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, ...)

⚠️ Alert symbol
D-TIME
Distributed Threadless Independent Malware Execution
D-TIME

Chunks (set of instructions)

Split
Difficulties of Distribution

➢ Sequence of Chunks
➢ Context
D-TIME

Offline Phase

Online Phase
D-TIME – Offline Phase

... ;
cmp [a], [b] ; if(a==b){
jne jmp a_unequal_b ;
mov [a], 0 ; a = 0
jmp a_unequal_b: ; }
mov [c], [d] ; c = d
... ;

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

... ;
cmp [a], [b] ; if(a==b){
jne jmp a_unequal_b ;
}

mov [a], 0 ; a = 0

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

Chunk 1

Chunk 2

Chunk 3

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

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

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

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

SCBC

Virtualization Layer - Emulator

OS
D-TIME – Online Phase

- Emulator
  - Inter-emulator Comm. Channels
  - SCBC

- A Function
- Gets executed in benign thread’s context
- Creates the virtualization layer
- Execute a chunk
- Maintaining
  - Sequence of chunks
  - Continuity of state (stack, registers, heap)
- Ensures completeness of malware
APC
(Asynchronous Procedural Call)
APC
(Asynchronous Procedural Call)
APC
(Asynchronous Procedural Call)

Thread 1

Function 1

Function 2

APC Queue

Alertable Wait State

SleepEx
SignalObjectAndWait
MsgWaitForMultipleObjectsEx
WaitForMultipleObjectsEx
WaitForSingleObjectEx
### D-TIME

**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

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

- Spawns 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

- Primary Channel (Shared Memory)
- Secondary Channel (Heap Memory)

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

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

Inter-emulator Comm. Channels

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

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
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)

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)

Semaphore

Counter: 0xC0DE

0xC0DE

0xC0DE

0xC0DE

Receivers

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 - 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>Feature</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

![Graph showing CPU usage over time with distinct phases: Startup, Normal activities, and Idle time.]

- **Chrome**
- **Opera**
- **Calc**
Evaluation & Results

CPU usage for different number of infected processes

![Graph showing CPU usage for different number of infected processes with labels for malwares and different infection levels.]
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