Protean General Purpose Guard (PGPG):

Detecting and Mitigating Cache-based Microarchitectural Attacks Using Protean Code

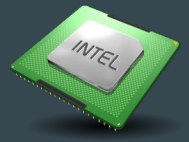
Jeremy Erickson
jericks@umich.edu

Akshitha Sriraman
akshitha@umich.edu

Sai Gouravajhala
sairohit@umich.edu

EECS 583: Advanced Compilers

Background



- **Flush+Reload Attack [Yarom14]**
  - Spy repeatedly loads an instruction in shared memory to infer secret key in GnuPG process
  - Time the load and flush
  - Timing differences are distinguishable

```
unsigned long probe(char *adrs) {
  volatile unsigned long time;
  asm __volatile__ (
      "  mfence         \n"
      "  lfence         \n"
      "  rdtsc          \n"
      "  lfence         \n"
      "  movl %%eax, %%esi   \n"
      "  movl (%1), %%eax    \n"
      "  lfence              \n"
      "  rdtsc               \n"
      "  subl %%esi, %%eax   \n"
      "  clflush 0(%1)       \n"
      : "=a" (time)
      : "c" (adrs)
      :  "%esi", "%edx");
  return time;
}
```

Time

Reload

Time

Flush

- **GnuPG + Microbenchmark**
  - For a given bit of the secret exponent, code branches help spy determine if bit is a ‘0’ or a ‘1’
  - Microbenchmark is proxy for GnuPG encryption routine

Vulnerable

```
for(int i = 0; i < 32; i++)
{
  lsb = exp & 1;
  exp = exp >> 1;
  val = square(val);
  val = reduce(val);
  if (lsb == 1)
  {
    val = mul(val);
    val = reduce(val);
  }
}
```

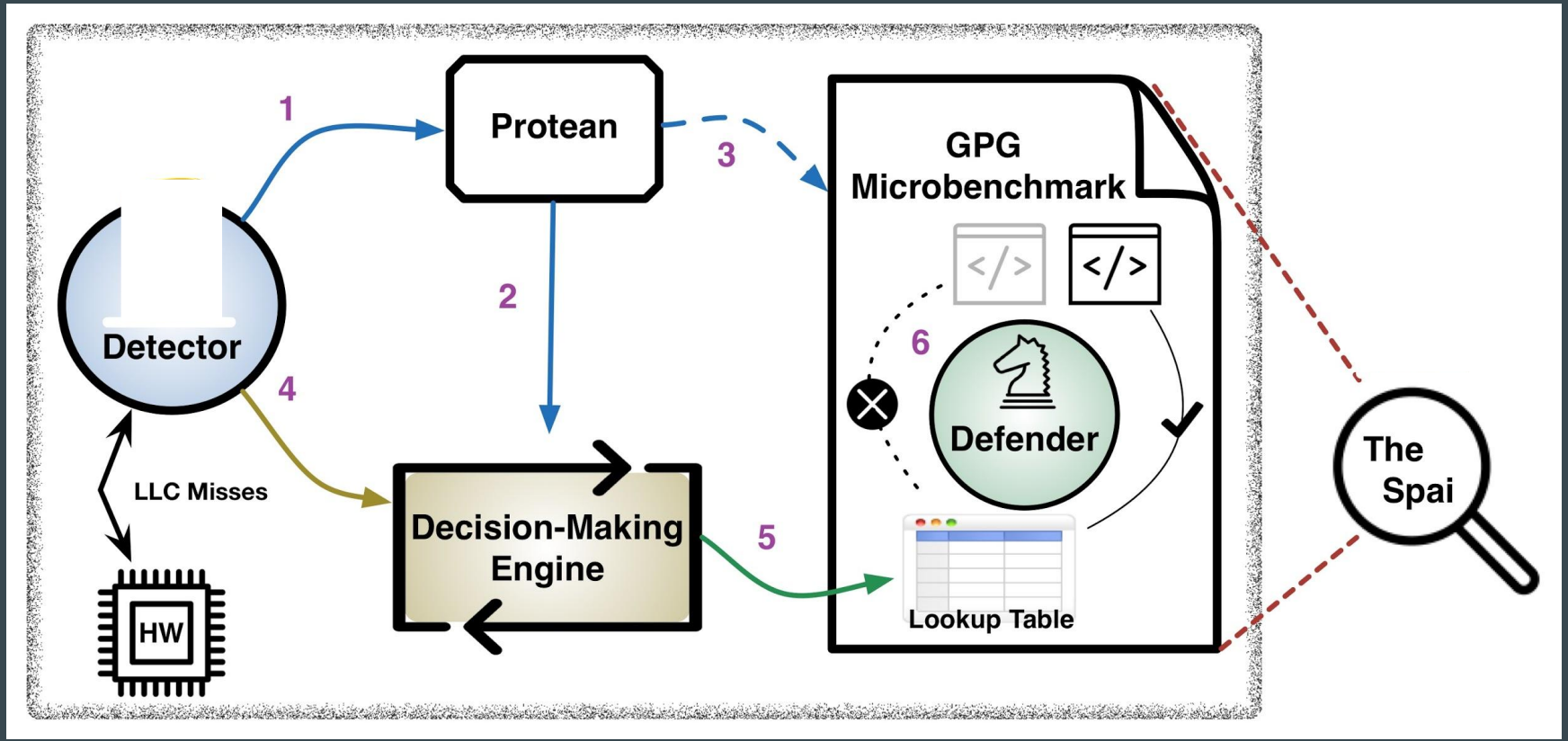
Safe

```
for(int i = 0; i < 32; i++)
{
  lsb = exp & 1;
  exp = exp >> 1;
  val = square(val);
  val = reduce(val);
  val2 = mul(val);
  val2 = reduce(val2);
  if (lsb == 1)
  {
    val = val2;
  }
}
```

Main Contributions

- First to develop a **system** that leverages dynamic compilation to **overcome the security-performance gap**
- **Extend the use of Protean code** to make modifications to program semantics, such as including a dynamic defense
- Develop and evaluate an **end-to-end implementation**: Detection and Mitigation
- **Reimplement the Flush+Reload attack** to test the attack detection and defense mechanism

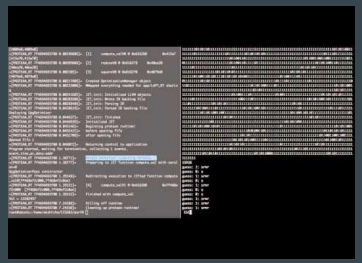
PGPG System Overview



Demo

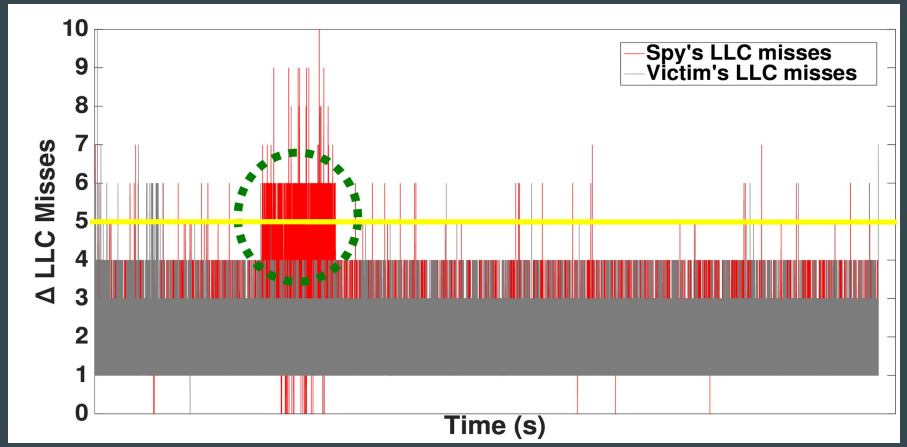
- Probe & Flush+Reload & Interpret.py

- Probe & Flush+Reload & PGPG & Interpret.py



Evaluation

- Attack detection:



- Execution time model:

$$TE = (1 - AP) * IE + (AP * SE)$$

TE = total execution time
AP = attack percentage
IE = insecure execution time
SE = secure execution time

- Average execution times:

	Vulnerable	Safe
Static	3.64 s	4.82 s
Protean	3.63 s	4.84 s

$$E[TE] = (0.8 * 3.63s) + (0.2 * 4.84s)$$
$$= 3.87s$$

19.7% speedup!

PGPG - NOW OPEN SOURCE!

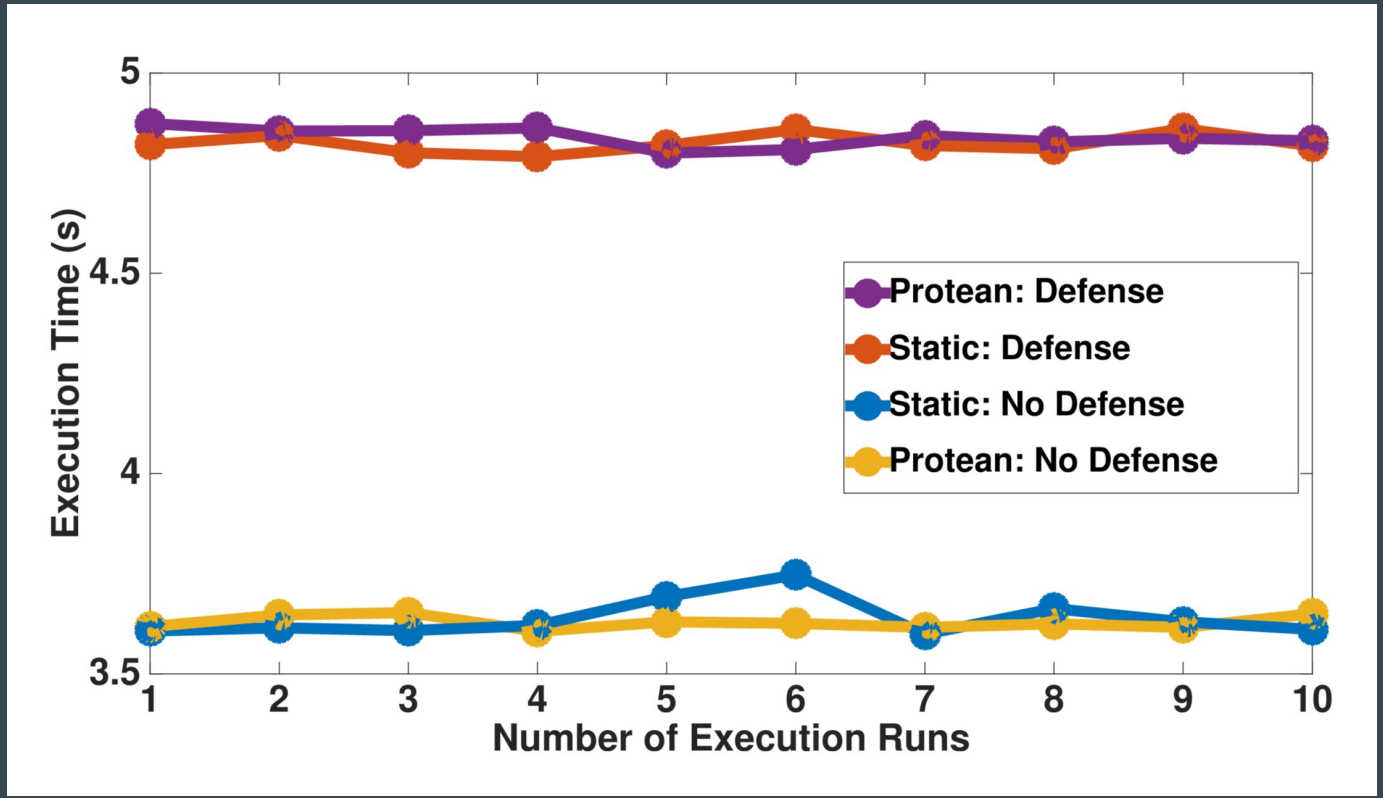
https://github.com/akshithasriraman/EECS583-Project.git

Download your copy today.

Backup Slides

...

Execution Times



GnuPG Code Vs. Microbenchmark (Vulnerable)

```
for(;;) {
    while( c ) {
  mpi_ptr_t tp;
  mpi_size_t xsize;

  /*mpihelp_mul_n(xp, rp, rp, rsize);*/
  if( rsize < KARATSUBA_THRESHOLD )
      mpih_sqr_n_basecase( xp, rp, rsize );
  else {
      if( !tspace ) {
    tsize = 2 * rsize;
    tspace = mpi_alloc_limb_space( tsize, 0 );
      }
      else if( tsize < (2*rsize) ) {
    mpi_free_limb_space( tspace );
    tsize = 2 * rsize;
    tspace = mpi_alloc_limb_space( tsize, 0 );
      }
      mpih_sqr_n( xp, rp, rsize, tspace );
  }

  xsize = 2 * rsize;
  if( xsize > msize ) {
      mpihelp_divrem(xp + msize, 0, xp, xsize, mp, msize);
      xsize = msize;
  }

  tp = rp; rp = xp; xp = tp;
  rsize = xsize;
```

Multiply

Square

Reduce

```
  if( (mpi_limb_signed_t)e < 0 ) {
      /*mpihelp_mul( xp, rp, rsize, bp, bsize );*/
      if( bsize < KARATSUBA_THRESHOLD ) {
    mpihelp_mul( xp, rp, rsize, bp, bsize );
      }
      else {
    mpihelp_mul_karatsuba_case(
          xp, rp, rsize, bp, bsize, &karactx );
      }

      xsize = rsize + bsize;
      if( xsize > msize ) {
    mpihelp_divrem(xp + msize, 0, xp, xsize, mp, msize);
    xsize = msize;
      }

      tp = rp; rp = xp; xp = tp;
      rsize = xsize;
  }
  e <<= 1;
  c--;
    }

    i--;
    if( i < 0 )
  break;
    e = ep[i];
    c = BITS_PER_MPI_LIMB;
}
```

Conditional

Reduce

```
for(int i = 0; i < 32; i++)
{
  lsb = exp & 1;
  exp = exp >> 1;
  val = square(val);
  val = reduce(val);
  if (lsb == 1)
  {
    val = mul(val);
    val = reduce(val);
  }
}
```

*GnuPG is licensed GPLv3

GnuPG Code Vs. Microbenchmark (Safe)

```
for(;;) {
    while( c ) {
  mpi_ptr_t tp;
  mpi_size_t xsize;

  /*mpihelp_mul_n(xp, rp, rp, rsize);*/
  if( rsize < KARATSUBA_THRESHOLD )
      mpih_sqr_n_basecase( xp, rp, rsize );
  else {
      if( !tspace ) {
    tsize = 2 * rsize;
    tspace = mpi_alloc_limb_space( tsize, 0 );
      }
      else if( tsize < (2*rsize) ) {
    mpi_free_limb_space( tspace );
    tsize = 2 * rsize;
    tspace = mpi_alloc_limb_space( tsize, 0 );
      }
      mpih_sqr_n( xp, rp, rsize, tspace );
  }

  xsize = 2 * rsize;
  if( xsize > msize ) {
      mpihelp_divrem(xp + msize, 0, xp, xsize, mp, msize);
      xsize = msize;
  }

  tp = rp; rp = xp; xp = tp;
  rsize = xsize;
```

Conditional Changed

```
            /* To mitigate the Yarom/Falkner flush+reload cache
             * side-channel attack on the RSA secret exponent, we
             * do the multiplication regardless of the value of
             * the high-bit of E.  But to avoid this performance
             * penalty we do it only if the exponent has been
             * stored in secure memory and we can thus assume it
             * is a secret exponent.  */
            if (esec || (mpi_limb_signed_t)e < 0) {
      /*mpihelp_mul( xp, rp, rsize, bp, bsize );*/
      if( bsize < KARATSUBA_THRESHOLD ) {
    mpihelp_mul( xp, rp, rsize, bp, bsize );
      }
      else {
    mpihelp_mul_karatsuba_case(
          xp, rp, rsize, bp, bsize, &karactx );
      }

      xsize = rsize + bsize;
      if( xsize > msize ) {
    mpihelp_divrem(xp + msize, 0, xp, xsize, mp, msize);
    xsize = msize;
      }
            }
  if ((mpi_limb_signed_t)e < 0) {
      tp = rp; rp = xp; xp = tp;
      rsize = xsize;
  }
  e <<= 1;
  c--;
    }

    i--;
    if( i < 0 )
  break;
    e = ep[i];
    c = BITS_PER_MPI_LIMB;
}
```

New Conditional

```
for(int i = 0; i < 32; i++)
{
  lsb = exp & 1;
  exp = exp >> 1;
  val = square(val);
  val = reduce(val);
  val2 = mul(val);
  val2 = reduce(val2);
  if (lsb == 1)
  {
    val = val2;
  }
}
```

*GnuPG is licensed GPLv3

Self-Evaluation

Akshitha Sriraman
* Hardware event counter code
* Attack detection algorithm
* GnuPG Microbenchmark
* Paper writing
* Slide production

Sai Gouravajhala
* Defender (Protean) code
* GnuPG Microbenchmark
* Paper writing
* Slide production

Jeremy Erickson
* Reimplementation of the Flush+Reload attack
* Defender (Protean) code
* GnuPG Microbenchmark
* Paper writing
* Slide production

These bullets are a rough outline of what each group member produced, but all group members participated in regular project discussion (several times per week) and helped develop workarounds to problems and original, failed approaches (not listed).