series of emulator executions,

**Emulators**

- Spawn in different threads
- Executing one chunk each
- Collectively completing the sequence.
D-TIME

Emulator Regeneration

Target System
D-TIME

Emulator Regeneration

Target System
D-TIME

Emulator Regeneration

Target System
D-TIME

Inter-emulator Comm. Channels

- Emulator
- SCBC

Process 1
- Thread 1
- Thread 2
- Secondary Channel (Heap Memory)

Process 2
- Thread 3
- Thread 4
- Secondary Channel (Heap Memory)

Primary Channel (Shared Memory)
D-TIME

Inter-emulator Comm. Channels

- Emulator
  - Process 1
    - Thread 1
    - Thread 2
    - Secondary Channel (Heap Memory)
  - Process 2
    - Thread 3
    - Thread 4
    - Secondary Channel (Heap Memory)
  - Process 3
    - Thread 5
    - Thread 6
    - Secondary Channel (Heap Memory)
- SCBC
  - Primary Channel 1 (Shared Memory)
  - Primary Channel 2 (Shared Memory)

Partially adopted from MalWASH (WOOT’16)
D-TIME

Emulator

Inter-emulator Comm. Channels

Heap and Shared Memory

1. Commonly used → Hard to classify as malign

2. Once allocated/attached, lives until
   a) De-allocated by application or
   b) Process exit
D-TIME

Emulator

Inter-emulator Comm. Channels

SCBC

Unknown location!

1. First emulator comes up
   a) Attaches the shared memory
   b) Executes a chunk
   c) Exit

2. The second emulator comes up
   ✓ Has access rights to shared memory
   ? Doesn’t know the starting address
SCBC
(Semaphore based Covert Broadcasting Channel)
SCBC
(Semaphore based Covert Broadcasting Channel)

Sender

Semaphore

Counter: 0xC0DE

Creates a semaphore and set semaphore-counter = data
SCBC
(Semaphore based Covert Broadcasting Channel)
SCBC
(Semaphore based Covert Broadcasting Channel)
SCBC
(Semaphore based Covert Broadcasting Channel)

Works in Linux as well as Windows
SCBC
(Semaphore based Covert Broadcasting Channel)

➢ High Integrity
➢ Broadcasting
➢ Persistent
➢ As Convenient as a shared integer

Converts semaphore to a shared integer
D-TIME Execution

Process 1

Thread 1

APC 8

Chunk 7

APC 5

Chunk 5

APC 4

Chunk 4

Heap Memory

Process 2

Thread 3

APC 6

Chunk 6

APC 3

Chunk 3

APC 2

Chunk 2

Heap Memory

Actual Malware

Chunk 7
Chunk 6
Chunk 5
Chunk 4
Chunk 3
Chunk 2
Chunk 1

Shared Memory
D-TIME - Summary

➢ Evades Detection by Distribution
➢ Performs Threadless execution
➢ Independent of Victim Process
➢ Completely Decentralized System
➢ High resilience
MalWASH – Phase 2

Process 1
T1 T2 T3

Process 2
T4 T5

Process 3
T6

Process 4
T7 T8
MalWASH – Phase 2

Creates threads in benign processes
MalWASH – Phase 2

Created threads are well exposed!
### MalWASH vs D-TIME

<table>
<thead>
<tr>
<th></th>
<th>MalWASH</th>
<th>D-TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Needs to Create Threads</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Has Dedicated Threads</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Needs Administrative Privilege</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Resilience</td>
<td>-</td>
<td>IMPROVED</td>
</tr>
<tr>
<td>Communication mechanism</td>
<td>-</td>
<td>IMPROVED</td>
</tr>
</tbody>
</table>
Evaluation & Results

- Key-logger Offline
- Key-logger Online
- Backdoor
- Ransomware
- Screenshot malware
- Window Monitor
Evaluation & Results

1. BitDefender
2. Norton
3. Kaspersky
4. WEBROOT
5. McAfee
6. ESET
7. Avast
8. AVG
9. Windows Defender
10. Avira
Evaluation & Results

CPU usage vs Time
Evaluation & Results

CPU usage for different number of infected processes

<table>
<thead>
<tr>
<th>Malware</th>
<th>1 process infected</th>
<th>2 processes infected</th>
<th>3 processes infected</th>
<th>4 processes infected</th>
<th>5 processes infected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offline-keylogger</td>
<td>60</td>
<td>40</td>
<td>20</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>Remote-keylogger</td>
<td>70</td>
<td>50</td>
<td>30</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>Screenshoter</td>
<td>80</td>
<td>60</td>
<td>40</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>Window-monitor</td>
<td>90</td>
<td>70</td>
<td>50</td>
<td>40</td>
<td>30</td>
</tr>
</tbody>
</table>
Evaluation & Results

Performance Counters

![Graph showing performance counters over time](image-url)
Countermeasures

**Prevention**
- Redesign APC
- Create QPC (Queue Privilege Class)
- Limit Recursive Queuing
- Principle of minimum disclosure

**Detection**
- Detecting Comm. Channels
- Analyzing
  - CPU Usage
  - Context switches

**Removal**
- Removing all Emulators
- Destroying all Comm. Channels
Contributions

➢ D-TIME: Distributed, Decentralized, threadless malware execution framework

➢ SCBC (Semaphore based Covert Broadcasting Channel)

➢ Tested D-TIME using 6 Malware against 10 anti-virus

➢ Suggested counter measures
Thank you